Carbon: Domain-Independent Automatic Web Form Filling

Samur Araujo, Qi Gao, Erwin Leonardi, and Geert-Jan Houben

Delft University of Technology, P.O. Box 5031, 2600 GA Delft, The Netherlands
{s.f.cardosodearaujo,q.gao,e.leonardi,g.j.p.m.houben}@tudelft.nl

Abstract. Web forms are the main input mechanism for users to supply data to web applications. Users fill out forms in order to, for example, sign up to social network applications or do advanced searches in search-based web applications. This process is highly repetitive and can be optimized by reusing the user's data across web forms. In this paper, we present a novel framework for domain-independent automatic form filling. The main task is to automatically fill out a correct value for each field in a new form, based on web forms the user has previously filled. The key innovation of our approach is that we are able to extract relevant metadata from the previously filled forms, semantically enrich it, and use it for aligning fields between web forms.

Keywords: Auto-filling, auto-completion, concept mapping, web forms, semantic web.

1 Introduction

Current applications on the Web show a high degree of user interaction. A large amount of the data that users input into web applications, is supplied through web forms. This process of filling out forms is highly repetitive and can be optimized by intelligently reusing the user's data across web forms, basically following the observation that web forms from applications in a similar domain demand the same data from a user. As a simple example, most sign-up forms require the user's email and first name.

Recently, auto-filling and auto-completion emerged as techniques to assist the users in reusing their data for filling out web forms. **Auto-filling** is a mechanism for automatically filling out web forms. It exists as tools, in web browsers, and when the user visits a web page containing a form, auto-filling can be triggered by a simple mouse click. Google Toolbar Auto-fill [6] is one of the tools available, however it is the simplest form of auto-filling, as it only works in sign-up forms that demand the user's personal data. Another tool is the Firefox Auto-fill Forms plug-in [1] that is also limited to the user's personal data. However, it allows the user to extend the pre-configured fields. Both approaches demand the user to fill out an application-proprietary form containing some basic fields (e.g., name, address, zip code, etc.) before they can be used to automatically fill out web forms. The Safari browser has an auto-fill feature that reuses data from previously filled out forms for automatically

B. Benatallah et al. (Eds.): ICWE 2010, LNCS 6189, pp. 292–306, 2010.

filling out different forms with this data. However, its matching mechanism is only based on the *string matching* of field names. **Auto-completion** is a feature provided by many applications for suggesting a word or phrase that the user wants to enter without the user actually entering it completely. Most of the browsers have native support for auto-completion. It stores values that have been filled before and based on this history it recommends values for a field that has been already visited. However, this mechanism demands user interaction for each field that needs to be filled out.

In this paper, we propose and validate a new concept-based approach for automatically filling out web forms re-using data from previously filled out forms.

Manipulating (the code behind) web forms is not a trivial task, due the high heterogeneity among them. Web forms have different shapes, different numbers of fields, different labels, different representation for values, and different purposes. To be able to derive proposed values for a new form from the knowledge obtained from the already filled out forms, a concept-based structure is used. This helps to represent the knowledge from the filled out forms at a conceptual level and exploit that conceptual level to reason about the proposal for values for fields from the new form. Figure 1 shows an example of the translation of a simple string representing a field name to a meaningful concept in some known vocabulary, for example, WordNet for forms in English. With these concepts we can thus construct a conceptual model with which it is then possible to connect (the concepts from) the target form and its fields to (the concepts from) the data gathered from the previously filled out forms. An essential step in that process is the mapping of 'target' concepts to 'source' concepts. Several options are possible here and we will choose in this paper a semantic-based approach to show the feasibility. Figure 2 shows a mapping between two form fields using a path of concept relations.

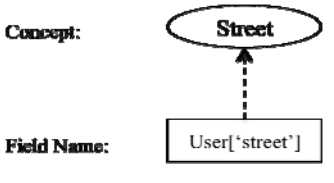

Fig. 1. Example Translation

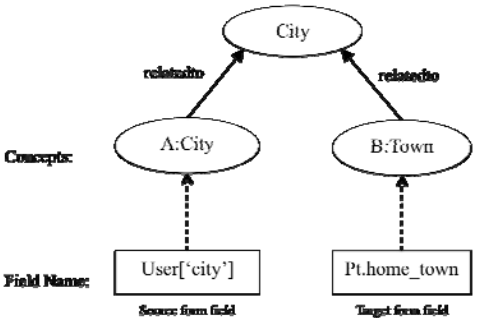

Fig. 2. Example Mapping

The concept-based structure and mappings can thus be used to suggest values for a target form to be filled out automatically. In this respect, we see two main notions that exist in current techniques, auto-completion and auto-filling, which we mentioned before. The main difference between them is that *auto-completion* guesses a value from a fixed and defined set of values (e.g. rows in a database column) based on what the user types and only applies for text fields (not for select, checkbox or radio button fields), while *auto-filling* guesses a value based on the concept that the field

represents and applies to all types of fields. Both techniques are instantiations of a *general* (meta) model for automatically filling out forms that discerns the following aspects:

- **Mapping technique:** Mapping can use string or semantic-based techniques.
- **Data source:** Proposals for values to be filled out can be made on knowledge gathered through forms that have been filled out previously, from a pre-defined list of values extracted from a database (e.g. list of cities), or it can be gathered by explicitly forcing the user to fill out an application-specific form, like in the case of Google Toolbar Autofill.
- **Field identifier:** For matching fields the name of the INPUT tag can be used or the label of the field (the latter is not always available), for example a field can have the label *"First Name"* and the name *"F_name"*.
- **User-intervention:** User interaction can be required as in the case of completion or it can be omitted in the fully automatic case.
- **Domain-dependence:** Approaches can be using domain-specific intelligence, like Firefox Autofill, or be domain-independent.

Existing *auto-filling* approaches are typically string-matching-based tools. They do not consider the concepts associated with the field label and/or field name. Consequently, they are not able to exploit available background knowledge. A simple example is the use of synonym relationships between concepts (e.g., city is the synonym of town).

In this paper, we present Carbon, a domain-independent automatic web form filling approach that exploits the semantic of concepts behind the web form fields. Carbon uses semantic techniques to accumulate and connect the data from previously filled out forms to the target form to be filled out. Carbon goes beyond the state of the art in *auto-filling* since it can be applied for auto-filling any kind of form in the English language.

This paper is organized as follows. After this motivation, we discuss the related work in Section 2. In Section 3, we present a deep look at web forms, with an evaluation of the distribution of field labels in web forms. Section 4 presents our approach for auto-filling forms. Section 5 elaborates on our Carbon implementation in details by describing the architecture and data model for the implementation. We then present an experimental validation of the approach, with a study of the performance of the approach in Section 6. Finally, Section 7 concludes this paper.

2 Related Work

Form Extraction. Extracting labels from web forms is a challenging and non-trivial problem. In [17], Raghavan et al. presented HiWE (Hidden Web Exposer) that used a *layout-based* information extraction technique to find candidate labels and a set of heuristic rules to compute and rank the distance between form elements and candidate labels. In [8], He et al. presented a technique for extracting labels from web forms by capturing the *textual layout* of elements and labels, and based on their position the relationships between elements and labels are determined. Recently, Nguyen et al.

presented LABELEX for extracting element labels from web form interfaces by using *learning classifiers* [14] and they introduced the idea of mapping reconciliation.

Auto-completion. Auto-completion is a feature found in web browsers, e-mail clients, text editors, query tools, etc. Based on the user's input, it *predicts* a word or phrase that a user wants to type before the user types the *whole* word or phrase. One of the early auto-completion facilities was command line completion in the Unix Shell. In [2], the authors introduced an auto-completion feature for full-text search using a new compact *indexing* data structure called HYB to improve the response time of this feature. In most *web browsers*, an auto-completion feature is available for completing URLs, suggesting search terms, and for auto-completing form fields. In the context of web form filling, the web browser typically reuses previously inputted form data for the prediction of values for fields to be filled out in a new form. Recently, Hyvönen et al. [9] generalized the idea of syntactic text auto-completion onto the *semantic* level by matching the input string to ontological concepts.

Auto-filling. In spite of the usefulness of auto-completion in helping users to fill out forms, auto-filling is a step further and often more suitable for this task because it does not require *explicit* user intervention. With an auto-completion tool, a user has to type at least one letter for each field (and thus helps the tool with the proposal of values), where with auto-filling the only action needed is to signal the system to start its process of finding suggestions; note that tools can allow the user to confirm or adjust the given proposal. A number of auto-filling tools [1,6,18] require the users *beforehand* to fill out a predefined "form" (outside the use of any form-based application) that will act as the source of data for later when the tools attempts to automatically fill out a target form. Note that the predefined form usually has a very small number of fields that are typically related to standard *personal* information [18]. Even though some of the tools have the capability of adding extra fields and defining rules [1,18], the users have to explicitly define them beforehand. This task is not trivial and is often perceived as cumbersome by many users. Another tool, called iMacros [10], *records* the interactions between a user and a web page when she fills out a form and then generates macros for these interactions. These macros are then later used to automatically fill out the *same* form. While this tool is useful if the user wants to fill out one kind of form with different values (e.g., in the process of searching for products using various keywords), it *cannot* be used to automatically fill out *different* forms. In [21], a framework called iForm was presented for automatically filling out web forms using data values that are implicitly available in a text document given by the user as input.

Syntactical Matching and Semantic Matching. A host of works [11,13,19,22] has addressed the problem of measuring *syntactical* similarity between strings, and in many approaches for predicting form values such work is used. They measured how similar two strings are by computing the minimum number of *operations* needed to transform one string into the other [13,19], by computing the numbers of matches and transpositions, or by indexing the strings by their pronunciations [11]. Semantic matching is a technique used to identify information that is *semantically* related. Typically, this is achieved through concept-based structures that represent semantic relations between concepts, and the matching process then tries to find connections between concepts based on such relations. The semantic relations are obtained from

ontologies such as WordNet, DBpedia, etc. In [3,4,5,15] the problem of mapping and aligning ontologies was addressed. Tools exist [12,16] that are specifically built to measure semantic similarity between concepts. WordNet::Similarity [16] is a freely available software package that implements a variety of semantic similarity and relatedness measures based on information from the lexical database WordNet[1]. The DBpedia Relationship Finder [12] is a tool for exploring connections/relationships between objects in a Semantic Web knowledge base, specifically DBpedia[2].

3 Web Forms

Web forms are the main input mechanism for users to enter data for web sites and web applications. They occur in different shapes, different numbers of fields, different labels, different values representations (e.g., 'Brazil', 'BR' or 'BRA'), and different purposes (e.g., for signing up, searching or commenting). Forms in HTML are by far the most used ones on the web, but there also exist forms in others standards such as XFORMS[3] and Adobe Flash[4]. Our focus in this paper is on forms expressed in HTML (also XHTML).

In this section we consider the format of forms and report on a study into the nature of forms, reporting how the form elements are distributed and shared between forms.

3.1 Format of Data in HTML Forms

A web form is a set of HTML INPUT tags enclosed by a tag HTML FORM that when rendered in the browser, allows users to enter data in a HTML page. After a user has filled out a web form, it can be submitted to a server application for further processing. Each INPUT field in a form is associated with a name (e.g., "user['First_Name']"), a value (e.g., "John"), and a type (e.g., checkbox, textarea, text, select, radio button or button). Also, a form field can be associated to a human-readable label (e.g. "First name"). In spite of the fact that the HTML specification defines the tag LABEL to represent a human-readable label, it is not widely used by HTML programmers and most forms represent labels in an alternative way. However, even that is not always the case, as form fields can come without label, can have labels in a position hard to detect by machines, or can have meaningless names. In the last case, it is hard to process it for auto-filling, due to the fact that we cannot map an opaque field (with a meaningless name such as "$er32") using any matching strategy.

3.2 Nature of Web Forms

For the challenge of filling out web forms automatically, the first step is to "understand" how similar web forms can be. One way to measure this is by looking for the field labels, and how these labels (or concepts) are distributed and shared between forms. For this purpose, we conducted a study using a typical dataset on web forms.

[1] http:// wordnet.princeton.edu
[2] http://dbpedia.org/About
[3] http://www.w3.org/TR/2009/REC-xforms-20091020/
[4] http://www.adobe.com/products/flashplayer/

3.2.1 Experimental Setup

When we consider web forms in a specific domain, there are two characteristics that are worth further investigation. One phenomenon is that the same labels occur distributed in different forms. For example, in the book domain a label named "author" occurs in many different forms. Another characteristic is that different labels can be connected to a single concept. For example, both of the labels "postal code" and "zip code" can be associated with one concept representing the "postal code". Identifying and exploiting these two phenomena, we have the capability of a significant reduction from labels to concepts.

In an experiment to evaluate these characteristics mentioned above, we adopted the TEL-8 dataset [20], a manually collected dataset, which contains a set of web query interfaces of 447 web sources across 8 domains in the Web, including Airfares, Automobiles, Books, Car Rentals, Hotels, Jobs, Movies, and Music Records. We note that they are typical domains on the Web and that this dataset captures the structures of forms on the Web nowadays. For each domain, we gathered all labels from TEL-8 files and we grouped them into sets of distinct labels. Then we categorized these distinct labels by mapping them to concepts, manually. Afterwards, we investigated the distribution of labels and concepts in the forms for each domain and analyzed the reduction from labels to concepts.

3.2.2 Experimental Results

The results show the distribution of forms, (distinct) labels and concepts and also the average number of labels and concepts in the forms, for each domain (see Table 1). In Table 1 we see the number of forms per domain and the number of labels that are contained in all those forms. We see also how many of those labels are distinct labels. Further, we see how many concepts are associated with these distinct labels.

Table 1. Nature of Web Forms

	Airfares	Auto mobiles	Books	Car Rentals	Hotels	Jobs	Movies	Musics Records
forms	65	47	84	25	39	49	73	65
labels	332	505	402	189	269	284	455	421
distinct labels	168	241	175	126	197	221	287	183
concepts	132	164	116	110	149	162	213	134
reduction rate	60%	68%	71%	42%	45%	43%	53%	68%

We define a measure named "Reduction Rate" to represent the reduction from the labels to the concepts in the web forms due to the conceptual mapping. Given one specific domain, let N_c be the number of concepts and N_l be the number of labels. The reduction rate is defined as follows:

$$ReductionRate = \frac{N_l - N_c}{N_l}$$

The reduction rate ranges from 42% to 71% and is 55% in average. The higher this number is, the fewer concepts we need to cover all the fields in one domain. In Table 1, we can see that the reduction rate is composed of two parts. The first part of the

reduction is from labels to distinct labels, due to the fact that labels are repeated among different forms. The second part is caused by the application of concept mapping that groups distinct labels into clusters of concepts that they represent. The intelligent exploitation of these two reduction steps is the main motivation for our approach.

4 Conceptual Framework for Form Auto-Filling

The main idea behind our approach for automatically filling out web forms is to extract data from previously filled out forms and propose them for the new form. Thus we exploit the observations we made in the study presented above. The use of data from previously filled out forms, increases, progressively, the variety of forms that can be automatically filled, since, once a user manually has filled out a form in a specific domain (e.g., social networks or hotels), this data is then available as a solid basis for automatically filling out other forms in the same domain.

This approach allows Carbon to go beyond of the state of the art of available auto-filling tools by exploiting the semantic overlap of knowledge contained in the previously filled out forms.

4.1 Extracting Concepts from Web Forms

We represent a form field by a conceptual structure containing the field name, the label, the type, the values, the URL of the page containing the form, the domain of the URL, the universal unique identifier of the form, and the update date. The process of auto-filling forms starts with the instantiation of this conceptual structure by extracting metadata from web forms that the user fills out. Afterwards this metadata is stored and enriched for further use. The next step is to instantiate a similar structure for a target form, and subsequently to map this instance to the previously obtained knowledge structure, to obtain suggestions for values for empty fields. The mapping between form representations can exploit any attribute of the knowledge structure, but in the implementation for the experiment described in this paper, we just exploited the attribute representing the form field's name.

As we typically do not have access to (the database behind) the web application, we choose to collect the form field metadata in real-time - i.e. after a user fills out the form and before submitting it. We extract this metadata from the DOM[5] (Document Object Model) tree of the HTML page that contains the form. The DOM is a programming API for documents that has a logical structure that represents a document as a tree of objects. By navigating through the HTML DOM tree we can access any web page element and their attributes, including form input fields. Carbon processes the DOM tree, extracting name, value, and type of the HTML INPUT tags. The Carbon version used in this experiment does not implement a label extraction strategy, however this can be easily plugged into the architecture.

The extraction intelligence applied for previously filled out forms and for an (empty) target form is logically similar, but obviously occurs at distinct moments.

[5] http://www.w3.org/DOM/

4.2 Mapping Web Form Concepts

The main idea behind Carbon is to connect fields at the conceptual level. For example, Carbon can connect the (different) concepts "city" and "town" using WordNet, a large lexical database for English, for bringing two words together that represent synonyms.

Carbon starts by mapping a field name string to atomic syntactic elements or lexical words in WordNet. Carbon splits the string by eliminating all non-letter characters, e.g., the string "reg.name[last]" is split into "reg", "last" and "name". Afterwards, Carbon builds a tree of prefixes and suffixes for each of the thus split strings and looks for English terms in the WordNet database that match these substrings. So, in the example it only retrieves (the concepts for) the terms "last" and "name". By exploiting WordNet collations (sequences of words that go together, such as "zip code"), Carbon also retrieves the term "last name". Carbon uses WordNet synsets (a set of word or collation synonyms) to retrieve synonyms of the found terms. So, Carbon maps the field name string to all WordNet terms thus found, e.g., "reg.name[last]" is mapped to the WordNet terms "name", "last", "last name", "surname" and "family name".

Like this, Carbon can use this knowledge to propose values for a target form field.

5 Auto-Filling Web Forms with Carbon

Carbon has two main parts: Carbon Client and Carbon Server. Figure 3 shows an overview of the interactions between the Carbon Client and Carbon Server.

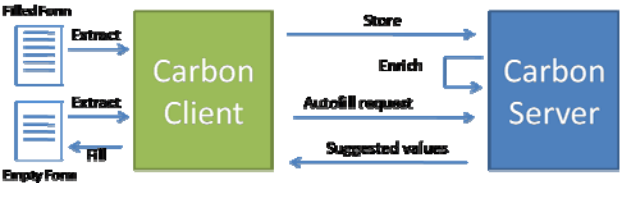

Fig. 3. Client-Server Interactions.

Carbon Server is a semantic application that stores and enriches metadata about web forms in order to recommend values in the process of automatically filling forms. Carbon Client is a browser extension that processes web pages extracting relevant metadata from previously filled out web forms, extracts relevant metadata from an empty target form, and automatically suggests values for the empty form's fields.

5.1 Carbon Client

Carbon Client was implemented as a Greasemonkey[6] script. Greasemonkey scripts can be easily added to the Firefox browser and can be enabled or disabled with a simple mouse click. Carbon uses it to have access to the DOM tree of the HTML page

[6] http://www.greasespot.net/

that the user visits. The communication between Carbon Client and Carbon Server is done via the *XMLHttpRequest*[7] object.

Since Carbon uses previously filled out forms for the purpose of auto-filling an empty form, Carbon Client extracts metadata about all forms that the user fills out. For doing this, for every page that the user visits Carbon Client accesses the *DOM tree*, searches for all input elements, and adds to them a *Javascript*[8] *Onblur DOM event*. An *Onblur* event triggers a function when the user moves the focus from an HTML INPUT element to another. At this moment, Carbon Client extracts the metadata (name, value, and type) of the input tag that triggered the event and sends, together with the page URI, such information to Carbon Server via an *AJAX*[9] *request*. Once the request has been received, Carbon Server processes it and stores it.

When the user visits a page with an empty form, Carbon Client can be triggered to automatically fill out the form. This triggering can occur upon the explicit request of the user or automatically whenever the user visits a page with an empty form. At that moment, for each field, Carbon Client extracts metadata, and sends, with the page URI, an *autofill* request to Carbon Server. Carbon Server retrieves suggestions for values that fit those fields. Carbon Client uses these values to fill out the empty fields.

5.2 Carbon Server

Carbon Server plays a main role in Carbon as it is responsible for storing and enriching metadata about web forms. Also, Carbon Server implements the logic behind the mapping of form fields.

Carbon Server is a web application that was implemented using the Ruby on Rails[10] framework. As storage technology we use Sesame[11], an RDF (Resource Description Framework)[12] open source framework. We also use ActiveRDF[13], a Ruby library for manipulating RDF data following the object-oriented paradigm. Figure 4 gives an overview of the Carbon Server implementation architecture.

Carbon Server uses 3 different data models: a Form Data Model, a Configuration Data Model, and an Enriched Data Model. All these models are instances of the Carbon Ontology that we will not detail in this paper due to space limitations. Using RDF as the representation model, Carbon can be easily extended to include any relevant metadata about web forms. Also, RDF is the foundation of the Semantic Web, and a lot of open data is being published in this environment, such as WordNet that Carbon uses in the process of mapping form fields. The main benefit of the use of such a model is that Carbon can consume data from any external source in the Semantic Web, and thus provides a flexible and extensible environment that allows for the definition of new rules and their application in the conceptual mapping of fields. For example, Carbon could use dictionaries in other languages, in addition to the English version of WordNet that is already used in the process of mapping fields.

---

[7] http://www.w3.org/TR/XMLHttpRequest/
[8] https://developer.mozilla.org/en/About_JavaScript
[9] http://en.wikipedia.org/wiki/Ajax_%28programming%29
[10] http://rubyonrails.org/
[11] http://www.openrdf.org/
[12] http://www.w3.org/RDF/
[13] http://www.activerdf.org/

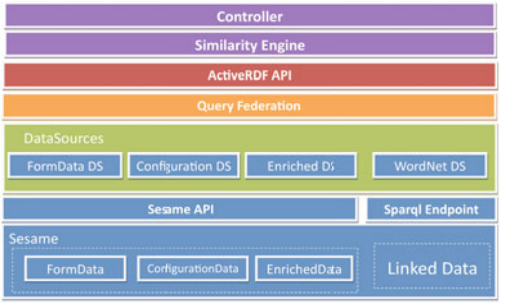

Fig. 4. Carbon Server Implementation Architecture

The Form Data Model stores the conceptual structures of forms and fields, for all previously filled out forms. For example, the concept for the field with the label "Last Name", the name "reg.lastName", the type "text", and the value "Donald" is represented as follows in the Carbon Ontology using the RDF notation N3[14].

```
@prefix carbon: <http://www.carbon-autofill.org/> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>  .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>  .
<carbon:7382> <rdfs:type> <carbon:Field>  .
<carbon:7382> <carbon:fieldLabel> 'Last Name'  .
<carbon:7382> <carbon:fieldId> <carbon:7382>  .
<carbon:7382> <carbon:originalFieldName> 'reg.name[last]'  .
<carbon:7382> <carbon:name> 'regnamelast' .
<carbon:7382> <carbon:formUri>
<https://secure.gettyimages.com/register/>  .
<carbon:7382> <carbon:urlDomain> <https://secure.gettyimages.com>  .
<carbon:7382> <carbon:updated_at> '1254390507.71' .
<carbon:7382> <carbon:fieldValue> 'Donald ' .
```

Following this data model, all *form data* is stored in a Sesame repository called FormData.

The Enriched Data Model stores extra knowledge about instances of the Form Data Model. It can store any meta-knowledge about the conceptual structures; in the implementation that we used in the current experiment it stores concepts that are related to fields' names and labels. The example below shows the enrichment related to a form field labeled "Last Name":

```
@prefix carbon: <http://www.carbon-autofill.org/> .
@prefix wordnet: <http://www.w3.org/2006/03/wn/wn20/instances/>  .
<carbon:7382> <carbon:relatedto> <wordnet:synset-last_name-noun-1> .
<carbon:7382> <carbon:relatedto> <wordnet:synset-surname-noun-1>  .
<carbon:7382> <carbon:relatedto><wordnet:synset-family_name-noun-1> .
```

Here, the WordNet synset consisting of the words or collations "last name", "surname" and "family name" is associated to the concept 7382 that represents the form field labeled "Last Name". The property *relatedto* defined in the Carbon

[14] http://www.w3.org/DesignIssues/N3Resources

Ontology was used to denote this association. Following this data model, all *enriched data* is stored in a Sesame repository called EnrichedData.

The Configuration Data Model allows to extend the knowledge from the Form Data Model and the Enriched Data Model. For example, the concepts "street" and "address" are not synonyms in the WordNet vocabulary, so Carbon will not map these two. Even another semantic matching strategy such as WordNet::Similarity[16], will give them a low degree of similarity. However, the designer can add extra triples in this repository (using Carbon's named graph), see the example below, which extends the WordNet synsets enabling Carbon to perform better for this example.

```
  @prefix carbon: <http://www.carbon-autofill.org/> .
  @prefix wordnet: <http://www.w3.org/2006/03/wn/wn20/instances/>  .
  <wordnet:synset-address-noun-1><wordnet:containsWordSense>
<wordnet:wordsense-street-noun-1> .
```

The triple above connects these two concepts using the property "containsWordSense" which is the main relation exploited by Carbon to determine the similarity among concepts.

6 Evaluations

In this section, we present two sets of evaluations. In the first set of experiments we want to see how many concepts in a new form have been found in the previously filled out forms. We define this as concept completeness. In the second set of experiments, we examine the effectiveness of Carbon in terms of precision and recall. Both sets of experiments use the TEL-8 (see Section 3.2) data set with Airfares, Automobiles, Books, Hotels, Jobs, and Movies domains.

6.1 Concept Completeness: Definition and Evaluation

We first formally define the notion of concept completeness. Given a set CS of concepts related to fields in a set S of filled out forms and a set Cf of concepts related to fields in a new form f, the concept completeness of form f given a set of filled forms S is defined as follows:

$$Completeness(f,S) = \frac{|C_S \cap C_f|}{|C_f|}$$

The value of concept completeness ranges from 0 to 1. If it equals 1, then it means that all the concepts in the new form are completely covered by the concepts in the previously filled out forms. If the concept completeness equals 0, then the previously filled forms are not useful in filling out the new form.

For all the forms in a domain *d* in the TEL-8 data set, we did the following:

1) We mapped the fields in all forms to WordNet concepts. In this set of experiments, we used the field labels that were extracted and available in the data set and mapped them to WordNet concepts.

2) For each form f in domain d, we determined all possible subsets of forms in domain d with size r, and used these subsets as sets of source forms. Then, for each subset S_r, we computed the $Completeness(f,S_r)$.
3) Finally, we computed the concept completeness average grouped by d and r.

Table 2 shows the concept completeness for 6 different domains based on the concept mapping performed by human experts (denoted by "Experts") and by our Carbon Server (denoted by "Carbon"). The results of the Experts act as benchmark for Carbon. Note that r denotes the number of forms that a user has previously filled out and in this table r ranges from 1 to 6.

Table 2. Concept Completeness Evaluation Results

r	Airfares		Automobiles		Books		Hotels		Jobs		Movies	
	Carbon	Experts	Carbon	Experts	Carbon	Experts	Carbon	Experts	Carbon	Experts	Carbon	Experts
1	25.21%	49.24%	19.45%	26.27%	34.07%	45.97%	16.53%	21.51%	7.92%	15.01%	11.64%	17.83%
2	40.59%	64.85%	29.57%	39.42%	48.10%	60.05%	27.91%	34.67%	14.66%	26.13%	19.86%	29.29%
3	50.37%	71.50%	35.84%	47.22%	54.90%	66.05%	36.05%	43.33%	20.46%	34.57%	25.91%	37.07%
4	56.93%	75.32%	40.34%	52.60%	58.90%	69.59%	42.11%	49.47%	25.49%	41.15%	30.54%	42.65%
5	61.61%	78.00%	43.89%	56.70%	61.69%	72.15%	46.83%	54.13%	29.91%	46.42%	34.24%	46.88%
6	65.14%	80.10%	46.86%	60.03%	63.88%	74.17%	50.67%	57.88%	33.83%	50.74%	37.30%	50.23%

We see that the concept completeness of a new form becomes higher if a user has filled out more forms previously. For Carbon, the concept completeness of the "Airfares" domain increases from 25.21% to 65.14%. For Experts, it increases from 49.24% to 80.10%. On average the concept completeness of Carbon and Experts for r = 6 is 50.73% and 62.31%, respectively. This means that even though a user has only filled out 6 forms, the concepts in the filled forms can cover 50-62% of the concepts in a new form. The uncovered concepts in the new form are usually application-specific, meaning that those concepts occur only in a small number of forms.

We also see that the increment rate becomes slower as the user fills out more forms: the more forms a user fills out, the fewer new concepts can be discovered. For example, the increment rate of the concept completeness for Carbon in the "Hotels" domains drops from 11% when r=2 to 4% when r = 6. A similar result is also revealed for Experts. Checking all results in the 6 domains, we found the increment in all domains to become rather small (all below 5%) when r reaches 6. Considering the increase of computation complexity and the decrease in increments for the concept completeness, we therefore set the maximum value of r to 6.

We also observe how the concept completeness of human experts is (expectedly) always higher than Carbon's. On average, the concept completeness of Carbon reaches almost 74% of that of the human experts. It shows that, while there is still room for improvement, the ability of this Carbon version to map labels and concepts is quite close to the one of human experts.

6.2 Effectiveness: Performance Measures and Dataset

In our second evaluation, precision and recall are used as the performance measures [7]. Precision expresses the proportion of retrieved relevant fields among all the

retrieved fields, while recall expresses the proportion of retrieved relevant fields from the total relevant fields:

$$Precision = \frac{|\{Relevant\} \cap \{Retrieved\}|}{|\{Retrieved\}|} \qquad Recall = \frac{|\{Relevant\} \cap \{Retrieved\}|}{|\{Relevant\}|}$$

The basic idea of this experiment is to compare the forms filled by the human experts according to users' profiles to those filled by Carbon automatically. In detail it is divided into four steps described as follows:

1) We randomly chose 10 forms for each domain as the test set.
2) We collected 6 user profiles according to the real personal information, for each domain. These profiles will be used as the facts for filling the forms.
3) In order to evaluate the performance of Carbon, we needed to set up a benchmark for the evaluation by filling out 10 forms for each domain according to these user profiles. This was accomplished by human experts.
4) For each domain we then chose one available form as a target form. Based on the results of the first set of experiments, another 6 forms were selected randomly as the source forms, representing the forms that users have filled out formerly. Then Carbon filled the target form automatically according to the knowledge extracted from these source forms. Repeating this process for 8 different target forms, we calculated the *precision* and *recall* for the domain.

6.3 Effectiveness Evaluation Results

Table 3 summarizes the results of the effectiveness evaluation on Carbon in terms of precision and recall for each domain. The precision ranges from 0.54 to 0.81 and on average is 0.73. The recall ranges from 0.42 to 0.61 and on average is 0.53. We can see that the recall is less than ideal. There are two explanations for this recall result. One is that when we use the field names instead of the labels in the experiments, some of the field names, e.g. "inp_ret_dep_dt_dy", "DEST-1", etc., are not meaningful enough to be parsed to words and mapped to concepts. Sometimes the field name is meaningful, but Carbon fails to map it to related concepts. Taking the example of the Airfare domain that has the lowest recall, among all the 77 relevant fields in this domain, 18 fields are meaningless. If we ignore these fields and re-calculate the recall, it will increase to 0.55 from 0.42. Furthermore, we observe that in those cases where there are labels they are more meaningful than the field names, and despite that these labels could improve Carbon's precision and recall, using the names we have shown the capability of the semantic matching approach for auto-filling forms.

Table 3. Precision and Recall Results

	Airfares	Automobiles	Books	Hotels	Jobs	Movies	Avg
Precision	0.54	0.73	0.75	0.79	0.81	0.73	0.73
Recall	0.42	0.61	0.60	0.53	0.59	0.47	0.54

7 Conclusion

Filling out forms is an essential aspect of many web applications and many users are confronted with a large degree of repetition in this process across applications. Tools exist to help users in this process, but an optimization step is not only welcome but also feasible if we are able to integrate form data across web forms. In this paper, we presented a novel framework for domain-independent automatic form filling. The main challenge behind the approach is to provide good suggestions for the values to be used for each field in a new form to be filled out, and to do so based on the web forms the user has previously filled out. We have approached this challenge with a number of innovative steps in which we are able to extract relevant metadata from the previously filled forms, semantically enrich this metadata, and use it for aligning fields between web forms. We have also given details of experimental validations of the approach. First, to describe the nature of the problem and challenge, we have done a study of the distribution of field labels in web forms. Second, we have conducted a study of the performance of the approach with the Carbon implementation.

As our focus in this paper is to show how to exploit the semantics of concepts behind the web form fields and to use semantic-based techniques to automatically fill out web forms, several usability and privacy issues could not be discussed in this paper. Amongst them are web form domain resolution, encrypted transfer of data and the management of form data for multiple users. The first issue can effectively be resolved by plugging a web page classifier into Carbon's architecture. Thus, Carbon is able to select data from previously filled forms that are classified in the same domain as the target form. The second and third issues can also be addressed, by adding an encryption and authentication system into Carbon's architecture, respectively.

In the continuation of this research, we study how a hybrid approach that uses a combination of auto-filling and auto-completion performs in terms of effectiveness, based on the observation that auto-completion could be exploited over the enriched knowledge structure that was created over the form data stored in the user's history.

References

1. Autofill Forms – Mozilla Firefox Add-on, `http://autofillforms.mozdev.org/`
2. Bast, H., Weber, I.: Type Less, Find More: Fast Autocompletion Search with a Succinct Index. In: The Proceedings of SIGIR 2006, Seattle, USA (August 2006)
3. Bouquet, P., Serafini, L., Zanobini, S.: Semantic Coordination: A New Approach and an Application. In: Fensel, D., Sycara, K., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 130–145. Springer, Heidelberg (2003)
4. Doan, A.H., Domingos, P., Halevy, A.Y.: Learning to Match the Schemas of Data Sources: A Multistrategy Approach. Machine Learning 50(3), 279–301 (2009)
5. Doan, A.H., Madhavan, J., Domingos, P., Halevy, A.Y.: Learning to Map between Ontologies on the Semantic Web. VLDB Journal, Special Issue on the Semantic Web 12(4), 303–319 (2003)
6. Google Toolbar Autofill, `http://toolbar.google.com/`
7. Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Morgan Kaufman, San Francisco (2001)

8. He, H., Meng, W., Yu, C.T., Wu, Z.: Automatic Extraction of Web Search Interfaces for Interface Schema Integration. In: the Proceedings of WWW 2004 - Alternate Track Papers & Posters, New York, USA (May 2004)
9. Hyvönen, E., Mäkelä, E.: Semantic Autocompletion. In: Mizoguchi, R., Shi, Z.-Z., Giunchiglia, F. (eds.) ASWC 2006. LNCS, vol. 4185, pp. 739–751. Springer, Heidelberg (2006)
10. iOpus Internet Macros, http://www.iopus.com/
11. Knuth, D.E.: The Art of Computer Programming. Sorting and Searching, vol. 3, pp. 394–395. Addison-Wesley, Reading (1973)
12. Lehmann, J., Schüppel, J., Auer, S.: Discovering Unknown Connections - the DBpedia Relationship Finder. In: The Proceedings of CSSW 2007, Leipzig, Germany (September 2007)
13. Levenshtein, V.I.: Binary Codes Capable of Correcting Deletions, Insertions, and Reversals. Soviet Physics Doklady 10(8), 707–710 (1966)
14. Nguyen, H., Nguyen, T., Freire, J.: Learning to Extract Form Labels. In: the Proceedings of VLDB 2008, Auckland, New Zealand (August 2008)
15. Noy, N.F., Musen, M.A.: PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In: The Proceedings of AAAI/IAAI 2000, Austin, USA (July-August 2000)
16. Pedersen, T., Patwardhan, S., Michelizzi, J.: WordNet: Similarity - Measuring the Relatedness of Concepts. In: The Proceedings of AAAI/IAAI 2004, San Jose, USA (July 2004)
17. Raghavan, S., Garcia-Molina, H.: Crawling the Hidden Web. In: The Proceedings of VLDB 2001, Rome, Italy (Septmeber 2001)
18. RoboForm, http://www.roboform.com/
19. Smith, T., Waterman, M.: Identification of Common Molecular Subsequences. Journal of Molecular Biology 147(1), 195–197 (1981)
20. TEL-8 Query Interfaces, http://metaquerier.cs.uiuc.edu/repository/datasets/tel-8/
21. Toda, G.A., Cortez, E., de Sá Mesquita, F., da Silva, A.S., de Moura, E.S., Neubert, M.S.: Automatically Filling Form-based Web Interfaces with Free Text Inputs. In: the Proceedings of WWW 2009, Madrid, Spain (April 2009)
22. Winkler, W.E.: The State of Record Linkage and Current Research Problems. Statistics of Income Division, Internal Revenue Service Publication R99/04 (1999)