Free-Text Search over 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 using free-text queries 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, free-text interfaces, query translation.

1 Introduction

The internet contains a large amount of information that is only accessible through complex web forms. Journey planners, real estate websites, online auction and shopping websites, and other websites commonly require the user to fill out 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. 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 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 can search faster with an FTI than

with a complex web form, and that they prefer the FTI over the complex 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 Sect. 3. Sections 4 and 5 discuss our work and overview related work. Finally, Sect. 6 concludes our work.

2 A Free-Text Interface to Web Forms

2.1 Requirements

The goal of our FTI is to offer simple textual access to content behind a web form that consists of multiple input fields and options. Given a user's query as free-text input, the FTI should display a ranked list of plausible interpretations (i.e. ways to fill out the complex web form) as results. Kaufmann and Bernstein showed the importance of guiding the user during the query formulation process [2]. Therefore, the FTI should offer query suggestions as a means to guide users in formulating their queries. Lastly, it should be easy for developers to specify the capabilities of the FTI. Examples of a complex web form for a travel planning website and an FTI to the same form, are given in Figs. 1 and 2, respectively.

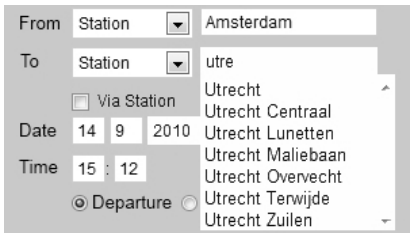

Fig. 1. A complex web form that offers interactive query suggestions, based on the Dutch Railways site.

Fig. 2. Trajectvinder ('Route Finder'): an FTI that offers interactive query suggestions, tailored to the complex web form.

2.2 Framework

On a high level, our FTI involves three processes: the query interpretation process, the query suggestion process, and the result generation process.

Basic query interpretation. Let us first define the following: A *region* is a contiguous segment or sequence of characters of the user's input. A *pattern* expresses relevant information (as prefix cues, types, and postfix cues; these are discussed later). An *annotation* is a label assigned to a region, denoting that the region contains some (part of a) pattern. Finally, an *interpretation* is a set of annotated regions. The query interpretation process consists of 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, all matching patterns starting from that word are stored. Pattern matching is greedy, meaning a pattern will match as much of the query as possible. This process yields a set of possibly overlapping input *regions*, where each region could have several annotations. (A region could be matched by multiple 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 Γ, which contains sets of non-overlapping regions with maximal coverage of the input, is generated;
3. *Generating interpretations:* for each region set $\gamma \in \Gamma$ all combinations of annotations are generated. This yields the set of possible interpretations;
4. *Filtering interpretations:* first, interpretations are 'cleaned' by removing extraneous annotations. Examples of extraneous annotations are prefix annotations that precede annotations which are not specified by the pattern to which prefix corresponds, and annotations that exceed the number of times they are allowed to appear in the underlying web form. Second, interpretations that are completely subsumed by other interpretations are removed;
5. and,*Ranking interpretations:* the interpretations are first ranked by the number of annotations they contain, then by the order in which the patterns are specified in the configuration file.

Steps one to four are illustrated in Fig. 3. Say, we are given the query "Wycombe to shopping paradise Bicester North Camp". Further, we have two simple patterns p_1 (*from* `station`), and p_2 (*to* `station`). Here, *from* and *to* are optional prefix tokens, and `station` is a type which denotes a set of tokens. In this example, the only valid tokens of the type `station` are 'Wycombe', 'Bicester North', and 'North Camp'. Lastly, each pattern may occur once or not at all.

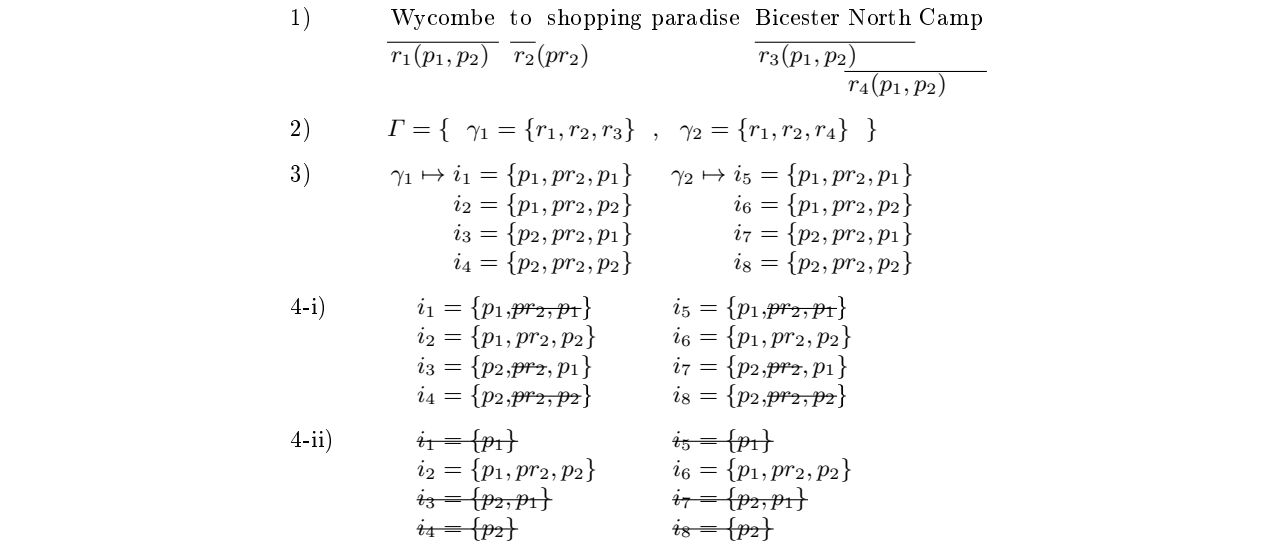

Fig. 3. An illustration of the basic interpretation process.

Step 1 underlines the regions of the input containing known tokens. The first region r_1 is matched by both patterns p_1 and p_2, and is annotated as such. Region r_2 contains the prefix of pattern p_2, annotated as pr_2. The overlapping regions r_3 and r_4 are split in step 2, yielding the non-overlapping sets of regions γ_1 and γ_2. For each set $\gamma \in \Gamma$, step 3 generates all possible annotation-combinations. Such a combination is in fact an interpretation, thus step 3 yields the interpretations $i_1 \ldots i_8$. Step 4-i removes the erroneous annotations in each interpretation, in this case, it removes the prefix pr_2 when it is not followed by the pattern p_2, and it removes an annotation if it already occurred. In step step 4-ii, an entire interpretation is removed if it is a subset of any other interpretation. At the end of step four, we are left with two interpretations i_2 and i_6, denoting "from Wycombe to Bicester North", and "from Wycombe to North Camp", respectively.

Generating suggestions. The suggestion process is an extension of the basic interpretation process, and generates three types of query suggestions: token expansions, pattern expansions, and relation expansions. The suggestions are interactive query expansions [6]; they are generated based on the last region, and filtered based on the entire interpretation. When the last region denotes a prefix of known tokens, all applicable token expansions are shown. When the last region denotes a complete prefix of a 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 region denotes 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. Finally, when the last region contains a token (like a car brand) for which there are related tokens (like the corresponding car models), then those tokens are shown.

Generating results. The result generation process is also an extension of the basic interpretation process; each interpretation is post-processed as follows. First, the default values for all fields that were not specified by the user are added to the interpretation. For instance, by default the current time could be added to the example query given earlier. Second, the interpretation is discarded if it does not satisfy all constraints in the FTI's configuration. Third and finally, a result snippet is generated according to the FTI's configured rules (described in the next sections).

2.3 Configurable Items

Our 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 user friendly as it generated a false positive and it may have wasted the user's valuable time. Simply scanning user 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, one 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 can 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. 4), 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 of the snippet, the web form's action-URL, the http-request method (i.e. get or post), and all the form's input fields must be specified. In the next section, we show how these configurable items fit together.

Fig. 4. Interpretation as result snippets.

2.4 An Example Configuration

Figure 5 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 `pattern` el-

```
<?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;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. 5. An example configuration file.

ement's `id` attribute contains the name of the input field to which the captured value should be assigned. A `capture` element specifies the *type* to be captured. The `prefix` and `postfix` elements specify a finite list of strings. This list may be specified by fully writing out all possibilities, or by a Kleene star-free regular expression, which will be automatically expanded to the list of possible strings. A pattern's `option` element relates a particular `prefix` with a particular `postfix`. The use of options is portrayed in the following example. Consider an input field to enter some minimum mileage, and three prefix-capture-postfix

combinations: "minimum ... kilometers", "minimum number of kilometers ...", and "minimum number of kilometers ... kilometers". The latter of these combinations is peculiar and it would be parsed if we specified just one pattern option consisting of: "<prefix>minimum(number of kilometers)?</prefix>" and "<postfix>kilometers</postfix>". Moreover, the system would also generate the postfix suggestion "kilometers" if it parsed "minimum number of kilometers ...". To prevent this behavior, we could specify two options, one containing "<prefix>minimum</prefix>" and "<postfix>kilometers</postfix>", and one containing just the prefix "<prefix>minimum number of kilometers</prefix>". The `constraints` element 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. An interpretation is valid if it satisfies at least one of the mandatory field combinations, and all field comparisons. Lastly, the `results` element specifies what the interpretation's title, description, and URL should look like. Here, a developer could also specify 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 (as well as the 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 complex 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 part, an online part, and a questionnaire. The information needs were randomly generated, and shown either graphically as a route on a map; or textually as a random sequence of (two or three) station names, a date, and a time. Dates were either

relative, such as "next week Wednesday", or absolute, such as "1-2-2011". Times were descibed either alphabetically, such as "half past ten", or numerically, such as "17.30". 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 route on a map, 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. Likewise, but based on a different information need, they filled out the FTI 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 graphical instead of textual descriptions of the information need. During the online part, the task was to search for specific routes. 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*). The subjects first familiarized themselves with the complex interface of the existing travel planner site. Then, they searched for 5 specific train routes and wrote down the departure and arrival times, while we recorded the total time to find all routes. Next, the subjects familiarized themselves with the FTI. After that, they searched for 5 specific routes and wrote down the departure and arrival times, and we recorded the total search time. Regarding the questionnaire, all questions were answered on a five-point Likert scale, except for the open questions and explanatory questions. The subjects indicated 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 rank 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 that subject's own 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 (median: 27, mean: 32); most subjects were between the age of 21 and 33. The background of the subjects ranged from (under)graduate students in various studies to people working in healthcare, consultancy, and IT-software development. Participation (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*, *nicer*, *better*, and *preferred*. The results are given in Table 1, where '1' indicates full agreement, and '5' denotes the opposite. All results differed significantly ($p < 0.05$) from neutral, except for the third aspect. On average, the subjects felt they could search a little faster using the FTI than using the complex web form. This was supported by the times measured for the web form and the FTI, shown in Table 2. The subjects were significantly ($p = 0.032$) faster, by about 9%, when using the FTI instead of the complex form.

Table 1. Average results of the questionnaire, comparing the FTI to the complex web form on a five point Likert-Scale from 1 (full agreement) to 5 (full disagreement). Results in bold are significant ($p < 0.05$).

Question	Score
Faster	**2.4**
Nicer	**1.8**
Better	2.5
Preferred	**2.0**

Table 2. Average time in minutes to complete all five search tasks for each interface. The results differ significantly ($p = 0.032$).

Interface	Average time in minutes
Free-text interface	6.7
Complex web form	7.3

Speed and success rate. We counted the number of incorrect routes reported by the subjects. Out of the 170 answers, 14 were wrong: 6 errors were made using the FTI, and 8 using the complex form. The most likely explanation for the errors is that the subjects misread the task, and entered a wrong time or station name. Out of the 17 subjects, 10 subjects made zero errors, 2 subjects made one error, 3 subjects made two errors, and 2 subjects made three errors. However, since we did not measure the time per query, we cannot omit the times for the failed queries for comparing the two systems. Yet if we would use only the data from the 10 subjects who made no erros, there would still be a 9% difference in time, in favor of the FTI.

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 (forcing the subjects to

'just type in something, and it worked'); 18% indicated that the interface was too simple, e.g. it lacked pictures; and 12% disliked that they had to explicitly click a result snippet to view the travel plan, even when only a single result snippet was returned. The following **positive** aspects were mentioned: 41% of the subjects liked how the system 'understood' dates like tomorrow and Tuesday, and written time like 'ten past nine'; 41% liked 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. As can be seen in Fig. 6, the three most frequent online query formulations were: ABDT 41%, ABVDT 15%, and, tied at third place with 6%, were ABTD, DTABV, and TABVD.

Now we inspect whether or not the subjects formulated their queries with the same order of information items as that of the online task descriptions. 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 four 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 task 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 remaining twelve 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, regardless of the task description, 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.

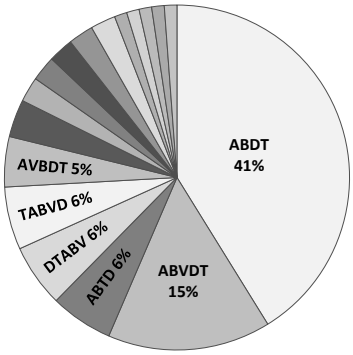

Fig. 6. Distribution of the most frequent online query formulations

Overall, the subjects were highly consistent in their query formulations individually; however, there was considerable query variation between subjects. Further, the task descriptions had little effect on the subjects' query formulations; the moderate correlation (0.42) is most probably an artifact caused by subjects consistently formulating their queries as ABDT. This explains the high correlations between the query formulations and the two tasks ABDT and BADT.

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

4 Discussion

4.1 Methodology and Results

Query variation. 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 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.

Time difference. 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 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 developers to add new functions.

4.3 Practicality of the Framework

For users. Our work could add to the solution of 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. Our work may benefit other search environments as well, particularly when there is some sort of semi-structured search involved, examples could be desktop search and intranet search.

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*) online 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 make serious investments 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. In the context of keyword-based retrieval systems over structured data, one of the earliest systems was DataSpot [12]. The DataSpot system used free-form queries and navigations to explore a *hyperbase* (a graph of associated elements) for publishing content of a database on the Web. Recent systems [13–17] 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 [18–21]; however, the majority of these systems were application-specific which made it difficult to port the systems to different applications [22]. 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 9%. Furthermore, they preferred the FTI over the complex web form. The results also showed that the subjects were highly consistent in their individual query formulations, and that there was considerable query variation between subjects, even in such a relatively simple scenario.

In future work we will investigate whether configuring the FTI is really simple 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.

Acknowledgment

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. Dar, S., Entin, G., Geva, S., Palmon, E.: Dtl's dataspot: Database exploration using plain language. In: Proceedings of the 24rd International Conference on Very Large Data Bases. VLDB '98, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (1998) 645–649
13. 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
14. 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
15. 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
16. Tata, S., Lohman, G.M.: Sqak: doing more with keywords. In: SIGMOD'08, New York, NY, USA, ACM (2008) 889–902
17. 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

18. 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)
19. 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
20. 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
21. 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
22. Androutsopoulos, I., Ritchie, G.D., Thanisch, P.: Natural language interfaces to databases – an introduction. Natural Language Engineering **1**(01) (1995) 29–81