Deriving Custom Post Types from Digital Mockups

Alfonso Murolo(✉) and Moira C. Norrie

Department of Computer Science, ETH Zurich, 8092 Zurich, Switzerland
{alfonso.murolo,norrie}@inf.ethz.ch

Abstract. Interface-driven approaches to web development often migrate digital mockups defining the presentation, structure and client-side functionality of a website to platforms such as WordPress that manage the content of the website and implement server-side functionality. In the case of data-intensive websites, generation of data types that manage the application-specific content is usually performed manually during the migration process. We propose an approach that allows WordPress custom post types to be derived based on an analysis of sample content used in digital mockups.

Keywords: Digital mockups · Data schemas · Custom post types

1 Introduction

It is common practice to develop websites using an interface-driven approach which starts with mockups of a website and, through a series of steps, adds first client-side and then server-side functionality. In the case of data-intensive websites, at some stage, data types need to be defined and data content added. Mockups provide a basis for communicating with the client to establish agreement not only on the visual appearance of their website, but also the required content and functionality. They frequently start as sketches on paper which are evolved into digital mockups implemented using HTML, CSS and JavaScript.

While filler text such as *lorem ipsum* is often used in mockups to represent content, some developers stress the importance of using real examples of content. Reasons for this include avoiding possible breaks in the design when real content is loaded as well as enhancing discussions with clients about the content to be managed and how it should be displayed. Moreover, it is reported in [1] that clients sometimes appear to feel more in control of the design process when they are presented with real samples of content and more likely to give feedback on the way that content is shown rather than purely the look and feel of the design.

Given estimates that nearly a quarter of the top ten million websites are running on WordPress[1], adding server-side functionality often involves transforming a digital mockup into a WordPress theme. As with other content management

[1] 23.7% according to w3techs.com on 8 Apr 2015.

P. Cimiano et al. (Eds.): ICWE 2015, LNCS 9114, pp. 71–80, 2015.
DOI: 10.1007/978-3-319-19890-3_6

systems (CMS), WordPress provides a generic schema for managing content which, in the case of WordPress, is based on posts and pages. This data schema can be extended with custom post types to manage application-specific data. A number of plugins are available to help users define their custom post types and integrate support into the administrative dashboard for creating and managing the associated data and, in some cases, data dependencies. Researchers have shown how developers could be further assisted by providing a meta-plugin that allows them to define their data schema in terms of an entity-relationship (ER) model and using this to automatically generate bespoke plugins for managing their data [2]. However, as reported in a survey of modern web development practices carried out in 2014 [3], many WordPress developers have no formal education in computer science and are unfamiliar with ER models.

We propose an approach where data types can be generated from sample content used in mockups to further simplify the process of developing data-intensive sites. The process consists of two steps: first generating a conceptual model of data entities based on an analysis of sample content and, second, creating an implementation for that model as WordPress custom post types. The first part is semi-automatic in that users annotate parts of the content and then guide the generation process which is based on automatic matching and clustering techniques. The generation of WordPress custom post types also requires code to be generated for every layer, including the server-side code capable of storing data in the database as well as the GUI to input and edit data. The user can also choose to populate the database with the extracted sample data.

We start with a review of related work in Sect. 2. We then present an overview of the approach in Sect. 3 before outlining our content matching algorithm and process for generating custom post types in Sect. 4. Concluding remarks are given in Sect. 5.

2 Background

Although mockups are widely used in practice, relatively little research has investigated how paper or digital mockups of websites could be used to automate parts of the development process. Within the HCI community, DENIM was an early project that generated simple versions of a website from sketches of pages and storyboards [4]. More recently, some researchers within the web engineering community have proposed tools to automatically generate APIs (MockAPI) [5] and application prototypes (MockDD) [6] from digital mockups in the form of wireframes. In both cases, users annotate the mockups to specify data entities and operations. In the case of MockDD, either a WebML [7] or UWE model [8] is generated and the existing tools associated with these models can then be used to generate the code for the website. In this way, they combine interface-driven and model-driven approaches within the overall development process.

An approach commonly aimed for in practice is the use of a visual editor to create a high-fidelity mockup of a website from which the code can be generated automatically. For example, in the case of WordPress, a number of theme generators are available that allow a user to design their website using a graphical tool and then generate the HTML, CSS, JavaScript and PHP files that define the WordPress site. However, existing theme generators are often restricted in terms of the flexibility and functionality that they offer to users. For example, Templatr[2] offers a fixed set of layouts while Lubith[3] allows users to customise layout but not functionality. Further, they do not provide specific support for data-intensive websites where application-specific data has to be managed.

Our approach is based on high-fidelity mockups that detail not only visual features and functionality of a website but also content. By analysing the samples of real content provided in the mockup, we aim to generate the data schemas and code necessary to create and manage the associated data with a minimum amount of interaction from the user.

The problem of generating data schemas from sample content is closely related to previous work on tools to extract data from web pages - the so-called *deep web*. In this research area, the goal of numerous projects is to generate wrappers that enable data published on dynamically-generated web pages to be extracted and/or queried. Generally, a wrapper uses a set of extraction rules to perform pattern matching over a page. Various approaches exist for generating wrappers—a problem known as *wrapper induction*—and these can be classified according to three main characteristics: the difficulty of the task, the techniques used and the degree of automation [9].

Works such as *NoDoSE* [10], *IEPAD* [11], *DeLa* [12] and *RoadRunner* [13] aim at generating wrappers based on a semi- or fully-automatic analysis of DOM structures and models derived from them, taking one or more pages as input and trying to discover repeating patterns in these pages through regular expressions and clustering. However, these approaches have various drawbacks such as requiring large amounts of data as input, sometimes consisting of groups of pages from the same website, or extensive amounts of user interaction.

The work by Lu et al. [14,15] distinguishes itself from the ones previously mentioned because it proposes a system for aligning and annotating similarly structured data through clustering given a set of data records obtained from queries to website. The main difference here is that they use forms to perform queries on websites for which they want to generate a wrapper, and therefore detect similarities between the query results. This is one of the closest works to ours, however, the main difference is that their goal is to create an annotation wrapper to be used on similar pages of the same website through form inputs, while ours is purely to locate and extract data records within single web pages for use in the development process.

Other works such as ViWER [16] and ViDE [17] propose the use of visual cues to detect data records. An example of visual cues can be the size of bounding boxes of the data records or block trees which segment regions of a page to isolate data records. While these works demonstrate the significance of visual cues, their techniques do not scale well if data records can have an increasng amount of small differences in lower levels of the subtree local to each record, or with the

[2] http://templatr.cc
[3] http://www.lubith.com

increasing amount of data-rich visual block trees. However, we acknowledge the importance of visual cues in the detection process and also use them in our approach.

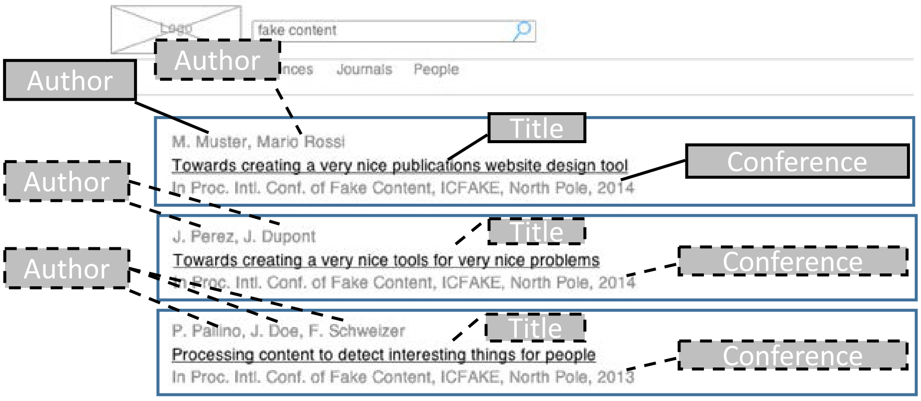
In summary, while we could build on many ideas from previous research on wrapper induction, none of these methods fully meets our requirements. Further, while their primary goal is to perform content matching to be able to query data published in web pages, our target is to derive custom post types for managing such data and its implementation in WordPress. In the next section, we introduce the approach that we have developed before explaining the techniques we use.

3 Approach

Our tool implements a semi-automated process which requires users to annotate parts of the sample data content in a digital mockup. For example, assume a user is developing a website for a research group where one of the pages will list publications. In the digital mockup, a real sample of content would be provided, as shown in Fig. 1. A user can then annotate parts of that data by selecting an element with the mouse and labelling it. For example, for the first publication in the list, they might label the first author, the title and the conference as denoted by labels with a solid border.

After labelling, the user invokes the matching process and the tool searches for similar examples, propagating the labelling to all similar data items found. The system generated labels are shown in Fig. 1 with a dashed border. Note that, as part of this matching process, fields which have more than one occurrence (e.g. *author* in a publication) are also detected and labelled.

The overall process involves performing incremental matching from different parts of the digital mockup until all sample content of application-specific data has been labelled. Once this has been done, conceptual data types for the

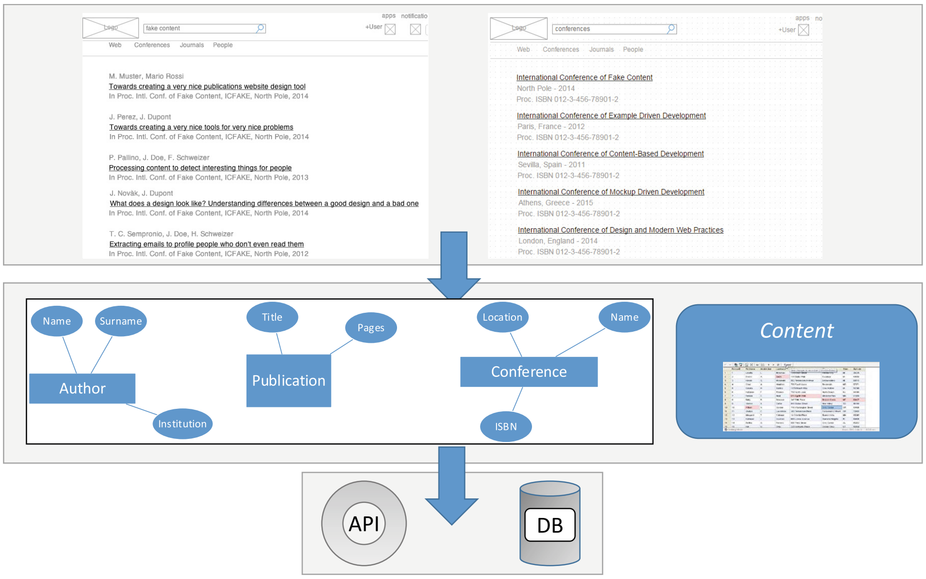


M. Muster, Mario Rossi
Towards creating a very nice publications website design tool
In Proc. Intl. Conf. of Fake Content, ICFAKE, North Pole, 2014

J. Perez, J. Dupont
Towards creating a very nice tools for very nice problems
In Proc. Intl. Conf. of Fake Content, ICFAKE, North Pole, 2014

P. Pallino, J. Doe, F. Schweizer
Processing content to detect interesting things for people
In Proc. Intl. Conf. of Fake Content, ICFAKE, North Pole, 2013

Fig. 1. Matching sample data in an HTML mockup



M. Muster, Mario Rossi
Towards creating a very nice publications website design tool
In Proc. Intl. Conf. of Fake Content, ICFAKE, North Pole, 2014

J. Perez, J. Dupont
Towards creating a very nice tools for very nice problems
In Proc. Intl. Conf. of Fake Content, ICFAKE, North Pole, 2014

P. Pallino, J. Doe, F. Schweizer
Processing content to detect interesting things for people
In Proc. Intl. Conf. of Fake Content, ICFAKE, North Pole, 2013

J. Novák, J. Dupont
What does a design look like? Understanding differences between a good design and a bad one
In Proc. Intl. Conf. of Fake Content, ICFAKE, North Pole, 2014

T. C. Sempronio, J. Doe, H. Schweizer
Extracting emails to profile people who don't even read them
In Proc. Intl. Conf. of Fake Content, ICFAKE, North Pole, 2012

International Conference of Fake Content
North Pole - 2014
Proc. ISBN 012-3-456-78901-2

International Conference of Example Driven Development
Paris, France - 2012
Proc. ISBN 012-3-456-78901-2

International Conference of Content-Based Development
Sevilla, Spain - 2011
Proc. ISBN 012-3-456-78901-2

International Conference of Mockup Driven Development
Athens, Greece - 2015
Proc. ISBN 012-3-456-78901-2

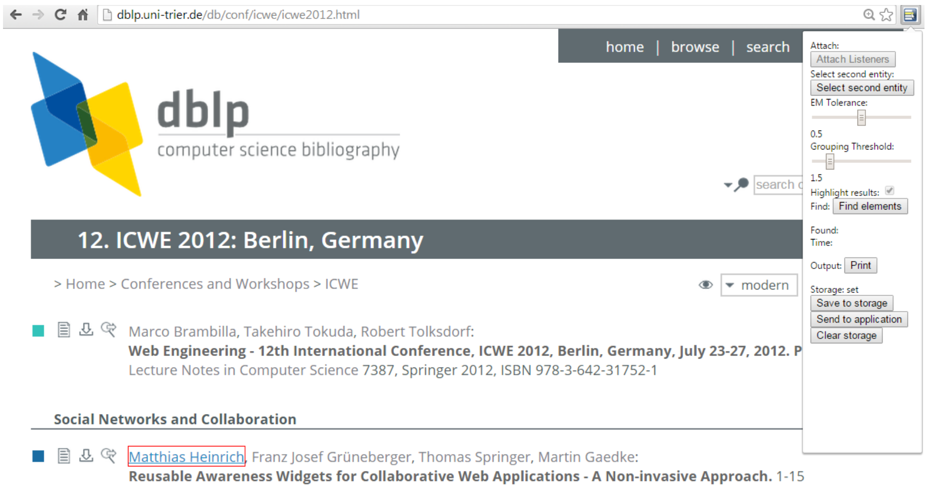
International Conference of Design and Modern Web Practices
London, England - 2014
Proc. ISBN 012-3-456-78901-2

Fig. 2. Derivation of WordPress custom post types from sample data content

website are generated as indicated in Fig. 2. In a second step, the tool generates an implementation of these data types as WordPress custom post types. This involves the generation of an API, which allows basic CRUD operations, together with the required elements of the user interface required to allow users to perform data management. We note that WordPress custom post types are simpler than relational data schemas, since there is no support for relationships. However, we are currently investigating techniques for detecting relationships between entities and implementing them in WordPress in order to be able to handle more general database schemas in the future.

The results of the matching process may not always work as well as the example shown in Fig. 1 on the first attempt. For this reason, the user is offered a control panel as shown in Fig. 3 where they can experiment with various settings until they get the desired result (e.g. adjustable tolerance levels to potentially enlarge the set of matched records at the cost of increasing false positives). We explain this using a second example where the sample content comes from an existing website rather than a mockup. Since there is no real distinction technically between a digital mockup of a website and an actual website, the same process can be applied to examples of existing websites that meet the criteria of parts of the website under development, thereby supporting a design-by-example paradigm. For instance, the DBLP website[4] could be used to provide an example of a site with a list of publications and it would even be possible to extract the data to populate the database of the website under development.

[4] http://dblp.uni-trier.de/


dblp.uni-trier.de/db/conf/icwe/icwe2012.html

home | browse | search

dblp computer science bibliography

Attach: Attach Listeners
Select second entity: Select second entity
EM Tolerance: 0.5
Grouping Threshold: 1.5
Highlight results: ✓
Find: Find elements
Found:
Time:
Output: Print
Storage: set
Save to storage
Send to application
Clear storage

12. ICWE 2012: Berlin, Germany

> Home > Conferences and Workshops > ICWE

modern

Marco Brambilla, Takehiro Tokuda, Robert Tolksdorf:
Web Engineering - 12th International Conference, ICWE 2012, Berlin, Germany, July 23-27, 2012. P
Lecture Notes in Computer Science 7387, Springer 2012, ISBN 978-3-642-31752-1

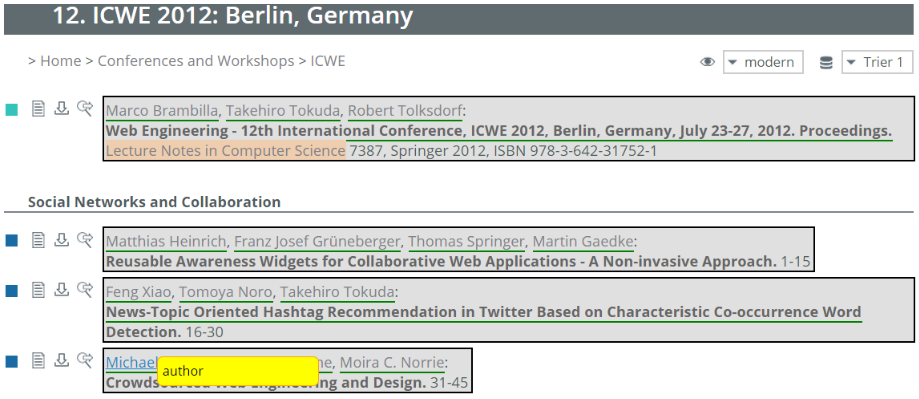
Social Networks and Collaboration

Matthias Heinrich, Franz Josef Grüneberger, Thomas Springer, Martin Gaedke:
Reusable Awareness Widgets for Collaborative Web Applications - A Non-invasive Approach. 1-15

Fig. 3. DBLP page with the content matcher user panel on the right

In such a use case, the user would load the page and start annotating elements as shown in Fig. 3. The elements that can be annotated are highlighted as the user moves the cursor over them. The users might select and label a single author as indicated in Fig. 3. Assume they then also label a single title and a page reference in proceedings.

The user can then start the matching process. In this example, although the user annotated only the first author, all the authors are correctly matched as similar and labelled by the system. Fig. 4 shows a screenshot after the matching process. Each publication is recognised as a data unit as indicated by the shading


12. ICWE 2012: Berlin, Germany

> Home > Conferences and Workshops > ICWE

modern Trier 1

Marco Brambilla, Takehiro Tokuda, Robert Tolksdorf:
Web Engineering - 12th International Conference, ICWE 2012, Berlin, Germany, July 23-27, 2012. Proceedings.
Lecture Notes in Computer Science 7387, Springer 2012, ISBN 978-3-642-31752-1

Social Networks and Collaboration

Matthias Heinrich, Franz Josef Grüneberger, Thomas Springer, Martin Gaedke:
Reusable Awareness Widgets for Collaborative Web Applications - A Non-invasive Approach. 1-15

Feng Xiao, Tomoya Noro, Takehiro Tokuda:
News-Topic Oriented Hashtag Recommendation in Twitter Based on Characteristic Co-occurrence Word Detection. 16-30

Michael ne, Moira C. Norrie:
Crowds ng and Design. 31-45

Fig. 4. Content matching results

and the elements underlined are the ones that have been labelled either by the user or the system. We can see therefore that all authors have been individually labelled. The user can inspect the labels by moving the cursor over the elements.

However, there are parts of the data that are not underlined which means that the system has successfully distinguished these as different from the labelled elements. For example, it recognises that *Lecture Notes in Computer Science* is neither an author nor a title based on various factors such as the form, the position and the presentation of the element, but has no way of determining what kind of entity it is. When the user detects errors in the identification of elements and their labelling, as well as incomplete coverage, they can then refine the process through a combination of annotating additional elements and experimenting with settings in the control panel that will be explained in the next section when we describe the matching algorithm.

All of the matched content, together with its detected structure, is sent to a WordPress plug-in that we have developed. This plug-in creates the custom post types along with the corresponding UI and server-side PHP code to manage the default CRUD operations. Once the generation has been completed, the generated PHP files can be embedded directly into a WordPress theme enabling the corresponding data to be created and managed through the administrative dashboard.

4 Algorithm

As described in the previous section, the matching process requires the user to first label individual fields of a data record. To minimise the demand on users, our goal was that they should only have to label parts of a sample data record such as the first occurrence of a repeated field, for example the first author in the list of authors of a publication, before starting the matching process.

The matching process is divided into various steps and it makes use of both structural and visual cues in several of the steps. We will now explain each step in turn.

1. **Record boundaries detection.** The preliminary phase starts by detecting the boundaries of the annotated data record. Once the labelled fields are stored, the matcher starts to look for a least common ancestor (LCA) for these fields. A least common ancestor is the closest node in the hierarchy that is an ancestor for all of the elements labelled by the user. Our algorithm performs best when each data record has a different LCA. However, this cannot be guaranteed in every possible case. There can be template-generated pages that will not necessarily have a unique LCA for every data record. We will provide details on how our algorithm behaves in such cases later in this section.
2. **Finding similar records.** Once the boundaries of the first record have been found, the algorithm has identified a DOM subtree similar to that of similar data records in the page, modulo some differences that will be record-specific. This phase starts retrieving all the elements in the page which have the same

HTML tag as the LCA, and compares their subtrees with the subtree with the LCA as root. To obtain a measure of the difference between two subtrees, we use an approximation algorithm called pq-gram distance [18], which calculates the tree edit distance problem efficiently. For each subtree, we calculate the pq-gram distance and compare it against a threshold called the *tolerance factor*. If the distance is less than this factor, we consider this subtree a possible match. At this stage, completely different elements which have a structure similar enough to be matched by the approximation algorithm would be part of the set of matched subtrees as false positives, so we next need to try and exclude them.

3. **Cross-record propagation.** We now propagate the labels specified by the user across all the matched records. We have to make sure we can replicate the user's selection in each subtree through XPath-based relative paths. If we are unable to replicate the selection, we remove the current subtree from the matched collection. On the contrary, if we can manage to replicate all of the user's selections in the current subtree, we keep it as a true positive.
4. **Local label propagation.** At this stage, each of the subtrees which has been recognised as similar contains the labelling made by the user and can be considered as a data record. We now aim at replicating the labels to all the elements which have a similar meaning, for example all authors of a publication within an identified publication data record. To do so, we use agglomerative hierarchical clustering. For each data record, we group siblings of the labelled elements according to a *distance function*, and use a complete linkage criterion between the clusters. As a stopping criterion for the hierarchical clustering, we check the distance between the clusters against a threshold, called the *grouping factor*. The distance function $dist(x, y)$ takes into consideration various factors which can be both visual and structural cues, namely:
   - The tag equality ϕ. Let t_1 and t_2 be the tags of the elements x and y, respectively. The tag equality is 1 if $t_1 == t_2$; otherwise, it is 0.
   - The structural tag discontinuity score Δ_t, which increases as the elements are further away, separated by elements of a different tag.
   - The field discontinuity score Δ_f, which increases as the elements between those being compared have been labelled as different fields.
   - The *style distance ratio* Δ_s as a measure of the distance between x and y in terms of their visual cues. It is defined as the following ratio:

     $$\Delta_s = \frac{\Delta_{css}}{max(\Delta_{css} + S, 1)}$$

     where Δ_{css} is the number of CSS rules which differ between the two elements being considered, and S is the number of CSS rules which are similar.

   Then, the distance function is calculated as follows:

   $$dist(x, y) := \Delta_s + \phi(t_1, t_2) + max(\Delta_t, \Delta_f)$$

   Since this distance function is used in hierarchical clustering, the elements which are considered to be very similar in structural and visual terms, and which happen to be presented in a contiguous fashion (e.g. all the links to the authors may be presented one after the other) will be put into the same cluster. If the elements are instead discontiguous, or are elements of a different type (e.g. ANCHOR or SPAN), the distance will be increased. We now need to decide which label should be applied to a cluster and this is done by majority voting. Each element in the cluster will have the chance to be counted as a vote for the corresponding label, if it has one. However, if it does not have one, the element will not be considered in the vote.

It could happen that the elements in the page appear to be visually separated in the browser, but have an LCA in common with all of the data records. The above algorithm would fail to identify a unique LCA for a data record, and the whole procedure might fail to correctly identify the other data records. In such a case, we need the user to specify a second example data record. This would allow the system to detect that the LCA for the two data records is the same element, and therefore they belong to the same subtree. However, the matching algorithm does more than just fault detection. We have developed a pattern matcher which behaves as a regular expression matcher for DOM elements. We can create a regular expression which, based on the user's selection, tries to detect the boundaries of a data record within the siblings. For example, we can create a regular expression which can detect a sequence of "one or more A elements followed by one or more SPAN elements". Once the boundaries of each data record in the page have been detected, we can artificially modify the DOM and create an element which will act as an LCA for each of them.

5 Conclusion

We have presented a method for deriving custom post types from digital mockups of websites with real samples of content. Although our primary aim was to support interface-driven development, the method also supports a design-by-example paradigm where users can base their design on parts of existing web sitse with similar data content.

We are currently investigating generalisations of the approach to support more powerful schema paradigms, such as relational schemas based on entity-relationship models. This involves the detection of more complex schema structures that involve relationships or aggregations, together with a collection mechanism which can incrementally match multiple data types and can be exploited to infer relationships. Additionally, we want to consider even more hybrid approaches in terms of structural and visual cues that can be considered while performing element clustering. We also plan to extend support for HTML text nodes which can be a challenge when detecting data records in HTML mockups or websites. Alongside these extensions and enhancements of the current method, we want to generalise the architecture of our tool to support the generation of data schemas and server-side APIs for target platforms other than WordPress so that it could be applied more generally to web application development.

References

1. Blakeley-Silver, T.: WordPress 2.8 Theme Design: Create Flexible, Powerful, and Professional Themes for Your WordPress Blogs and Websites. Packt Publishing Ltd. (2009)
2. Leone, S., de Spindler, A., Norrie, M.C.: A meta-plugin for bespoke data management in wordpress. In: Wang, X.S., Cruz, I., Delis, A., Huang, G. (eds.) WISE 2012. LNCS, vol. 7651, pp. 580–593. Springer, Heidelberg (2012)
3. Norrie, M.C., Di Geronimo, L., Murolo, A., Nebeling, M.: The forgotten many? a survey of modern web development practices. In: Casteleyn, S., Rossi, G., Winckler, M. (eds.) ICWE 2014. LNCS, vol. 8541, pp. 290–307. Springer, Heidelberg (2014)
4. Newman, M.W., Lin, J., Hong, J.I., Landay, J.A.: DENIM: An Informal Web Site Design Tool inspired by Observations of Practice. Human-Computer Interaction **18**(3) (2003)
5. Rivero, J.M., Heil, S., Grigera, J., Gaedke, M., Rossi, G.: MockAPI: an agile approach supporting API-first web application development. In: Daniel, F., Dolog, P., Li, Q. (eds.) ICWE 2013. LNCS, vol. 7977, pp. 7–21. Springer, Heidelberg (2013)
6. Rivero, J.M., Grigera, J., Rossi, G., Luna, E.R., Montero, F., Gaedke, M.: Mockup-Driven Development: Providing Agile Support for Model-Driven Web Engineering. Information and Software Technology **56**(6) (2014)
7. Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M.: Designing Data-Intensive Web Applications. Morgan Kaufmann (2002)
8. Hennicker, R., Koch, N.: A UML-Based methodology for hypermedia design. In: Evans, A., Caskurlu, B., Selic, B. (eds.) UML 2000. LNCS, vol. 1939, pp. 410–424. Springer, Heidelberg (2000)
9. Chang, C., Kayed, M., Girgis, M.R., Shaalan, K.F.: A Survey of Web Information Extraction Systems. IEEE Transactions on Knowledge and Data Engineering **18**(10) (2006)
10. Adelberg, B.: NoDoSE a tool for semi-automatically extracting structured and semistructured data from text documents. In: Proc. 9th ACM SIGMOD Intl. Conf. on Management of Data (SIGMOD). ACM (1998)
11. Chang, C., Lui, S.: IEPAD: information extraction based on pattern discovery. In: Proc. 10th Intl. Conf. on World Wide Web (WWW). ACM (2001)
12. Wang, J., Lochovsky, F.H.: Data extraction and label assignment for web databases. In: Proc. 12th Intl. Conf. on World Wide Web (WWW). ACM (2003)
13. Crescenzi, V., Mecca, G., Merialdo, P.: Roadrunner: towards automatic data extraction from large web sites. In: Proc. 27th Intl. Conf. on Very Large Data Bases (VLDB). Morgan Kaufmann (2001)
14. Lu, Y., He, H., Zhao, H., Meng, W., Yu, C.: Annotating structured data of the deep web. In: Proc. 23rd Intl. Conf. on Data Engineering (ICDE). IEEE (2007)
15. Lu, Y., He, H., Zhao, H., Meng, W., Yu, C.: Annotating Search Results from Web Databases. IEEE Transactions on Knowledge and Data Engineering **25**(3) (2013)
16. Hong, J.L., Siew, E., Egerton, S.: ViWER-Data extraction for search engine results pages using visual cue and dom tree. In: Proc. 1st Intl. Conf. on Information Retrieval & Knowledge Management (CAMP). IEEE (2010)
17. Liu, W., Meng, X., Meng, W.: Vide: A Vision-Based Approach for Deep Web Data Extraction. IEEE Transactions on Knowledge and Data Engineering **22**(3) (2010)
18. Augsten, N., Böhlen, M., Gamper, J.: Approximate matching of hierarchical data using Pq-Grams. In: Proc. 31st Intl. Conf. on Very Large Data Bases (VLDB), VLDB Endowment (2005)