# Table of Contents

# Form Identifying

## 1. Introduction

A lot of web sites use the interface consisting of an HTML *form* to let users submit their queries. Figure 1 shows such a typical HTML form. It will be helpful for us to perform automatic information collection if we can identify or recognize each field of these HTML forms and feed the user's query into the corresponding *input field*. However, the fact that different web sites might use HTML forms with different layouts and compositions because of the unstructured nature and flexibility of HTML syntax makes the identification process not that easy.
Since it is infeasible to specifically build a different information collector for every particular information source, we proposed a solution to the form identifying by utilizing the structural and semantic information in this project, and based on this solution we construct the architecture with a user feedback mechanism to achieve our identification goal.

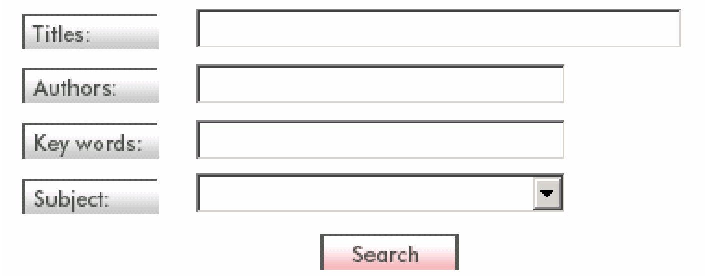


Figure 1 A typical HTML form

In particular, the domain we are focusing on is electronic paper searching interfaces in the form of HTML forms. For the purpose of this project, we intend to recognize or identify input fields such as "Title", "Author", "Date", and "ISSN" from the analyzed form.

## 2. Related work

Presently, few people are doing research on *HTML Form Recognition or Extraction*, although it in fact is promising for extensive applications. Currently, there exist some HTML Form Management Systems such as LiquidOffice developed by Cardiff Corp (http://www.cardiff.com/LiquidOffice). But most of them are doing form management by just specifying the semantic of each form manually, not doing the semantic recognition (semi-)automatically.

This report is organized as follows. In next section, we will discuss the physically basic elements of an HTML form. Based on this, we then discuss the logical structure of a form in section 4. Section 5 describes the whole process of the form identifying. A conclusion and the future work are given in section 6.

## 3. Basic elements in an HTML from

From the point of view of HTML syntax, a form may physically consist of the following three types of basic elements. These elements may serve as different logic or semantic roles in the form, which is discussed later in section 4.

**1) Plain text**

A *plain text* is the pure text we see from the HTML browser. But in HTML representation, a piece of plain text usually has markups around to decorate itself. For example,

```
<span color="red"><b>Author</b></span>.
```

We can always filter the *plain text*, like `Author` in this example, out of the surrounding markups.

**2) Input tag**

In HTML representation, an *input tag* usually contains three attributes such as *type*, *name*, and *value*. Below is an example of an *input tag*.

```
<input type="text" name="title" value="…">
```

The value of the *type* attribute might be "text", "checkbox", "radio", "image", "hidden", "submit", "reset", "button", and "password". We will only look at the first three types, since the latter ones are seldom or never used for the user to input data in a form. Sometimes, the *type* attribute is absent in an *input tag*. In this situation, it is by default of type "text".

In most cases, the value of the *value* attribute is blank or absent. Even if it exists, it just means the initial value of this *input tag* when the form is loaded into the HTML browser. In practice, the user will always fill his/her query data into an *input tag*, not using its initial value.

**3) Select tag**

A *select tag* also has the *name* attribute. But in stead of having the *value* attribute, its value is a list of *option tags*. For example,

```
<select name="year"> <option value="1999">1999 <option …> … </select>
```

Each *option tag* has a *value* attribute and a corresponding text. The text is what the user will see in the HTML browser.

We notice that the semantic of an element might be derived from **three** sources:

1. The *plain text* beside an *input* or *select tag*;
2. The values of the name attribute of an *input* or *select* tag.
3. We also notice that a *select tag* contains a list of more or less values (*option tags*), which may present certain features like "the value of a year field is sometimes a 4-digit number".

Based on this, we construct a *meta-dictionary* and a *value-feature base* for identifying each field of the form by deriving semantics from the above three sources in that form. In turn, knowledge of the *meta-dictionary* and the *value-feature base* can also be extended from these three sources during the process of recognition and the user's feedback. We will discuss this in detail later in this report.

## 4. Logic structure of an HTML form

In section 2, we discussed the basic elements that are included in a form. Logically, these tags may act as different roles in a form. For instance, a *plain text* is often used to convey semantic of a field in a form. In this section, we will discuss the logic structure of an HTML form and see how each logic part uses the basic tags in section 2.

Figure 2 is the hierarchical logic structure of an HTML form. Typical examples for each element of the structure are given below it.

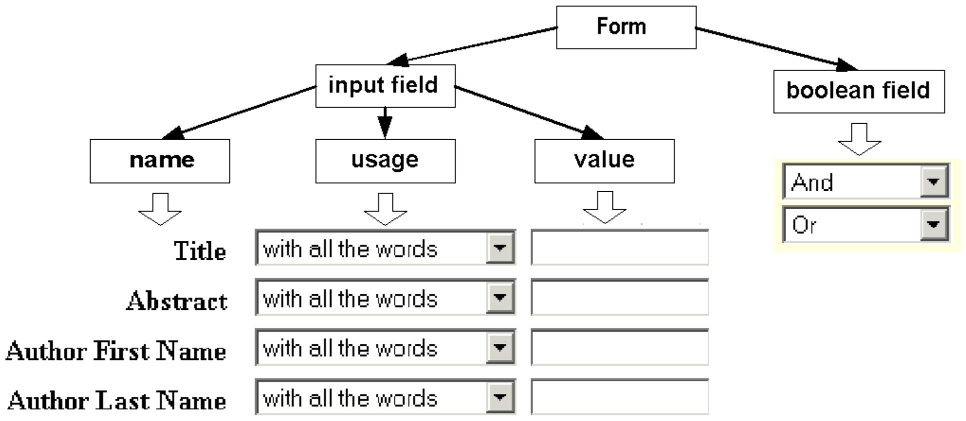


Figure 2 Hierarchical logic structure of an HTML form

In most cases, an HTML form logically contains a list of *input fields* combined by means of *Boolean fields.*

**1) *Boolean field*:** stands for the Boolean relationships between *input fields*. Usually, this field could be represented as either a *select tag* (like the example in Figure 2), or just *blank* (as shown in Figure 1). The *Boolean field* in the latter situation generally means "AND" operation.

**2) *Input field*:** An input field stands for a semantically complete field for the user to input the query data. It consists of three logic properties such as *name*, *usage*, and *value*. **These properties may have different HTML representations (i.e. the basic tags in section 3) from each other, and even a single property may have various HTML representations in different situation.**

- ***Name***: conveys the semantic of an *input field*. A *name* property can be in the form of a piece of *plain text* like "Title", "Abstract", and "Author First Name" in Figure 2.
  It also can be a *select tag* as shown in Figure 3. The *select tags* used as the *name* properties often present such a **characteristic** as: they have the same *value lists* (*option* tags), but have different initial values, which are the (semantic) names of different *input fields*.

For example, the form in Figure 3 has three *select tags*, whose *value lists* are all `{“author”, “Title”, “Anywhere in Article”}`, respectively serving as the *name* property of three *input fields*.

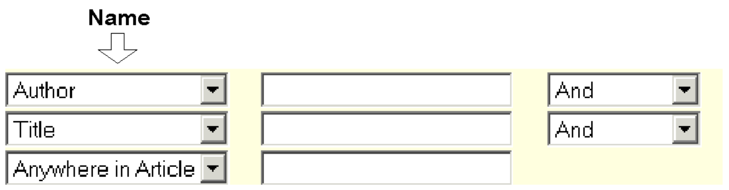


Figure 3 an example of the *name* property in the form of a *select tag*

- ***Usage***: is usually represented in a *select tag* as shown in Figure 3.1. The *usage* property is used to specify the usage of the query data inputted by the user. For example, the “*with all the words*” means that **only the documents containing all the words of the query data (inputted by the user as the *value* property) will be taken as matched results and returned to the user**.
  In many cases, the *usage* property is absent in an *input field*, which means exactly the same thing as what the “*with all the words*” means.
  The *select tags* used as the *usage* properties also present such a **characteristic** as: they have the same *value lists* (*option* tags), and have the same initial values. For example, the four select tags in Figure 2 all have the same initial values as “*with all the words*”.

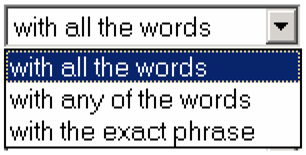


Figure 3.1 an example of *usage* property

- ***Value***: represents the value of an *input field*. This value is in fact the query data that the user inputs by means of HTML tags such as *input tags* (see Figure 2) or *select tags* (see Figure 4).

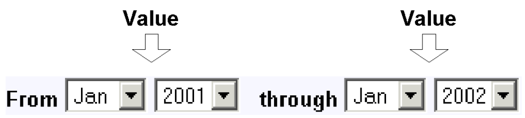


Figure 4 an example of the *value* property in the form of *select tags*

## 5. Implementation of Form Identifying

Figure 5 shows the architecture of the *form identifying* system. Roughly speaking, the whole process consists of four steps such as *filtering*, *partitioning*, *recognizing*, and *feedback*. And two simplified knowledge bases are used in our architecture, which are the *Meta-Dictionary* and the *Value-Feature*. They can be extensible based on the user’s feedback.

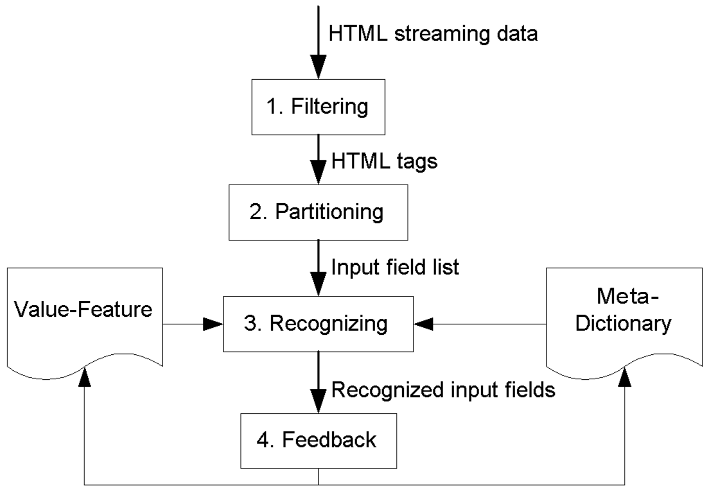


Figure 5 Architecture of the system of *Form Identifying*

In brief, the task of each step is as follows:

- *Filtering*: is mainly to analyze an HTML page in the form of a data streaming, and filter the tags that don't contribute to the HTML form(s), and then return a list of three types of basic tags (see section 3).
- *Partitioning*: is to utilize the syntax feature (i.e., the logic structure discussed in section 4) of the form to partition all the tags got from *Filtering* into pieces of *input fields*.
- *Recognizing*: is to utilize the *Meta-Dictionary*, the *Value-Feature*, and the *Trial-feedback* method to recognize the semantics of each *input field* got from the previous step.
- *Feedback*: means the system adjust itself according to the feedbacks from the user.

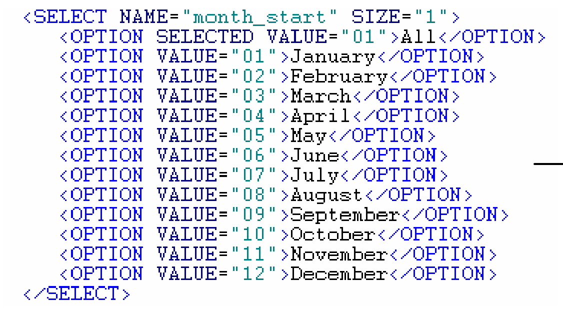
In the following, we will discuss each step in detail by using concrete examples.

**1) Filtering**

The IE system reads from an HTML file or an URL the HTML streaming data that contains one or more HTML forms. By analyzing this streaming data, the system will filter away all the useless HTML tags, just retaining the three basic elements (i.e., *plain text*, *input tags*, and *select tags*).

After this, the system will standardize these basic tags, including:

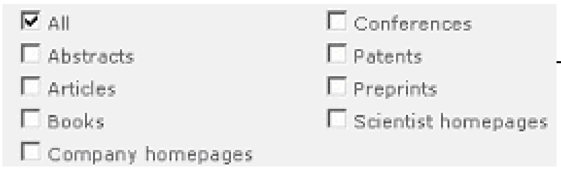
- Getting rid of redundant spaces and html tags across the *plain text*. For example, `<TD ALIGN="right"><B>Title: &nbsp;</B></TD>` ➔ `Title`
- Constructing a *value list* for each *select tag* by taking the text in each of its *option tag* as a value in this *value list*. For example,



```
<SELECT NAME="month_start" SIZE="1">
   <OPTION SELECTED VALUE="01">All</OPTION>
   <OPTION VALUE="01">January</OPTION>
   <OPTION VALUE="02">February</OPTION>
   <OPTION VALUE="03">March</OPTION>
   <OPTION VALUE="04">April</OPTION>
   <OPTION VALUE="05">May</OPTION>
   <OPTION VALUE="06">June</OPTION>
   <OPTION VALUE="07">July</OPTION>
   <OPTION VALUE="08">August</OPTION>
   <OPTION VALUE="09">September</OPTION>
   <OPTION VALUE="10">October</OPTION>
   <OPTION VALUE="11">November</OPTION>
   <OPTION VALUE="12">December</OPTION>
</SELECT>
```

```
HTMLTag tag{
        int type = HTMLTag.SELECT;
        String name= " month_start";
        ArrayList valueList = {"January",
        "February", "March", "April", "May",
        "June", "July", "August", "September",
        "October", "November","December"};
};
```

- Generating a *select tag* to replace neighboring *radio* or *checkbox input tags*, by taking the text of these *radio* or *checkbox input tags* as the *value list* of the *select tag*.



☑ All ☐ Conferences
☐ Abstracts ☐ Patents
☐ Articles ☐ Preprints
☐ Books ☐ Scientist homepages
☐ Company homepages

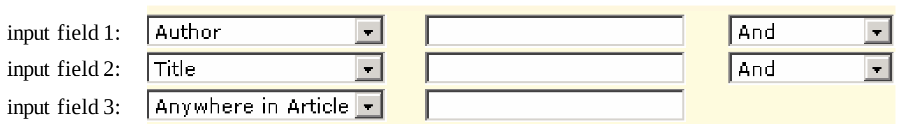
```
HTMLTag tag{
        int type = HTMLTag.SELECT;
        String name= " month_start";
        ArrayList valueList = {"All", "Conference",
        "Abstracts", "Patents", "Articles",
        "Preprints", "Books","Scientist homepages",
        "Company homepages"};
};
```

Thus the result of *filtering* is a list of standardized HTML tags.

## 2) Partitioning

After filtering away all the useless HTML tag and text, we get a form consisting of a list of standardized HTML tags such as *plain text*, *select tag*, and *input tag* (*type=text*). Now, we can utilize the following logic structure (syntax) features of a form to partition the form into *input fields* (which so far are not recognized semantically) for the use of the next step (*recognizing*).

- ***Boolean fields***: A *Boolean field* is usually in the form of *select tag* with *value list* as "AND", "OR", and "NOT". For example, in Figure 6, the form can partitioned by the *Boolean fields* (the *select tags* displayed as "And") into three *input fields*.

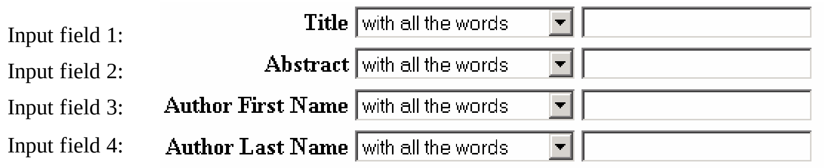


| | | | |
|---|---|---|---|
| input field 1: | Author | | And |
| input field 2: | Title | | And |
| input field 3: | Anywhere in Article | | |

Figure 6 Input fields partitioned by Boolean fields

- ***Name* property**: In the case of a form doesn't contain explicit *Boolean fields*, we can use the *name* property of the form to partition the form, since an *input field* must contain a *name* property. But we have to handle the two different situations when the *name* property is a *plain text* or a *select tag* (see the discussion in section 4).
    a) *Plain text*: For example, the form in Figure 7 is partitioned into four *Input fields* by *plain texts* such as "Title", "Abstract", "Author First Name", and

"Author Last Name".



Input field 1: **Title** with all the words

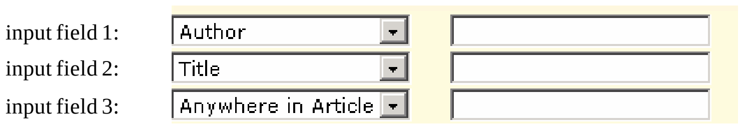
Input field 2: **Abstract** with all the words

Input field 3: **Author First Name** with all the words

Input field 4: **Author Last Name** with all the words

Figure 7 Input fields partitioned by plain text

b) *Select tag*: we can utilize the **characteristic** (see the discussion in the section "logic structure of an HTML form") to partition the form, even without recognizing its semantics.



input field 1: Author

input field 2: Title

input field 3: Anywhere in Article

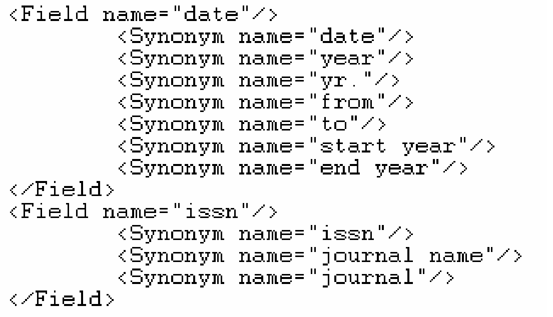
Figure 8 Input fields partitioned by select tags

## 3) Recognizing

After we get a form partitioned into pieces of *input fields*, we apply subsequently three methods to recognize the semantic of each *input field*, which are the *Meta-dictionary*, *Value-features*, and *Trial-feedback* methods. Give an extraction *objective* such as "Date", "Author", "ISSN" or "Title" etc., we initially assign 0 to the value of the similarity between the objective and the current *input field*. If any of the above three methods recognize the *input field* similar to the *objective*, we will increase the similarity by 1.

- **Meta-dictionary**

The *Meta-dictionary* contains a *synset* (set of synonyms) for each category (semantics of *input fields*) such as "Date", "Author", "ISSN" and "Title" etc. We call these categories as extraction ***objectives***. Figure 9 is a fragment of the *Meta-dictionary* at one moment in the system. We can see that the *objective* "Date" has currently seven synonyms. Physically the *Meta-dictionary* is represented as a file of XML syntax.



```
<Field name="date"/>
        <Synonym name="date"/>
        <Synonym name="year"/>
        <Synonym name="yr."/>
        <Synonym name="from"/>
        <Synonym name="to"/>
        <Synonym name="start year"/>
        <Synonym name="end year"/>
</Field>
<Field name="issn"/>
        <Synonym name="issn"/>
        <Synonym name="journal name"/>
        <Synonym name="journal"/>
</Field>
```

Figure 9 a fragment of the Meta-dictionary

When we utilize the Meta-dictionary to analyze an *input field* with a given extraction *objective* such as "Title" or "Date", we will see:

i) If the *plain text*, which acts as the ***name*** property of the *input field*, matches with the corresponding *synset*.

ii) If the *select tag*, which acts as the ***name*** property of the *input field*, has the initial value that matches with the corresponding *synset*.

iii) If the value of the *NAME* attribute of the *input tag* or *select tag*, which acts as the ***value*** property of the *input field*, matches with the corresponding *synset*.

For each of the above matches that are satisfied, we will consider the semantic of the current *input field* similar as *objective*, and increase the value of the similarity between the *input field* and the *objective* (initially 0) by 1. For example, in the *input field* in Figure 10, the *plain text* "From" matches with the synset "date" in the Meta-dictionary (see Figure 9), then this *input field* is similar to the *objective* "date".

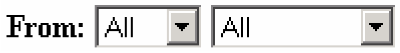


Figure 10 an example of recognition using the Meta-dictionary

- **Value-feature**

Here, the *Value-feature* means the features of the *value list* of the *select tags* in a form. Each *Value-feature* has 3 attributes such as *datatype*, *length*, and *pattern*.

i) *datatype*: identifies the data type of the *value list* of a *select tag*. Its value may be "number", "char", and "hybrid" (means either number or char).

*ii)* *length*: is the length of each value in the *value list* of a *select tag*.

*iii)* *pattern*: means the pattern the values of the *value list* of a *select tag* presents. For example, 'pattern="****-****"' in Figure 11 means the 5th element should be '-' with the left 8 elements being character or number.

Figure 11 is a fragment of the *Value-feature* knowledge base, which is also represented as a XML syntax file. We can see that the *objective* "date" has a data type of "number" and the length of four, and its pattern being four continuous digits. An "ISSN" *input field* may have a "hybrid" type, which means a mixture of digits and characters.

```
<issn datatype="hybrid" length="9" pattern="****-****"/>
<date datatype="number" length="4" pattern="****"/>
```
Figure 11 a fragment of the Value-feature

If any *select tag* in an *input field* matches with the *Value-feature* of a given *objective* (i.e., all the values of the *select tag* satisfy the 3 attributes of this *Value-feature*), we will consider the semantic of the current *input field* similar as *objective*, and increase the value of the similarity between the *input field* and the *objective* by 1.

For example, in the *input field* in Figure 12, the *select tag* right behind the "From" matches with the *Value-feature* of "date" (see Figure 9), thus this *input field* is similar to the *objective* "date".

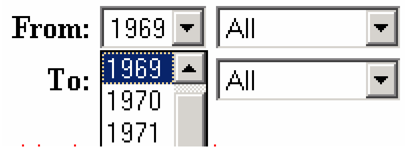


Figure 12 an example of recognition using the Value-feature

- **Trial-feedback**

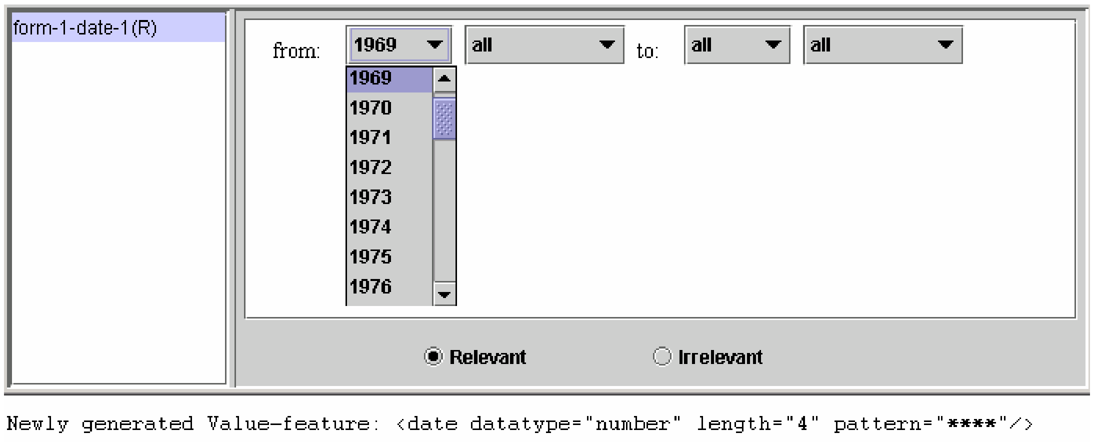
Here, the *Trial-feedback* method means, given an extraction *objective* and the current *input field*, we insert the typical testing data for this *objective* (e.g., "1234-32x3" for the objective "ISSN") into an *input tag* in that *input field* or a *select tag*, then we submit this form to get the results that are supposed to match the testing data for the *objective*. By analyzing the returned results, we recognize the semantic of the current *input field*.
If the number of the testing data contained in the returned results exceeds a predefined *threshold*, we will consider the current *input field* similar to the *objective*, by increasing its similarity by 1. Sometimes, a same testing data matches with different *input field*. In this case, we will pick the one with the return results containing the biggest number of the testing data.
The Trial-feedback is not implemented in our current demo system for some technical reason.

After recognized by the above three methods subsequently, for each *objective* the system may obtain a list of *input field* that are similar to the *objective* with different similarity. Then the system sorts the results in the descending order of similarity and returns them to the user for him/her to feedback.

- **User feedback**

After partitioning and recognizing *input field* from a HTML form, the results will be shown to the user. The user will do feedback according to the relevance of the result to the *objective*. The user may perform the following four different types of feedbacks.
**1) Relevant:** When the user specifies a result is relevant to the *objective*, the IE system will do a *positive* feedback, which includes the following five actions:

i. As to the ***plain text*** acting as the ***name*** property of the *input field*, add it into the corresponding *synset* in the *Meta-dictionary* if this *plain text* doesn't exist previously in the *synset*. If it exists before, increase its weight. (Although so far, we don't associate a synonym with a weight for the reason of simplifying the implementation.)
ii. As to the ***select tag*** acting as the ***name*** property of the *input field*, add its initial value into the corresponding synset in the *Meta-dictionary* in the same way as i).
iii. As to the ***input tag*** acting as the ***value*** property of the *input field*, add the value of the *name* attribute of this *input tag* into the corresponding *synset* in the *Meta-dictionary* in the same way as i).
iv. As to the ***select tag*** acting as the ***value*** property of the *input field*, add the value of the *name* attribute of this *select tag* into the corresponding *synset* in the *Meta-dictionary*, meanwhile try to generate a new *Value-feature* item for this *objective* and add it into the *Value-feature*. For example, in Figure 13, the user chooses the "Relevant" as to the *input field* that is similar to the *objective* "date". Then an item of *Value-feature* shown in the figure will be generated.

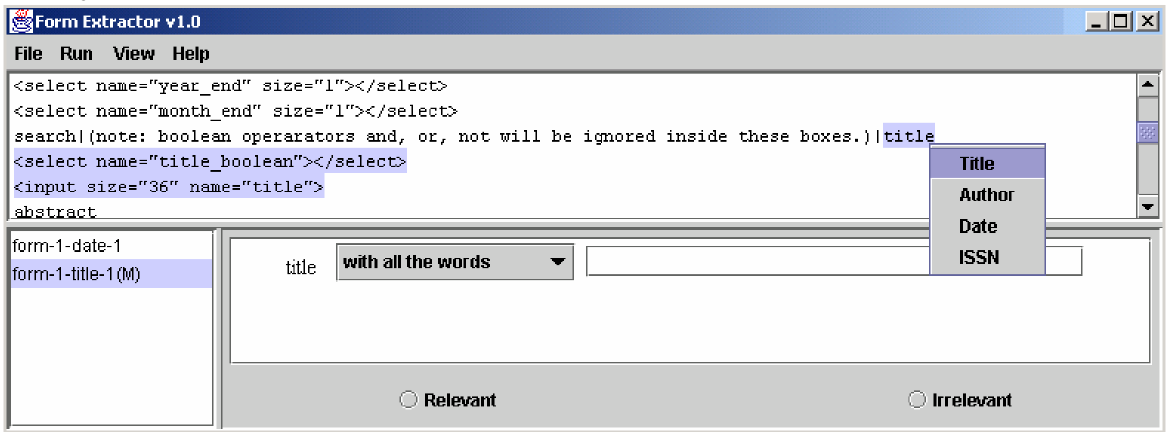


```
Newly generated Value-feature: <date datatype="number" length="4" pattern="****"/>
```

Figure 13 an example of the user's feedback

**2) Irrelevant:** When the user specifies a result is irrelevant to the objective, the IE system will do a negative feedback, which also includes five actions opposite to those in the above positive feedback.

**3) Marking:** If the system doesn't extract any *input field* for an *objective*, the user may be asked to mark the objective *input field* from the original page. The marked *input field* will be positively feedback to the system just in the same way as the case of "Relevant". Figure 14 gives an example of marking an *input field* for the "title" *objective.*



```
<select name="year_end" size="1"></select>
<select name="month_end" size="1"></select>
search|(note: boolean operarators and, or, not will be ignored inside these boxes.)|title
<select name="title_boolean"></select>
<input size="36" name="title">
abstract
```

Figure 14 an example of marking an *input field* for the "title" *objective.*

**4) No feedback:** When the user is tired of the feedback process, he/she may do nothing for the feedback. In this sense, the system has to feedback automatically by itself. In this case, the current demo system will choose the extracted result with biggest similarity to do positive feedback, which is the same as the situation of "Relevant". In the future, this should be refined.

## 6. Future work

1) Refine the Meta-dictionary and Value-feature by enriching their structure and semantic such as associating a *weight* with each item in these two knowledge bases.
2) Refine the process of the user feedback. Currently, for the purpose of simplifying the implementation, we just take the *positive* feedback into consideration.
3) Do experiments using the current *form identifying* system to find more missing considerations.