Using Similarity Measures for Context-Aware User Interfaces

Melanie Hartmann[1], Torsten Zesch[2], Max Mühlhäuser[1], Iryna Gurevych[2]
Technische Universität Darmstadt
[1]Telecooperation Group, [2]Ubiquitous Knowledge Processing Group
64289 Darmstadt, Germany
{melanie,zesch,max,gurevych}@tk.informatik.tu-darmstadt.de

Abstract

Context-aware user interfaces facilitate the user interaction by suggesting or prefilling data derived from the user's current context. This raises the problem of mapping context information to input elements in the user interface. We address this problem for web applications by (i) automatically extracting a textual representation of their input elements, and by (ii) mapping context information to them using these textual representations. In this paper, we present an approach for the representation extraction task that outperforms existing ones, and we explore the potential of similarity measures for the context mapping task.

1 Motivation

The increasing complexity of options available in today's applications mostly leads to a decreased usability of the user interface (UI). To counter this effect, we need UIs that support the user in performing her tasks by facilitating the interaction as much as possible in a proactive way. Thereby, the interaction –and thus the required support– strongly depends on the user's current context [13]. Context information ranges from physical information like the user's current location to more complex virtual objects like an entry in the user's calendar.

UIs that aim at facilitating the interaction between user and application by taking the user's current context into account are called context-aware UIs. A main feature of context-aware UIs is to provide context-based suggestions for required input to reduce interaction costs. An example of such a UI is shown in Figure 1. The importance of such suggestions, especially for mobile usage, is stressed by Rukzio [20]. He found that users are four times faster on a smart phone when they just have to correct prefilled form entries compared to entering the information from scratch. The multitude of open source and commercial applications available for automatically filling in form entries (e.g. iOpus Internet Macros[1], iNetForm Filler[2]) also shows the demand for this kind of interaction support.

The remainder of this paper is organized as follows: In Section 2, we introduce the context-aware UI system AUGUR that integrates the techniques presented in this paper, and we point out the two main challenges in using context information for facilitating the interaction, i.e. (i) finding a representation for the input elements and thus for the required input, and (ii) mapping context information to input elements. In Section 3, we give an overview of related work. Then, we describe our approaches for dealing with the representation (Section 4) and the mapping task (Section 5). In Section 6, we compare various similarity measures that can be applied for the mapping task. We conclude the paper in Section 7 with a summary and some ideas for future work.

2 Context-aware web application support

Our context-aware UI system called AUGUR [8] focuses on supporting the usage of web applications as most current applications are complemented or even replaced by a web version. Thereby, AUGUR provides support for any form-based web application, even for yet unknown ones. It is built as an overlay to existing web applications. This approach allows to provide support even across application boundaries. For example, when searching for a rental car, the same information (pick-up location, pick-up date, pick-up time, etc.) is often needed on various sites. AUGUR has an integrated context server that manages all kinds of context information, e.g. data gathered from the user's calendar or from a previously filled form. AUGUR uses the context information together with information about the user's interaction history to suggest or prefill data that is requested by the web application. In case that the system's suggestions are not entirely correct, the data that is finally entered by the user provides some additional information about the

[1]http://www.iopus.com/
[2]http://www.inetformfiller.com/

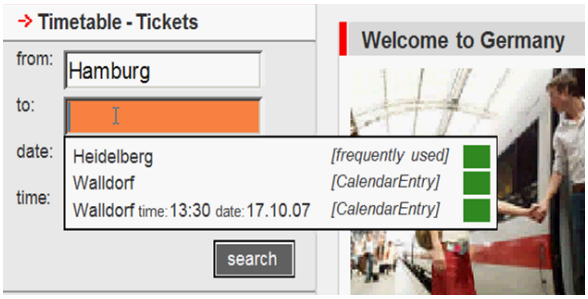

Figure 1. Example of a context-aware UI

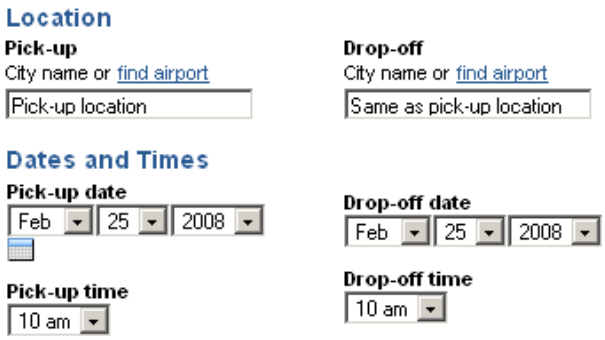

Figure 2. Example form

correct mapping. Thus, we store these mappings for future interactions, and use them to improve the quality of subsequently provided suggestions.

AUGUR's suggestions are not limited to one input element, but can combine suggestions for several input elements if they belong to the same data object. For example, AUGUR can consider the user's calendar to make suggestions, consisting of arrival station, date and time (see Figure 1). If the user chooses one of the combined suggestions, AUGUR fills all corresponding fields and highlights them to make the user aware of the system's actions.

Providing this context-aware support for arbitrary web applications raises the question: Which context object is relevant for the interaction with the application and how can it be mapped to the available input elements? Thus, we have to face two main challenges:

1. **Representing input elements:** How can we obtain a meaningful representation of the available input elements of the web application? Thereby, *representation* refers to a set of attributes that describe the input element, e.g. its label.

2. **Mapping context to input elements:** Which object in the user's current context matches the required input of a web application best and which part of this context object corresponds to which input element? We call this process "Context Mapping".

For example, consider the web application for renting a car in Figure 2: At first, we have to determine a representation for the relevant input elements, i.e. "Pick-up", "Pick-up date" etc. Next, we have to identify the most relevant object in the user's current context, e.g. a calendar entry object that contains information about a planned trip, and finally we have to map the values of the context object (e.g. the date, time and location) to the input elements.

The two main approaches for dealing with the representation and mapping tasks are (i) explicitly modeling all necessary information, or (ii) applying machine learning techniques. Modeling requires that the application developer or the user herself specifies a representation for every input element, or states the relationship to the available context information. However, this is not always feasible, considering the variety of different web applications and the enormous additional effort. In contrast, the machine learning approach attempts to learn these representations and mappings by analyzing multiple web forms. However, this requires initial training that is not possible if we want to support arbitrary domains. To counter these drawbacks, we introduce in this paper a heuristic-based approach for gathering a representation of the available input elements that is independent of the domain. Furthermore, we explore the potential of similarity measures for the task of mapping context representations to input elements. We analyze which matching techniques (e.g. substring matching [10, 15, 24] or concept vector based measures [6, 7]) and which domain-independent knowledge sources (e.g. WordNet [5] or Wikipedia) are best suited for this task. This approach is thereby not limited to web applications and can be applied for any other UI representation containing input elements.

3 Related work

As stated in the previous section, the two main steps of mapping context information to input elements are: (i) extracting a representation of the input elements, and (ii) mapping context information to these input elements. In this section, we review the state-of-the-art for these two tasks.

Representing input elements We focus on determining labels[3] for the input elements as we consider a label to be the most meaningful representation[4]. However, in contrast to other representations like the input element's name attribute, it cannot be easily extracted from the HTML representation.

Some approaches for extracting the label rely on a high-level description of the UI that is extracted from the HTML representation. Kaljuvee et al. [12] apply string matching

[3]Label denotes the text that accompanies an input element. For example, the label "Pick-up date" refers to the three input elements in Figure 2 where a date can be entered.

[4]This assumption is confirmed in Section 6.

to find the best match for the input element's name attribute from the text elements surrounding the input element. He et al. [9] define heuristics to determine the best label. However, these approaches have the drawback that they only consider a simplified textual representation of the website and not its actual visual layout. Raghavan and Garcia-Molina [18] address this problem by rendering a pruned version of the HTML representation with a custom layout engine for gaining the visual layout. However, this does not cope with the growing complexity and dynamic layout of today's web applications, e.g. by using Ajax. Zhang et al. [27] tackle this by using the HTML DOM API of a browser. They introduce a grammar (called 2P grammar) that describes the visual patterns in terms of directions (left, above, etc.). This grammar is used to gain a parse tree for the UI. Their best effort parser combines multiple possible parse trees to get the best representation of the web page. In Section 4, we compare their results with our presented approach.

Another system working on the actual visual representation is CoScripter [14], a popular Firefox plugin for recording intelligent macros, which also uses heuristics to assign a label to each input element. As CoScripter is publicly available, we also use it as a baseline system for our evaluation.

Mapping textual representations The task of mapping context information to input elements is strongly related to ontology mapping [11] and database schema matching [19], where concepts have to be mapped to ontology entries or column names, respectively. However, in these areas most approaches benefit from additonal information like constraints or instances that are more distinctive than the input element's label alone. Such information is usually not available for context-aware UIs. Bell and Sethi [2] rely only on the textual information for mapping records in a medical patient database. They apply synonyms, hypernyms/hyponyms, the equality of strings, common substrings, and soundex[5] similarity. However, their approach relies on user-crafted knowledge sources instead of generic ones.

Closely related to context mapping is also the research on the deep web[6] as it is concerned with mapping textual representations of several web forms. For example, Wu et al. [25] use cosine similarity for determining the similarity of label and name attributes. However, all approaches from the area of the deep web rely on a large corpus of web forms that is not available for context-aware UIs.

Other approaches for automatically filling in forms (e.g. [23]) either require apriori tagging of websites, or a manually crafted list which labels or names of input elements match which concepts. Thus, these approaches can only be applied to a specific domain (they focus on address information) or need explicit advice by the user.

Furthermore, in contrast to our approach none of the presented approaches deals with dynamic context information like calendar entries. They all rely either on predefined information or explicit user input.

4 Representing input elements

The first step of mapping context information to input elements is to obtain a representation of the available input elements. Some information can be directly extracted from the HTML representation. The most descriptive representation is the human readable **label** that is located somewhere around the input element (e.g. "Pick-up date" in Figure 2). However, the correct label mostly has to be inferred from the visual representation. Moreover, the label is often not sufficient for a meaningful representation, as more than one element can be associated with the same label (e.g. "Pick-up date" in Figure 2 is the label of three distinct input elements). For that purpose, we collect some further information for describing the input element (if available), i.e. the **name** attribute of the input element - that can give us its technical label, though this is often not human readable (e.g. "fcy" for the departure city)-, the corresponding **tooltip** ("alt" attribute), the data that is **prefilled** to give the user a hint (e.g. "Pick-up" in Figure 2 is prefilled with "Pick-up location"), and the **values** in dropdown menus, radio buttons, or grouped checkboxes. All this additional information can be directly gathered from the HTML representation of the page. The HTML syntax also defines a tag LABEL for marking a label that is associated to an input element; however, it is scarcely used in practice (only about 20% of the input elements we used for the evaluation in Section 6 had an associated label attribute). Thus, we focus on how we can determine the label for an input element.

As most web forms are similar in their layout, we assumed that we can easily define some common heuristics that are applicable to a wide range of web applications. Our approach called LabelFinder thereby focuses on the actual visual layout. In contrast to existing approaches, it also considers the exact visual position of texts to make it independent of the underlying HTML structure.

For identifying the best label for each input element, we at first determine all available input elements and all potential label candidates, i.e. all text elements on the website. Every input element is represented by its coordinates, its size, its type and its HTML LABEL attribute, if available. A label candidate is also described by its coordinates and its size. Further, its representation contains its textual content and its type as described in the following. The text elements are often embedded into larger divisions, thus the exact po-

[5] Soundex represents a string as a sequence of sounds.

[6] Deep web refers to all information in the web that cannot be accessed via conventional search engines following hyperlinks.

sition of the texts themselves cannot be directly determined. For coping with this problem, we temporarily insert a SPAN tag around them, and determine its position. The type of the corresponding label candidate is referred to as **inner label candidate**. However, as labels sometimes refer to various input elements and are thus not placed directly above all of them (see e.g. "Pick-up date" in Figure 2), we also keep the position information for the surrounding HTML tag that usually spans a greater section (**outer label candidate**).

From the analysis of various forms, we found that most labels are positioned on top or to the left of the corresponding input element. Checkboxes and radio buttons often do not have an explicit label. Especially ungrouped checkboxes are hardly ever explicitly labeled. For that reason, we also determine the labels of single checkboxes and radio buttons that are usually located on their top or to their right. Thus, grouped checkboxes and radio buttons have two possible labels: the **group label** or the **element label** in the beginning.

In the following, we list the heuristics which we apply to determine the best label candidate for an input element:

1. If the element has a corresponding LABEL element, we take it as label.

2. We ignore (i) all label candidates that are not located directly above or on the corresponding side (depending on its type) of the input element, i.e. all candidates that do not have a minimal predefined overlap in the corresponding dimension, (ii) all label candidates that have another label candidate between them and the input element, and (iii) *outer label candidates*, if an *inner label candidate* is available. For every remaining label candidate we compute the minimal Euclidean distance d between the input element and the label candidate. We take the square root of the distance for the horizontal dimension, as the horizontal distance grows much faster than the vertical distance (e.g. if another input element is arranged between the element and the label candidate). Finally, we take the label candidate with the smallest d.

3. For grouped checkboxes and radio buttons: If the *group label*'s distance d is smaller than n pixels (we empirically determined $n = 15$ to yield good results), we take it as the best label. Otherwise, we assign the label of the currently checked element to the group, or of the first element if no element is checked.

For evaluating the heuristics, we use the *IWRandom* dataset [1] provided by Zhang et al. [27]. The dataset contains 33 forms randomly sampled from the Web, mainly gathered from the website `invisible-web.net`. We assigned a label to each of the contained input elements. Two forms were dropped as they were difficult to annotate

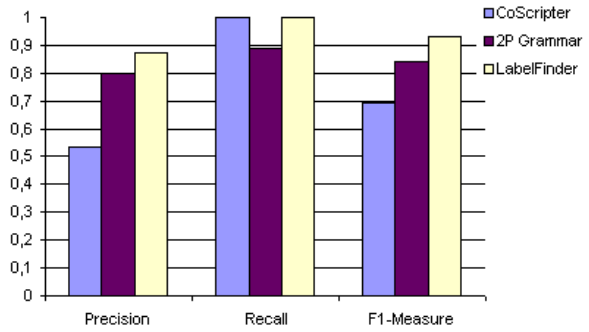

Figure 3. Overall performance of label recognition approaches

even for a human.[7] We compare our results with the results reported for the 2P Grammar by Zhang [27] and with the labeling component used in CoScripter [14]. Figure 3 shows the micro-average results in terms of precision, recall and the resulting F_1-measure. Our LabelFinder reaches a precision of .88 and perfect recall, resulting in an F_1-measure of .93. It thus clearly outperforms the label recognition of the 2P Grammar (.84) and of CoScripter (.69).[8] On the dataset that we used for the evaluation of the context mapping task in Section 6, LabelFinder yields a precision of .95 and perfect recall resulting in an F_1-measure of .97.

5 Mapping textual representations

Having determined a representation of the available input elements of a web application, we try to find relevant context objects and assign their content to the corresponding input elements. For example, if the user wants to rent a car, the interaction with the application can be supported by considering a context object that represents a trip. The travel object can be provided by the user's calendar, or derived from a previously filled web form. This context information can now be used to support the user in entering the travel information on another website. However, the components of the context objects and the input elements are not explicitly associated with a concept that expresses a semantic category like LOCATION. Concepts are represented by textual clues (e.g. "Pick-up location") with corresponding values (e.g. "New York Airport (JFK)"). For an input element on a website, the textual clues consist of values for the following attributes as described in Section 4: *label*, *name*, *tooltip*, *prefilled* data and *values*. The textual clues of a context object comprise a *name* and arbitrary other attributes (e.g. the subject of a calendar entry).

[7] Zhang et al. also used only 30 of the forms for their evaluation.
[8] The macro-average results are consistent with these findings.

The same concept may be represented by different textual clues (e.g. LOCATION as "Pick-up location" or "Rental station"). Thus, there is a need for a mapping process that bridges this vocabulary gap by finding a mapping between the concept representations from the context and from the web application. Relying only on the representations extracted as in Section 4 makes the AUGUR system independent of training data or apriori tagging of input elements.

We formalize the process of mapping concept representations as follows: A concept representation is a tuple (c, v), where c is a set of textual clues and v is the value of the concept. Thereby, all textual clues are separated into a set of tokens. A single textual clue is referred to as c^x where x denotes the attribute. For example, $(\{c^{label} = (\text{Pick-up, location})\}, (\text{New, York, Airport, (JFK)})$ is a possible representation of the concept LOCATION. Every object S in the context is represented by a set of source concepts $S = s_1, \ldots, s_n$, where each s_i is a (c, v) tuple. In the target web application T, each input field corresponds to an element t_i from the set of target concepts $T = t_1, \ldots, t_m$, where we only know the textual clues, but not their values.

Now, the task is to find the best matching source S from the context for a given web application T and then to assign the source concepts $s_i = (c_s, v_s)$ to the corresponding target concepts $t_j = (c_t, v_t)$. If we find a mapping between their textual clues c_s and c_t as described in Section 5.2, we can use the value v_s as a suggestion for the unknown value v_t. Each mapping is associated with a similarity score between 0 and 1, whereby 0 means no similarity, and 1 means perfect similarity. For determining the best matching source object, we compute the average similarity for all concepts of a source object and choose the one with the highest score.

Mapping concept representations relies on measuring the similarity between textual clues representing input elements and context objects. Thus, in the following, we present several similarity measures that can be used for that purpose.

5.1 Similarity measures

For determining the similarity between the values of textual clues c_s^x and c_t^y given some attributes x and y, we compute the maximum similarity among all pairs of tokens of the two strings:

$$sim(c_s^x, c_t^y) = \max_{\forall a \in c_s^x, b \in c_t^y} sim_{token}(a, b) \qquad (1)$$

For example, if the textual clues for the label attribute are "Pick-up location" and "Rental station", the overall similarity is the maximum similarity of the pairs (Pick-up, Rental), (Pick-up, station), (location, Rental), and (location, station).

We use two classes of similarity measures for sim_{token}: (i) string-based similarity measures, and (ii) semantic similarity measures.

String-based measures determine the similarity between two strings by comparing their characters. We use two baseline string measures: The exact string match measure (abbreviated as **exact**) returns 1 if the strings are exactly equal, and 0 otherwise. The bounded substring match measure (**b-substr**) returns 1 if the strings have a shared substring of at least 3 characters that is a prefix or a suffix of the other string (this matches strings like "arrival" and "arrive").

We also consider three more sophisticated measures that return a value in the interval $[0, 1]$: (i) the measure by Jaro [10] (abbreviated as **jaro**) that takes typical spelling deviations into account, (ii) an adaptation of the *jaro* measure by Winkler [24] (**jaro-w**) which increases similarity scores in the case of shared prefixes, and (iii) the measure by Monge and Elkan [15] (**monge-elkan**) that uses an affine gap model penalizing many small gaps in the string match more than a large gap.

Semantic measures As string-based measures are not likely to bridge the vocabulary gap between source strings and target strings, we also use semantic similarity measures relying on knowledge bases such as WordNet [5] (abbreviated as **wn**), Wikipedia[9] (**wp**), and Wiktionary[10] (**wkt**). A typical source of alternative wordings for the same concept is the use of synonyms (e.g. "city" and "town") or other terms closely related by a lexical semantic relation such as hypernymy/hyponymy or holonymy/meronymy (e.g. "city" and "New York"). Thus, we created a semantic similarity measure **relation** that returns 1, if the target string is a direct synonym, hypernym, hyponym, holonym or meronym of the source string, and 0 otherwise. The *relation* measure is very similar to the multitude of semantic similarity measures defined on WordNet (see [3] for an overview). However, these measures rely on special properties of WordNet, while the proposed measure can also be used with other knowledge bases like Wiktionary.[11]

Recent research on semantic similarity measures [6, 7] indicates that concept vector based measures are superior to these measures with respect to their performance and computational efficiency. For that reason, we also use a concept vector based measure **c-vector** [17], where the meaning of a string w is represented as a high dimensional concept vector $\vec{d}(w) = (d_1, \ldots, d_N)$. Each vector element d_i represents a document in the knowledge base, and the value of d_i is the string's tf.idf score [21] in the document. Semantic relatedness of two strings can then be computed as the cosine of their corresponding concept vectors.

When using semantic measures, we have to lemmatize inflectional forms of textual clues, as semantic knowledge

[9] http://www.wikipedia.org

[10] http://www.wiktionary.org

[11] We do not use the *relation* measure with Wikipedia, as it does not contain explicitly labeled lexical semantic relations.

bases usually contain only lemmas.

5.2 Determining the best mapping

For determining the best mapping between source and target concepts, we rank the available textual clues (e.g. *label*, *name*, *tooltip*, etc.) according to their descriptiveness (we consider "departure city" to be more descriptive than "fcy") and their occurrence probability (e.g. there is always a *name*, but the *tooltip* might be missing). We then take the top-ranked attributes m_1 and m_2 of the textual clues (i.e. *name* for context objects and an empirically determined attribute for input elements - see Section 6) and map all source concepts c_s and target concepts c_t that have a similarity value $sim(c_s^{m_1}, c_t^{m_2})$ above a predefined threshold θ. If two source concepts are mapped to the same target concept, the source concept with the higher similarity value is taken. However, this can also lead to tied cases if they have the same similarity values, e.g. if the two source representations "Pick-up date" and "Drop-off date" are both mapped to a target element "date". As we assume that each concept is only represented once in a context object or in a web application, we try to solve these tied mappings. For that purpose, we introduce a heuristic called **solveTies**. It relaxes Equation 1 by taking the average similarity instead of the maximum similarity among all pairs of tokens of the two strings. We then compute the similarity of all tied mappings and select the one with the highest score.

However, the initially chosen attributes may not suffice for finding an unambiguous mapping. Therefore, we define the heuristic **remap** that – step by step – takes more textual clues according to their ranking (less descriptive, lower occurrence probability) into account (e.g. *values* or *prefilled*) if no mapping is found. Thereby, already mapped elements are not taken into account.

6 Evaluation

For evaluating similarity measures for context mapping, we need a dataset containing possible context objects for a number of web applications and their mappings. As such data is hard to obtain, and we also want to be independent of how the context information is actually represented, we decided to use the representations used in the web forms as possible context representation. This means that we take the representation given by a source web form as a potential context object and try to map it to a target web form from the same domain. We repeat this process for every possible combination of web forms from the same domain (i.e. $n(n-1)$ context mapping cases, given n web forms).

Evaluation dataset We took 45 randomly picked web forms from 4 domains: cars (consisting of 7 web forms), flights (12), hotels (9), and address (17). Most web forms for the cars, flights, and hotels domains were taken from the TEL-8 dataset of the UIUC dataset [1]. We annotated each web form with the different concepts that exist in the corresponding domains. We automatically determined all textual clues including the label (using our LabelFinder as described above), but we manually corrected the label if necessary to avoid error propagation. However, as the LabelFinder reaches a precision of 95% for the data used, we assume that the influence of the incorrect labels would be minimal.

Experimental setup For our experiments, we implemented the *exact* and *b-substr* measures, and used the SecondString library [4] for the *jaro*, *jaro-w* and *monge-elkan* measures.

The semantic similarity measures rely on the following lexical semantic knowledge bases: (i) WordNet 3.0 together with the freely available JWNL WordNet API[12], (ii) the English Wikipedia dump from February 6th, 2007 together with the JWPL Wikipedia API [26], and (iii) the English Wiktionary dump from Oct 16th, 2007 with the JWKTL Wiktionary API [26].[13] For normalizing inflectional forms of textual clues, we used lemmatization as provided by the TreeTagger [22]. For stemming, we used the Porter Stemmer [16].

We decided to optimize our system for desktop settings. Thus, we aim at high precision as wrong suggestions are considered more disturbing in a desktop setting than e.g. in a mobile setting, where interaction costs are higher and even partially correct suggestions are normally considered beneficial. Thus, we use a rather conservative similarity threshold θ for the context mapping. We empirically determined the optimal value of the threshold on a dataset that is not used in the experiments. We used a threshold of 0.1 for the *c-vector* measures and a threshold of 0.85 for the *jaro*, *jaro-w*, and *monge-elkan* measures. All other measures return either 0 or 1, thus no threshold is needed.

System configuration For finding the best system configuration, we have to determine the influence on the system performance of (i) lemmatization or stemming, (ii) the best attribute m that is used for initial mapping, and (iii) the heuristics introduced in Section 5.2.

As was to be expected, **lemmatization** always improved the performance of the semantic similarity measures that rely on knowledge bases containing only lemmas. There was no significant influence of lemmatization or stemming on the performance of string-based similarity measures.

[12] `http://jwordnet.sourceforge.net/`

[13] JWPL is available from our website `http://www.ukp.tu-darmstadt.de/software/JWPL`. JWKTL will be publicly released in Summer 2008.

Domain	Heuristic		
	-	solveTies	remap + solveTies
Cars	.34	.49 (+15)	.61 (+27)
Flights	.48	.53 (+5)	.63 (+15)
Hotels	.70	.73 (+3)	.81 (+11)
Address	.64	.75 (+11)	.79 (+15)

Table 1. Average F_1 scores over all measures without using heuristics (-), using *solveTies* heuristic alone, and in combination with *remap*

We then tested which of the **attributes** described in Section 4 is best suited for the initial context mapping. We excluded the *tooltip*, *prefilled*, and *values* attributes, as they are not present for most input elements.[14] We compared the performance for the remaining *label* and *name* attributes and found that *label* clearly outperforms *name* across all measures and domains in terms of F_1 values.

We then assessed the influence of the **heuristics**. Table 1 gives an overview of the micro-average F_1-measure scores on the four domains (averaged over all measures). The *solveTies* heuristic leads to significant performance improvements for most measures in all domains, and never decreases performance. Additionally, applying the *remap* heuristic in combination with *solveTies* increases the F_1-measure in all cases over using *solveTies* alone. The amount of the average performance increase is thereby domain-dependent and ranges from .11 in the hotels domain, to .27 in the cars domain.

Thus, we found that the optimal system configuration for our evaluation is: using lemmatized textual clues from the *label* attribute for initial mapping and then applying the *remap* and *solveTies* heuristics in combination.

Results and discussion Table 2 shows the results of the best system configuration (as obtained in the previous section) in terms of precision, recall, and F_1-measure. There is no measure that performs best in all settings. However, the best F_1 scores in all domains are yielded by string-based rather than semantic measures. The results range from an F_1-measure of .71 in the *cars* domain to .97 in the *hotels* domain. The comparably low scores of the *relation-wkt* measure are due to the still low recall of the Wiktionary resource.

When looking at single mappings, we observe that semantic measures find a lot of mappings that the string-based measures cannot find, e.g. "destination" is correctly mapped to "drop-off airport or city" or "leave from" to "departure airport". However, semantic measures do not find mappings that are easily identified by string-based measures, e.g. mapping "e-mail" to "email" or "pick up" to "pick-up", as such spelling variants are usually not reflected in the semantic knowledge bases.

Furthermore, we used a conservative threshold θ that puts semantic measures at a disadvantage as they are known to have a lower precision than string-based measures, but are supposed to improve recall. The semantic measures also cannot show their full potential, as the high scores of the simple *exact* measure show that the vocabulary mismatch inside a single domain is relatively low. We expect the semantic measures to yield better results, when trying to map context objects across domain boundaries.

To sum it up, string-based measures reach a high performance in the context mapping task that is sufficient for applying it in a context-aware UI like AUGUR. Moreover, we assume that the performance can still be increased when combining them with semantic measures in a cascaded approach.

7 Conclusion

In this paper, we presented a new approach for mapping context information to web applications to facilitate the interaction with them. We address the two main challenges that arise from context mapping: (i) finding a representation for the input elements, and (ii) mapping the context information to them. For the first challenge, we introduced a simple heuristic based approach for determining the best label for an input element that clearly outperforms existing approaches. For the second challenge, we explored the potential of semantic similarity measures, and found that string-based measures outperform the semantic measures. The overall performance (.71 – .97 F_1-measure) allows our AUGUR system to correctly suggest data for most input elements without the need for any training or manual tagging of web applications.

In this pilot study, we tested each domain separately. Thus, we only had to deal with a small vocabulary mismatch, preventing the semantic measures from showing their full potential. In future work, we are going to evaluate the semantic similarity measures across various domains that are all related to the same context information (e.g. the cars, flights and hotels domain). Furthermore, we plan to evaluate a cascaded approach that combines the advantages of several measures, i.e. at first finding a mapping with string-based measures and then with semantic measures.

In our evaluation, we focused on high precision values and thus favored desktop applications. However, for mobile scenarios it is often of advantage to have a high recall as the interaction costs for changing or deleting an item are lower than for inserting it from scratch. For that purpose, we aim to adapt our mapping strategies to the devices used, i.e. whether high precision or high recall is needed.

[14] In the evaluation dataset, *tooltip* attributes can be found for 1% of all input elements, *prefilled* for 12%, and *values* for 44%.

	Domain	Cars			Flights			Hotels			Address		
		P	R	F_1	P	R	F_1	P	R	F_1	P	R	F_1
string-based	exact	.97	.56	**.71**	.97	.67	.79	.99	.83	.91	.99	.79	.88
	b-substr	1.00	.39	.57	.94	.67	.78	1.00	.95	**.97**	.98	.85	**.91**
	jaro	.78	.57	.65	.88	.73	.80	.97	.85	.91	.92	.82	.86
	jaro-w	.78	.57	.65	.87	.72	.79	.96	.85	.90	.92	.81	.86
	monge-elkan	.85	.56	.67	.93	.72	**.81**	.94	.83	.88	.93	.82	.87
semantic	relation-wn	.94	.53	.68	.78	.31	.44	1.00	.88	.93	.92	.67	.78
	relation-wkt	1.00	.10	.18	.33	.03	.05	.96	.10	.18	.78	.12	.21
	c-vector-wn	.86	.55	.67	.89	.47	.62	.94	.73	.82	.97	.81	.88
	c-vector-wkt	.86	.54	.66	.80	.47	.59	.94	.74	.83	.95	.80	.86
	c-vector-wp	.68	.56	.61	.72	.50	.59	.82	.74	.78	.78	.74	.76

Table 2. Micro-average precision, recall and F_1-measure using the best system configuration

Acknowledgements We would like to thank SAP Research Darmstadt for supporting our research in the AUGUR project. This work was also carried out as part of the project "Semantic Information Retrieval from Texts in the Example Domain Electronic Career Guidance" (SIR) funded by the German Research Foundation under the grant GU 798/1-2.

References

[1] The UIUC web integration repository. Computer Science Department, University of Illinois at Urbana-Champaign. http://metaquerier.cs.uiuc.edu/repository, 2003.

[2] G. B. Bell and A. Sethi. Matching records in a national medical patient index. *Commun. ACM*, 44(9):83–88, 2001.

[3] A. Budanitsky and G. Hirst. Evaluating WordNet-based Measures of Semantic Distance. *Computational Linguistics*, 32(1), 2006.

[4] W. Cohen, P. Ravikumar, and S. Fienberg. A comparison of string metrics for matching names and records. In *Proceedings of KDD*, 2003.

[5] C. Fellbaum. *WordNet An Electronic Lexical Database*. MIT Press, Cambridge, MA, 1998.

[6] E. Gabrilovich and S. Markovitch. Computing Semantic Relatedness using Wikipedia-based Explicit Semantic Analysis. In *Proceedings IJCAI*, pages 1606–1611, 2007.

[7] I. Gurevych, C. Müller, and T. Zesch. What to be? - electronic career guidance based on semantic relatedness. In *Proceedings of ACL*, pages 1032–1039, 2007.

[8] M. Hartmann, D. Schreiber, and M. Kaiser. Task Models for Proactive Web Applications. In *Proceedings of WEBIST*, pages 150–155, Mar. 2007.

[9] H. He, W. Meng, C. Yu, and Z. Wu. Automatic integration of web search interfaces with wise-integrator. *The VLDB Journal*, 13(3):256–273, 2004.

[10] M. A. Jaro. Probabilistic linkage of large public health data file. *Statistics in Medicine*, 14:491–498, 1995.

[11] Y. Kalfoglou and M. Schorlemmer. Ontology mapping: The state of the art. In *Semantic Interoperability and Integration*, number 04391 in Dagstuhl Seminar Proceedings, 2005.

[12] O. Kaljuvee, O. Buyukkokten, H. Garcia-Molina, and A. Paepcke. Efficient web form entry on pdas. In *Proceedings of WWW*, pages 663–672, 2001.

[13] H. Lieberman and T. Selker. Out of context: computer systems that adapt to, and learn from, context. *IBM Systems Journal*, 39(3-4):617–632, 2000.

[14] G. Little, T. A. Lau, A. Cypher, J. Lin, E. M. Haber, and E. Kandogan. Koala: capture, share, automate, personalize business processes on the web. In *Proceedings of CHI*, 2007.

[15] A. E. Monge and C. P. Elkan. The field matching problem: Algorithms and applications. In *Proceedings of KDD*, 1996.

[16] M. F. Porter. An algorithm for suffix stripping. pages 313–316, 1997.

[17] Y. Qiu and H. Frei. Concept Based Query Expansion. In *Proceedings of the ACM International Conference on Research and Development in Information Retrieval*, 1993.

[18] S. Raghavan and H. Garcia-Molina. Crawling the hidden web. In *Proceedings of VLDB*, pages 129–138, 2001.

[19] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal: Very Large Data Bases*, 10(4):334–350, 2001.

[20] E. Rukzio. Privacy-enhanced intelligent automatic form filling for context-aware services on mobile devices. In *Workshop on Artificial Intelligence at UbiComp*, 2004.

[21] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, 1983.

[22] H. Schmid. Probabilistic Part-of-Speech Tagging Using Decision Trees. In *International Conference on New Methods in Language Processing*, 1995.

[23] J. Stylos, B. A. Myers, and A. Faulring. Citrine: providing intelligent copy-and-paste. In *Proceedings of UIST*, pages 185–188, 2004.

[24] W. E. Winkler and Y. Thibaudeau. An Application of the Fellegi-Sunter Model of Record Linkage to the 1990 U.S. Decennial Census. Statistical Research Report Series RR91/09, Washington, D.C., 1991.

[25] W. Wu, C. Yu, A. Doan, and W. Meng. An interactive clustering-based approach to integrating source query interfaces on the deep web. In *Proceedings of SIGMOD*, pages 95–106, 2004.

[26] T. Zesch, C. Müller, and I. Gurevych. Extracting Lexical Semantic Knowledge from Wikipedia and Wiktionary. In *Proceedings of LREC*, 2008.

[27] Z. Zhang, B. He, and K. C.-C. Chang. Understanding web query interfaces: best-effort parsing with hidden syntax. In *Proceedings of SIGMOD*, pages 107–118, 2004.