Onebox: Free-Text Interfaces as an Alternative to Complex Web Forms

Kien Tjin-Kam-Jet, Dolf Trieschnigg, and Djoerd Hiemstra

University of Twente, Enschede, The Netherlands
{tjinkamj,trieschn,hiemstra}@cs.utwente.nl

Abstract. This paper investigates the problem of translating free-text queries into key-value pairs as an alternative means for searching 'behind' web forms. We introduce a novel specification language for specifying free-text interfaces, and report the results of a user study where we evaluated our prototype in a travel planner scenario. Our results show that users prefer this free-text interface over the original web form and that they are about 9% faster on average at completing their search tasks.

Keywords: query processing, natural language interfaces, deep web, structured data, query translation, query reformulation.

1 Introduction

The internet contains a large amount of information that is only accessible through complex web forms. Journey planners, real estate websites, second-hand car websites, and other websites commonly require the user to fill in a form consisting of a number of fields in a graphical interface. The user should first interpret the form and then translate his information need to the appropriate fields. Moreover, filling out these forms can be slow because they require mixed interaction with both the mouse and keyboard. A natural language interface (NLI) alleviates these problems by allowing the user to enter his information need in a single textual statement. Rather than navigating between and entering information in the components of the web form, the user can focus on formulating his information need in an intuitive way. NLIs require or assume syntactically well-formed sentences as input, in essence restricting the range of textual input. However, describing all possible natural language statements and dealing with query ambiguity can be a time-consuming process [1–4]. Therefore, we introduce a free-text interface (FTI) which allows the user to *freely input any text* without any restrictions. In this paper, we describe and evaluate a prototype system for specifying FTIs to access information behind complex web forms. The system has been designed to specify flexible FTIs with relatively little effort.

This work is a stepping stone for further investigation of a single textual interface to access the deep web [5]. Ideally, we wish to use these techniques to build a distributed search system which can search multiple resources, including information behind complex web forms, simultaneously. The contributions of this paper are as follows: *i*) we introduce a specification language for describing free-text interfaces (FTIs) to complex web forms; *ii*) as a proof of concept, we show that this language can be effectively used to describe a flexible FTI to a travel planner web form; and *iii*) we demonstrate that users find results faster with an FTI than with a existing web form.

The remainder of this paper is structured as follows: Section 2 describes the requirements and our prototype framework. Our experiment setup and the results are described in Section 3. Sections 4 and 5 discuss our work and overview related work. Finally, Section 6 concludes our work.

2 Free-Text Interfaces to Web Forms

The goal of the FTI is to offer simple textual access to content behind a web form. The FTI should rank plausible interpretations (basically, ways to fill out a web form) of the user's free-text input. Kaufmann and Bernstein [2] showed that it is important to guide the user during the query formulation process. Therefore, the FTI should also offer query suggestions to guide users in formulating their queries. Lastly, it should be easy to specify the capabilities of the FTI (from the developer's perspective). Figures 1 and 2 give examples of a complex web form of a travel planner website (with many input fields and options) and an FTI to the same web form, respectively.

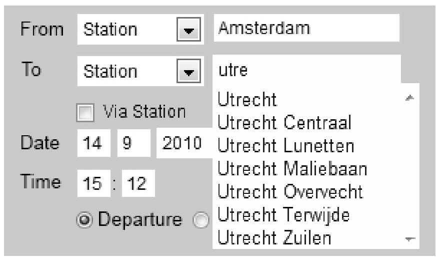

From: Station — Amsterdam
To: Station — utre
Utrecht
Utrecht Centraal
Utrecht Lunetten
Utrecht Maliebaan
Utrecht Overvecht
Utrecht Terwijde
Utrecht Zuilen
Via Station
Date: 14 9 2010
Time: 15 : 12
Departure

Fig. 1. A complex web form (that offers interactive query suggestions), based on the Dutch Railways site.

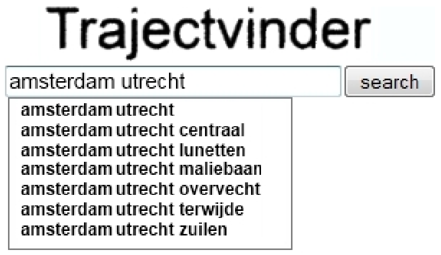

Trajectvinder
amsterdam utrecht
search
amsterdam utrecht
amsterdam utrecht centraal
amsterdam utrecht lunetten
amsterdam utrecht maliebaan
amsterdam utrecht overvecht
amsterdam utrecht terwijde
amsterdam utrecht zuilen

Fig. 2. Trajectvinder ('Route Finder'): a free-text interface (that offers interactive query suggestions), tailored to the complex web form.

2.1 Framework

Our framework basically scans the user input for known patterns. Particularly, it tries to interpret the input by finding valid pattern combinations. The patterns, and other configurable items, will be described in more detail in Section 2.2.

Obtaining and ranking interpretations. The process of interpreting the user input follows the following five steps:

1. *Scanning for input regions:* the user input is scanned for known patterns from left to right, on a word-by-word basis (a word is delineated by a white space). At each word, the longest matching pattern(s) starting from that word is (are) stored. This process yields a set of possibly overlapping input *regions*, where each region could have several annotations. (A region could be matched by different patterns, e.g. a region containing the token '2000' could be annotated as: a year, an amount of money, or a car model);
2. *Generating non-overlapping region sets:* the set $\mathit{\Gamma}$ of all non-overlapping sets of regions with maximal coverage of the input is generated;
3. *Generating interpretations:* for each region set $\gamma \in \mathit{\Gamma}$ all possible combinations of annotations are generated. This results in the set of possible interpretations;
4. *Filtering interpretations:* first, interpretations are 'cleaned' by removing extraneous annotations. Examples of extraneous annotations are prefix (or postfix)-annotations that precede (or follow) annotations which are not specified by the pattern to which prefix (or postfix) corresponds. Also, all annotations that exceed the number of times they are allowed to appear in the underlying web form are extraneous annotations. Second, the complete interpretation is removed if it does not satisfy the constraints; and,
5. *Ranking interpretations:* the interpretations are ranked by the number of annotations they contain, and by the specified pattern order in the configuration file.

Generating suggestions. We generate two types of suggestions: token expansions and pattern expansions. The suggestions are interactive query expansions [6], based on the last few input characters, and are only shown when they are applicable. When the last few input characters denote a prefix of some pattern, this triggers token suggestions of the expected type, only if the type was defined by a list of tokens. If the expected type was defined by a regular expression, no suggestions can be shown. When the last few input characters denote the body of some pattern, and if the set of postfix strings of this pattern is non-empty, then the default (longest) postfix is shown.

2.2 Configurable Items

The FTI can be configured by specifying the following items: *i*) the web form's lexicon; *ii*) the constraints, *iii*) the patterns; and *iv*) the result generation rules.

Lexicon. The input fields of a web form often syntactically constrain user input, e.g. limiting the number of characters, or only accepting input from a pre-defined list of tokens. Regular expressions are used to pose even more syntactic restrictions, such as, allowing only numbers or zip-codes. Input fields (e.g. drop-down menus) may map external strings to internal form values. The lexicon contains known values (both internal and external): it consists of the regular expressions, the list of tokens, and the mapping from external to internal values.

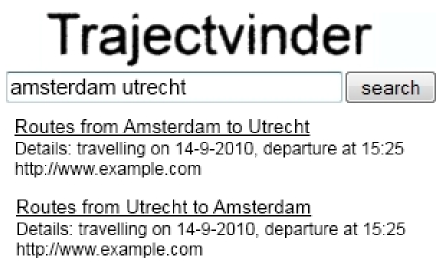
Constraints. The constraints denote value restrictions and relations. Example restrictions are mandatory values (i.e. values that must be found in the input), or value comparisons (e.g. a price value should be greater than zero). Value relations are apparent in some web forms. For example, in the car-sales domain, each value from the class "car brands" can have an associated set of values from the class "car models". Whenever a brand is selected, the list of available models changes accordingly. These relations are useful for: limiting the set of valid queries, ranking query interpretations, and generating query suggestions.

Patterns. Consider the input query "find me a trip to Amsterdam from Paris". Here, the values are 'Amsterdam' and 'Paris', and the contexts are 'to' and 'from', respectively. A system that responds with "from Amsterdam, to Paris" in the first place and "from Paris, to Amsterdam" in the second place, is not considered user friendly (as it generated a false positive and it may have wasted the user's valuable time). Simply scanning text input for known values, without considering the context of the extracted values, may lead to unnecessary query interpretations.

Therefore, we adopt a bottom-up approach for capturing the context: a set of patterns must be specified for each input field. Each pattern consists of three parts: a prefix, a body, and a postfix part. The affixes (the prefix and postfix) each denote a finite list of strings, including the empty string. Note that the affixes can be used to incorporate idioms of natural language. If a particular value is found (i.e. it matched the body part) as well as a corresponding (non-empty) affix, that value is then bound to the field to which this pattern belongs.

Two patterns could be combined into a range pattern. A range pattern is useful for disambiguation. For example, the input '1000 - 2000 euros' would be interpreted as 'minimal 1000 euros and maximal 2000 euros'. Without range patterns, we would find '1000' (it could be a car model, a min price, or a max price) and '2000 euros' (it could be min price or max price), which would have to be further processed in order to remove erroneous interpretations.

Result generation rules. A query interpretation is displayed as a *result snippet* (see Fig. 3), containing a title, a description, and a URL. To generate the title and the description, an ordered set of *field templates* must be specified. A field template specifies how a field's value should be displayed. To generate the URL, the http-request method (i.e. get or post), the actual action-URL, and all the form's input fields (including hidden fields, fields displayed as radio buttons, etc.) must be specified.



Routes from Amsterdam to Utrecht
Details: travelling on 14-9-2010, departure at 15:25
http://www.example.com

Routes from Utrecht to Amsterdam
Details: travelling on 14-9-2010, departure at 15:25
http://www.example.com

Fig. 3. Interpretation as result snippets.

2.3 A Toy Example

Figure 4 depicts an example configuration file and shows how the lexicon, the constraints, the patterns, and the result generation rules, are specified.

The `tokens` element contains token instances. Each `instance` belongs to a specific *type*, has one internal value and a list of external values (treated as synonyms by the system). Multiple instances can belong to a single type.

The `id` attribute of a `pattern` element contains the name of the input field to which the captured value should be assigned. A pattern's `capture` element specifies the *type* to be captured. A pattern's `prefix` (`postfix`) element specifies a list of strings. (This may be specified as a Kleene star-free regular expression, which is expanded to the list of possible strings.)

The `constraints` may contain: *i*) a list of mandatory field combinations; or *ii*) a list of comparisons, e.g. comparing the value of one field to the value of another or to some constant value.

```
<?xml version='1.0' encoding='UTF-8' standalone='yes' ?>
<root>
  <tokens>
    <instance type='station' internal='1'>
      <external>amsterdam amstel</external>
      <external>amstel</external>
    </instance>
    <instance type='station' internal='2'>
      ...
  </tokens>
  <patterns>
    <pattern id='fromloc'>
      <option>
        <prefix>((depart(ing|ure)? )?from)?</prefix>
        <capture>station</capture>
      </option>
    </pattern>
    <pattern id='toloc'>
      ...
  </patterns>
  <constraints>
    <mandatory_fields>
      <fieldset>
        <field>fromloc</field>
        <field>toloc</field>
      </fieldset>
    </mandatory_fields>
    < field_field  not_equal='fromloc' to='toloc' />
  </constraints>
  <results>
    <url method='get'>http://www.example.com/search.html?loc1={fromloc}&amp;...</url>
    <title max='3' starttext='Example.com: search results for '>
      <fieldtemplate id='fromloc' prefix='from ' postfix=' ' />
      <fieldtemplate id='toloc' prefix='to ' postfix=' ' />
      ...
    </title>
    <defaults>
      <field id='arrivalTime' external='arriving on ' internal='true'/>
    </defaults>
  </results>
</root>
```

Fig. 4. An example configuration file.

Lastly, the `results` element specifies what the interpretation's title, description, and URL should look like. It also contains possible default (internal) values (with corresponding external values) for input fields. The `url` element specifies both the action-URL and http-request method. The `title` element (just like the omitted `description` element) contains an ordered list of `field templates`. Each template corresponds to exactly one of the form's input fields, indicated by the `id` attribute. The title (description) is generated by listing the contents of its `start text` attribute and concatenating the contents of the active field templates, up to the specified `max` number of templates. A template is active if the value of the input field it refers to is not empty.

3 Experiment and Results

We developed a prototype framework and evaluated it for an existing travel-planner web form. The web form is as depicted in Fig. 1, the resulting FTI is depicted in Fig. 2. A total of six information items can be specified in the form: a departure location, an arrival location, an optional via location, the time, the date, and a flag indicating whether the date and time are for arrival or departure.

In this experiment, we tried to answer the following questions: *i*) do people prefer to use such an FTI over the existing web form in the first place? *ii*) is searching by means of an FTI faster than searching by means of a complex web form? *iii*) how much variation exists in the query formulations? *iv*) are people consistent in their query formulations? *v*) what are the most positive and negative aspects of the FTI? and *vi*) why is the FTI better, or worse, than the complex web form?

3.1 Experimental Setup

Experimental procedure. The experiment consisted of an offline and an online part, followed by a questionnaire.

During the offline part, the subjects first provided background information (e.g. age, study). Then, they wrote down their 'most recent travel question' if they could remember it. Next, an information need was shown as a topographical picture with lines connecting two or three places, along with a desired date and time. The subjects were asked to fill out the complex web form (on paper) based on this information need. Next, they were shown a different information need and had to fill out the FTI (also on paper). Finally, the subjects were shown a filled out complex web form, and they reformulated that into a question suitable for the FTI. We aimed to collect query formulations, with as little bias to the question as possible. That is why we asked the subjects to formulate a query from memory, and to formulate a query based on pictures instead of textual descriptions of the information need.

During the online part, the subjects were given time to experiment with the complex interface of the existing travel planner site. Then, their task was to find 5 specific train routes and to write down the departure and arrival times.

We recorded the total time to complete all five tasks. Each route was described textually, with a different order of the information items (i.e. the date, time, and locations), and with different wordings (e.g. *ten past one*, or *13:10*). Next, the subjects were given time to experiment with the FTI. After that, they were asked to find 5 specific routes and to write down the departure and arrival times, and the total time was also recorded.

All questions in the questionnaire were answered on a five-point Likert scale, except for the open questions and explanatory questions. The subjects had to indicate whether they thought the FTI was easy to use, if they could find results faster using the FTI, and whether the results of the FTI were correct. They indicated whether or not the FTI was nicer and better, and explained why they thought so. There were two open questions, asking the subjects to indicate the most negative and the most positive aspects of the system. Finally, they indicated which system they preferred.

Analysis. We tested whether the task completion times of the FTI differed significantly ($p < 0.05$) from those of the complex web form, using the Paired Samples T-Test [7]. We also tested whether the five-point Likert scale values differed significantly from neutral (i.e. the number '3'), also using the T-Test.

Further, we evaluated the query formulation consistency by looking at the order of the information items. Each item was first replaced by a symbol as follows. We replaced the 'from' (location) with A, 'to' with B, 'via' with V, the 'date' with D, and the 'time' with T. For example, the input "from Amsterdam via Haarlem to The Hague, tomorrow at 10am." was represented as AVBDT. We then measured the correlation between the subject's query formulation and the task description using Kendall's τ [8]. Lastly, for each subject, we measured the average Kendall's τ over the combinations of the subject's formulations.

3.2 Results

The subjects. A total of 17 subjects (11 male, 6 female) participated in the study. The age distribution ranged from 21 to 66, with a mean of 32, as most subjects were between the age of 21 and 33. The background of the subjects ranged from (under)gradate students in various studies to people working in healthcare, consultancy, and IT-software development. The overall experiment (including the questionnaire) took around 30 minutes on average for each subject.

The questionnaire. Comparing the free-text interface (FTI) against the complex web form, the subjects indicated on a five-point Likert scale whether the FTI was: *faster* (**2.4**), *nicer* (**1.8**), *better* (**2.5**), and *preferred* (**2.0**). The numbers in parentheses are the average scores, where '1' indicates full agreement, and '5' denotes the opposite. All results were significant ($p < 0.05$) except for the third aspect, whether the FTI was better, which was not significant.

On average, the subjects felt they were a little faster at completing their tasks with the FTI compared to the complex web form. This was supported by the times measured for the web form and the FTI, with **7.3** and **6.7** minutes on average, respectively. The subjects were significantly ($p = 0.032$) faster, by about 8–9%, when using the FTI instead of the complex form.

Pros and cons. The subjects listed the most negative and most positive aspects of the FTI. The following **negative** aspects were mentioned: 24% of the subjects indicated that there was no example or short manual (obliging the subjects to 'just type in something, and it worked'); 18% indicated that the appearance of the interface was too simple, e.g. it lacked pictures; and 12% mentioned the fact that they had to 'clickthrough' to obtain the same results as with the complex web form. The following **positive** aspects were mentioned: 41% of the subjects liked the fact that the system 'understood' dates like tomorrow and Tuesday, and written time like 'ten past nine'; 41% liked the fact that you only had to type (without clicking on menus); 35% mentioned the query-suggestions as a useful feature; and 18% appreciated the fact that the input order of information items (e.g. time, date, places) did not matter.

Consistency. When considering only the order of the information items[1] in a query, there were 17 different query formulations. The three most frequent online query formulations were: ABDT 41%, ABVDT 15%, and, tied at third place with 6%, were ABTD, DTABV, and TABVD.

The mean Kendall's τ between the online task descriptions and the query formulations was **0.42**. The task with the highest average τ (0.96) was sequenced ABDT, the other tasks were BADT (0.67), TABVD (0.39), DTABV (0.09), and TBAD (-0.02). Two subjects always followed the same information order of the descriptions and had an average τ of 1.0 (though they used different wordings). Three subjects had an average τ between 0.6 and 1.0, and the rest of the seventeen subjects had an average less than or equal to 0.3.

The mean Kendall's τ for the (within subjects) online query formulations was **0.64**. Six subjects always formulated their questions in the same order and had an average τ of 1; six subjects averaged between 0.7 and 0.9; and, five subjects had an average τ less than 0.2.

Overall, the subjects were highly consistent in their query formulations. Further, the descriptions had little effect on the subjects' query formulations; the moderate correlation (0.42) is most probably an artifact of the fact that the subjects consistently formulated their queries as ABDT. (This explains the high correlations between the query formulations and the two tasks ABDT and BADT.)

4 Discussion

4.1 Methodology and Results

We tried to prevent the subjects from mindless copying of the task descriptions by presenting the tasks on paper instead of on screen. Nevertheless, the large

[1] i.e. the 'date' (D), 'time' (T), and the 'from' (A), 'to' (B), and 'via' (V) locations.

number of different query formulations we collected was surprising, since: *i*) the subjects could have just retyped the task descriptions; *ii*) there were only 17 subjects; and *iii*) the travel-planner web form was relatively simple. With so much query variation in this limited scenario (in both the order of information items and wordings used), even higher variation might be expected in a more complex scenario.

The paper&pencil-approach demanded manual time measurement. We measured the total time to complete all 5 search tasks, as it would be more difficult to obtain accurate measurements for individual tasks. Consequently, we could not determine whether the time per task decreased or not. Even though we noticed several times that subjects were clearly experimenting with the free-text interface during the tests (as they were talking out loud, saying 'what if I typed...'), the average time of the FTI is still significantly lower than that of the complex web form.

4.2 Limitations

We used shallow parsing to extract 'bags of key/value-pairs'. No grammar is used, therefore, a natural limitation is that we cannot parse nested structures. For example, we would not be able to parse nested mathematical equations, nor complex language constructs like "if X, what happens to Y and Z" (where X, Y, and Z are complex statements containing conjunctions and disjunctions). Note that this is not necessary since typical web forms do not allow such constructs either.

4.3 Specialized Features

In some cases, it could be handy to invoke a suitable function with the detected values as arguments. For instance, to extract the actual 'dd-mm-yyyy' time format from the input 'next week Friday', some function similar to 'getDate()' should be called to obtain the current date in order to calculate the intended date. The framework contains several pre-defined functions (e.g. for extracting dates and times) which can be invoked simply by specifying the field(s) that accept(s) a function value. Future versions of the framework will allow users to add user-defined functions.

4.4 Practicality of the Framework

For users. Our work has the potential for solving the deep web problem. Given a free-text search query, we can generate "real-time deep web results" (i.e. result snippets with valid query-URLs as data-entry points). Deep web search ultimately enhances our search experience by allowing users to search more content and to specify attribute or facet restrictions, besides merely a list of key words.

For providers. We believe that web companies will be encouraged to create their own configuration files for the following reasons: *i*) we showed that end users prefer such an interface; *ii*) (we claim that) it is easy to write a configuration file; and *iii*) visibility and user-friendliness are crucial for web companies.

Evidence for the last point can be found in a study by Alba et al. [9], where they observed that: (1) the revenues of websites depend on the data that users see on the site's web pages; (2) websites are extremely motivated to ensure correctness, accuracy, and consistency on the web pages shown to the end user; and (3) websites do not accord the same level of significance to the data delivered by the APIs. Alba et al. show that web companies care greatly for their 'public image', since: *i*) selling products or services is difficult if users do not know about you; and *ii*) users are more inclined to make a purchase if they feel positive about the website. Furthermore, the large number of articles on the web about search engine optimization strongly indicates that web companies are spending a lot to increase their visibility to the users.

Our free-text interface for searching over web forms has the potential to both increase the visibility of a web site (i.e. deep web search, enabling search over otherwise uncrawlable data) and to provide more user-friendly search interfaces for websites that implement this interface.

5 Related Work

A similar problem of filling out a web form for a given text query was tackled by Meng [10]. Meng used various statistical disambiguation techniques. However, a drawback of his statistical approach is that it requires (training) data that is often difficult to obtain (e.g. it requires domain-specific queries in order to obtain the 'global' statistics, and the data must often be annotated manually). Instead of statistical disambiguation, we scan for valid pattern combinations and present a ranked list of alternative interpretations to the user.

Weizenbaum described Eliza [11], one of the earliest systems with a natural language interface. The input text is parsed using decomposition rules triggered by keywords. Responses are generated based on reassembly rules pertaining to the decomposition rule. These rules are stored in a script which can be easily modified. During a session, a re-triggered decomposition rule may generate a different response. Unlike Weizenbaum, we generate responses depending on a set of detected patterns instead of a single decomposition rule, and we do not vary the responses. Recent keyword-based retrieval systems over structured data [12–16] generate a ranked list of structured queries or query interpretations, such that the user can select the right interpretation. However, most model the query as a bag of terms, disregarding the context of the extracted values, whereas we use patterns to capture the context. Further, they use a probabilistic or heuristic approach to rank the interpretations. Other grammar-based natural language interfaces have been developed [17–20]; however, the majority of these systems were application-specific which made it difficult to port the systems to different applications [21].

The difficulty of porting a system from one application (domain) to another is also apparent in information extraction systems, i.e. systems that extract all entities from large bodies of texts. To overcome the difficulty of porting, Appelt and Onyshkevych [4] propose the Common Pattern Specification Language (CPSL). At the heart of the CPSL grammar are the *rules*. Each rule has a priority, a pattern and an action. Input matched by the pattern part can be operated on by the action part of the rule. Ambiguity arises when multiple rules match at a given input location, and is resolved as follows: the rule that matches the largest part of the input is preferred, and if two rules match the same portion of the input, the rule with the highest priority is preferred. In case of equal priorities of matching rules, the rule declared earlier in the specification file is preferred. Like Appelt and Onyshkevych, we propose a pattern specification language, and the patterns are used to scan the input text. However, we generate interactive query suggestions and we produce a ranked list of interpretations instead of a single interpretation.

6 Conclusion and Future Work

We introduced a novel specification language for describing a free-text interface (FTI) to complex web forms. Our system uses patterns to scan the user input and extract 'bags of key/value-pairs'. The system is capable of both generating query suggestions on the fly and generating ranked query interpretations.

We carried out a user study to compare the FTI with an existing travel planner web form. Our results showed that the subjects were significantly faster at finding information when using the FTI instead of the complex form by about 8–9%. Furthermore, they significantly preferred using the FTI over using the original web form. The results also showed that the subjects were highly consistent in their query formulations, and that there was considerable query variation between subjects, even in such a limited scenario.

In future work we will investigate whether configuring the FTI is really easy or not, by building FTIs in different domains and analyzing the builders' opinions about the configuration process. Also, we will investigate optimizations of the parsing process, and examine different ways to combine patterns of multiple websites in one domain.

Acknowledgments This research was supported by the Netherlands Organization for Scientific Research, NWO, grants 639.022.809 and 612.066.513.

References

1. Sun, J., Bai, X., Li, Z., Che, H., Liu, H.: Towards a wrapper-driven ontology-based framework for knowledge extraction. In: KSEM'07, Berlin, Heidelberg, Springer-Verlag (2007) 230–242
2. Kaufmann, E., Bernstein, A.: Evaluating the usability of natural language query languages and interfaces to semantic web knowledge bases. Web Semantics: Science, Services and Agents on the World Wide Web (2010)

3. Papakonstantinou, Y., Gupta, A., Garcia-Molina, H., Ullman, J.D.: A query translation scheme for rapid implementation of wrappers. In: DOOD'95, London, UK, Springer-Verlag (1995) 161–186
4. Appelt, D.E., Onyshkevych, B.: The common pattern specification language. In: Proceedings of a workshop on held at Baltimore, Maryland, Morristown, NJ, USA, Association for Computational Linguistics (1996) 23–30
5. Madhavan, J., Ko, D., Kot, L., Ganapathy, V., Rasmussen, A., Halevy, A.: Google's deep web crawl. Proc. VLDB Endow. **1**(2) (2008) 1241–1252
6. White, R.W., Marchionini, G.: Examining the effectiveness of real-time query expansion. Information Processing and Management **43**(3) (2007) 685–704
7. Kutner, M.H., Nachtsheim, C.J., Neter, J., Li, W.: Applied linear statistical models. 5th edn. McGraw-Hill (2005)
8. Kendall, M.: Rank Correlation Methods. 4th edn. Second impression. Charles Griffin (1975)
9. Alba, A., Bhagwan, V., Grandison, T.: Accessing the deep web: when good ideas go bad. In: OOPSLA Companion '08: Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications, New York, NY, USA, ACM (2008) 815–818
10. Meng, F.: A natural language interface for information retrieval from forms on the world wide web. In: ICIS, Atlanta, GA, USA, Association for Information Systems (1999) 540–545
11. Weizenbaum, J.: Eliza—a computer program for the study of natural language communication between man and machine. Commun. ACM **9**(1) (1966) 36–45
12. Demidova, E., Fankhauser, P., Zhou, X., Nejdl, W.: Divq: diversification for keyword search over structured databases. In: SIGIR '10, New York, NY, USA, ACM (2010) 331–338
13. Tran, T., Cimiano, P., Rudolph, S., Studer, R.: Ontology-based interpretation of keywords for semantic search. In: ISWC'07/ASWC'07, Berlin, Heidelberg, Springer-Verlag (2007) 523–536
14. Zhou, Q., Wang, C., Xiong, M., Wang, H., Yu, Y.: Spark: adapting keyword query to semantic search. In: ISWC'07/ASWC'07, Berlin, Heidelberg, Springer-Verlag (2007) 694–707
15. Tata, S., Lohman, G.M.: Sqak: doing more with keywords. In: SIGMOD'08, New York, NY, USA, ACM (2008) 889–902
16. Kandogan, E., Krishnamurthy, R., Raghavan, S., Vaithyanathan, S., Zhu, H.: Avatar semantic search: a database approach to information retrieval. In: SIGMOD'06, New York, NY, USA, ACM (2006) 790–792
17. Burton, R.R.: Semantic grammar: An engineering technique for constructing natural language understanding systems. Technical report, Bolt, Beranek and Newman, Inc., Cambridge, MA. (December 1976)
18. Hendrix, G.G., Sacerdoti, E.D., Sagalowicz, D., Slocum, J.: Developing a natural language interface to complex data. ACM TODS **3**(2) (1978) 105–147
19. Carbonell, J.G., Boggs, W.M., Mauldin, M.L., Anick, P.G.: The XCALIBUR project: a natural language interface to expert systems. In: IJCAI'83, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (1983) 653–656
20. Carbonell, J.G., Hayes, P.J.: Dynamic strategy selection in flexible parsing. In: Proceedings of the 19th annual meeting on ACL, Morristown, NJ, USA, Association for Computational Linguistics (1981) 143–147
21. Androutsopoulos, I., Ritchie, G.D., Thanisch, P.: Natural language interfaces to databases – an introduction. Natural Language Engineering **1**(01) (1995) 29–81