Automatic Ontology Matching using Application Semantics

Avigdor Gal[*], Giovanni Modica[†], Hasan Jamil[‡], Ami Eyal[§]

Abstract

We propose the use of application semantics to enhance the process of semantic reconciliation. Application semantics involves those elements of business reasoning that affect the way concepts are presented to users: their layout, *etc.* In particular, we pursue in this paper the notion of *precedence*, in which temporal constraints determine the order in which concepts are presented to the user. Existing matching algorithms use either syntactic means (such as term matching and domain matching) or model semantic means, the use of structural information that is provided by the specific data model to enhance the matching process. The novelty of our approach lies in proposing a class of matching techniques that takes advantage of ontological structures and application semantics. As an example, the use of precedence to reflect business rules has not been applied elsewhere, to the best of our knowledge. We have tested the process for a variety of Web sites in domains such as car rentals and airline reservations, and share our experiences with precedence and its limitations.

1 Introduction and motivation

The ambiguous interpretation of concepts describing the meaning of data in data sources (*e.g.*, database schemata, XML DTDs, RDF schemata, and HTML form tags) is commonly known as *semantic heterogeneity*. Semantic heterogeneity, a well-known obstacle to data source integration, is resolved through a process of *semantic reconciliation*, which matches concepts from heterogeneous data sources. Traditionally, the complexity of semantic reconciliation required that it be performed by a human observer (a designer, a DBA, or a user) [15]. However, manual reconciliation (with or without computer-aided tools) tends to be slow and inefficient in dynamic environments, and for obvious reasons does not scale. Therefore, the introduction of the semantic Web vision and the shift towards machine-understandable Web resources has made clear the importance of automatic semantic reconciliation.

As an example, consider the Web search, an information-seeking process conducted through an interactive interface. This interface may be as simple as a single input field (as in the case of a

[*]Technion, Israel Institute of Technology, Technion City, Haifa 32000, Israel
[†]Mississippi State University, Mississippi State University MS 39762, USA
[‡]Mississippi State University, Mississippi State University MS 39762, USA
[§]Technion, Israel Institute of Technology, Technion City, Haifa 32000, Israel

general-purpose search engine). Web interfaces may also be highly elaborate: consider a car rental or airline reservation interface containing multiple Web pages, with numerous input fields, that are sometimes content dependent (*e.g.*, when a rented car is to be returned at the point of origin, no input field is required for the return location). A Web search typically involves scanning and comparing Web resources, either directly or via some information portal – a process hampered by their heterogeneity. Following the semantic Web vision, semantic reconciliation should be inherent in the design of smart software agents for information seeking. Such agents can fill Web forms and rewrite user queries by performing semantic reconciliation among different HTML forms.

To date, many algorithms have been proposed to support either semi-automatic or fully automatic matching of heterogeneous concepts in data sources. Existing matching algorithms make comparisons based on measures that are either syntactic in nature (such as term matching and domain matching) or based on model semantics. By model semantics, we mean the use of structural information that is provided by the specific data model to enhance the matching process. For example, XML provides a hierarchical structure that can be exploited in identifying links among concepts and thus allow a smooth Web search.

In this paper, we propose the use of application semantics to enhance the process of semantic reconciliation. Application semantics involves those elements of business reasoning that affect the way concepts are presented to users, such as layout. In particular, we pursue in this paper the notion of *precedence*, in which temporal constraints determine the order in which concepts are presented to the user.

All matching techniques aim at revealing latent semantics in data model descriptions and utilize it to enhance semantic reconciliation. To illustrate the differences among syntactic measures and data model semantics on the one hand, and application semantics on the other hand, consider a specific data model, XML, providing a domain description. Many matching techniques advocate the comparison of linguistic similarity, based on the assumption that within a single domain of discourse, terminology tends to be homogeneous. Linguistic similarity is based on terms that appear in the XML file. XML also has a hierarchical structure, allowing nesting of terms within other terms. This is a data model specific feature (that does not exist in a relational model, for example), and may drive another approach towards matching. The underlying assumption here is that hierarchy is a feature designers of all applications can use to model the domain of discourse better and thus can be used to identify similarities.

We aim at moving beyond the data model, and to do so one has to analyze the domain of discourse (or several similar domains) to identify basic business rules and how they impact data modeling. As an example, say the XML file describes a car rental application. Analyzing this domain (and other similar domains, such as airline reservation systems) reveals temporal constraints that control the reservation process. For example, pickup location always precedes drop off locations (both because renters typically drop off their rental at the same location and because the pickup location enforces constraints on the rest of the reservation, *e.g.*, the availability of car types). Equipped with this observation, one can interpret the ordering within the XML file as a representative of such temporal constraints. To summarize, application semantics analysis starts at the application (and not at the data model as in the other approaches) and then projected into the available data model to assist in the semantic reconciliation process.

The utilization of application semantics entails two immediate problems. First, it is likely that the data model does not support the application semantics features (or else they would have been used as data model semantics means). Therefore, there is the issue of formal representation of application semantics. Secondly, the lack of data model support means that algorithms that utilize application semantics are much harder to devise, having no underlying data model features upon which to be based.

To answer the first requirement of a rich data model for formal representation of application semantics, we choose to use ontologies. Ontologies are used as an interface conceptualization tool for representing model and application level semantics to improve the quality of the matching process. Four ontological constructs are used in this work, namely terms, values, composition, and precedence. Terms, values, and composition are borrowed from [5, 6]. Precedence, a unique feature of our model, represents the sequence in which terms are laid out within forms, imitating temporal constraints embedded in business rules.

In the general area of data integration, utilizing a full-fledged ontology, manually crafted to represent a domain of discourse with clear semantics, and detached from a specific application is a rare privilege. More often than not, semantics is hidden in the application code, and only hints to it are divulged through interfaces and database schemata. Since our ontologies correspond directly to the semantics of the application, we propose (untraditionaly) to abstract away ontologies from interfaces, thus exposing latent semantics. Therefore, composition can be extracted from the structure of a form, and precedence can be extracted from the ordering of elements in a form.

Given two ontologies (in the sense given above), algorithms to match terminologies in two Web resources are needed. We propose syntactical comparison, based on terms and values, enhanced by basic Information Retrieval techniques for string matching. We also discuss what is needed to generate an algorithm that utilizes application semantics and discuss the difficulties in crafting such an algorithm, relating to the second problem presented above.

The novelty of our approach lies in the introduction of a sophisticated matching technique that takes advantage of ontological constructs and application semantics. In particular, the use of precedence to reflect business rules has not been applied elsewhere, to the best of our knowledge. We have tested the process for a variety of Web sites in domains such as car rentals and airline reservations, and evaluated the performance of our algorithms. We highlight the benefits and limits of using the precedence construct as a guideline for future research into application semantics.

To support our research into application semantics, we developed OntoBuilder, a tool that extracts ontologies from Web applications and maps ontologies to answer user queries against data sources in the same domain. The input to the system is an HTML page representing the Web site main page. Using OntoBuilder, HTML pages are parsed using a library for HTML/XML documents, to identify form elements and their labels, and to generate an ontology. Ontologies are then matched to produce a mapping using the algorithms presented in Section 3. OntoBuilder supports an array of matching and filtering algorithms, and is extensible. OntoBuilder was developed using Java, and is available at `http://ie.technion.ac.il/OntoBuilder`.

1.1 Research background and related work

The study builds upon two existing bodies of research, namely heterogeneous databases and ontology design. Each is elaborated below.

1.1.1 Heterogeneous databases

The evolution of organizational computing, from "islands of automation" into enterprise-level systems, has created the need to homogenize databases with heterogeneous schemata (referred to as *heterogeneous databases*). More than ever before, companies are seeking integrated data that go well beyond a single organizational unit. In addition, a high percentage of organizational data is supplied by external resources (*e.g.*, the Web and extranets). Data integration is thus becoming increasingly important for decision support in enterprises. The growing importance of data integration also implies that databases with heterogeneous schemata face an ever-greater risk that their data integration process will not effectively manage semantic differences.

Current research into heterogeneous databases is largely geared towards manual (or semi-manual at best) semantic resolution (e.g., [16, 10]), which may not effectively scale in computational environments with dynamically changing schemata that require a rapid response. In addition, schema descriptions differ significantly among different domains. It is often said that the next great challenge in the semantic matching arena is the creation of a generalized set of automatic matching algorithms. Accordingly, the goal of this research is to propose the use of application semantics for automatic matching.

Over the past two decades, researchers in both academia and industry have advanced many ideas for reducing semantic mismatch problems, with the goal of lessening the need for manual intervention in the matching process. A useful classification of the various solutions proposed can be found in [23]. Of the categories presented there, we focus on those that deal with the algorithmic aspect of the problem.

The proposed solutions can be grouped into four main approaches. The first approach recommends adoption of Information Retrieval techniques. Such techniques apply approximate, distance-based matching techniques, thus overcoming the inadequacy of exact, "keyword-based," matching. This approach is based on the presumption that attribute names can be mapped using similarity techniques. Attribute names are rarely, however, given in explicit forms that yield good matchings. Furthermore, they need to be complemented by either a lengthier textual description or an explicit thesaurus, which mandates greater human intervention in the process. Protègè utilizes this method (among others) in the PROMPT algorithm, a semi-automatic matching algorithm that guides experts through ontology matching and alignment [9].

A second approach involves the adoption of machine learning algorithms that match attributes based on the similarity between their associated values. Most efforts in that direction (*e.g.*, GLUE [7] and Autoplex [3]) adopt some form of a Bayesian classifier. In these cases, mappings are based on classifications with the greatest posterior probability, given data samples. Machine learning was recognized as playing an important role in reasoning about mappings in [18].

Third, several researchers have suggested the use of graph theory techniques to identify similarities among schemata, where attributes are represented in the form of either a tree or a graph. To give but one example, The TreeMatch algorithm [19] utilizes XML DTD's tree structure in evaluating the similarity of leaf nodes by estimating the similarity of their ancestors.

In a fourth approach, matching techniques from the first three groups are combined. Here, a weighted sum of the output of algorithms in these three categories is used to determine the similarity of any two schema elements. Cupid [19] and OntoBuilder are two models that support this hybrid approach. OntoBuilder, however, is the only framework, to the best of our knowledge, in which application semantics is used as a tool in matching heterogeneous schemata.

1.1.2 Ontology design

The second body of literature we draw upon focuses on ontology design. An ontology is "a specification of a conceptualization" [14], where conceptualization is an abstract view of the world represented as a set of objects. The term has been used in different research areas, including philosophy (where it was coined), artificial intelligence, information sciences, knowledge representation, object modeling, and most recently, eCommerce applications. For our purposes, an ontology can be described as a set of terms (vocabulary) associated with certain semantics and relationships. Typically, ontologies are represented using a Description Logic [8], where subsumption typifies the semantic relationship between terms; or Frame Logic [17], where a deductive inference system provides access to semi-structured data.

The realm of information science has produced an extensive body of literature and practice in ontology construction (*e.g.*, [26]). Other undertakings, such as the DOGMA project [25], provide an engineering approach to ontology management. Finally, researchers in the field of knowledge representation have studied ontology interoperability, resulting in systems such as Chimaera [20] and Protègè [9].

The body of research aimed at matching schemata by using ontologies has focused on interactive methods requiring human intervention, massive at times. In this work, we propose a fully automatic process, which is a more scalable approach to semantic reconciliation. Our approach is based on analyzing model-dependent and application-level semantics to identify useful ontological constructs, followed by the design of algorithms to utilize these constructs in automatic schema matching. It is worth noting that automation carries with it a level of uncertainty as "the syntactic representation of schemas and data do not completely convey the semantics of different databases" [21]. In [11] we have formally modeled the uncertainty, inherent in an automatic semantic reconciliation, and offered an evaluation tool for the quality of algorithms that were designed for that purpose.

2 Ontological constructs

The methodology for the process of schema matching is based on ontological analysis of application classes and the generation of appropriate ontological constructs that may assist in the matching process. We base the ontological analysis on the work of Bunge [5, 6]. We adopt a conceptual modeling approach rather than a knowledge representation approach (in the AI sense). While the latter requires a complete reflection of the modeled reality for an unspecified intelligent task to be performed by a computerized system in the future [4], the former requires a minimal set of structures to perform a given task (a Web search in this case). Therefore, we build ontologies from a given application (such as Web forms) rather than with the assistance of a domain expert.

To exemplify the methodology, we focus on ontological constructs in the general task of the Web search. We recognize the limited capabilities of HTML (and for that matter, also XML) in representing rich ontological constructs, and therefore we have eliminated many important constructs (*e.g.*, the class structure), simply because they cannot be realistically extracted from the content of Web pages. Therefore, the ontological analysis of this class of applications yielded a subset of the ontological constructs provided by Bunge and added a new construct, which we term *precedence*, for posing temporal constraints.

Terms: We extract a set of terms[1] from a Web page, each of which is associated with one or more form entries. Each form entry has a label that appears on the form interface and internal entry names, that are not presented by the browser but still available in HTML. The label provides the user with a description of the entry content. The latter is utilized for matching parameters in the data transfer process and therefore resembles the naming conventions for database schemata, including the use of abbreviations and acronyms. A term is a combination of both the label and the name. For example, `Airport Location Code (PICKUP_LOCATION_CODE)` is a term in the Avis reservation page, where `Airport Location Code` is the label and `PICKUP_LOCATION_CODE` is the entry name.

Values: Based on Bunge [5], an attribute is a mapping of terms and value-sets into specific statements. Therefore, we can consider a combination of a term and its associated data entry (value) to be an attribute. In certain cases, the value-set that is associated with a term is constrained using drop lists, check boxes, and radio buttons. For example, the entry labeled `Pick-Up Date` is associated with two value-sets: {*Day, 1, 2, ..., 31*} and {*January, February, . . . , December*}. Clearly, the former is associated with the date of the month (and the value *Day* was added to ensure the user understands the meaning of this field) and the latter with the month (here, there is no need in adding a *Month* value, since the domain elements speak for themselves).

Composition: We differentiate atomic terms from composite terms. A composite term is composed of other terms (either atomic or composite). In the Avis reservation Web page, all of the terms mentioned above are grouped under `Rental Pick-Up & Return Information`. It is worth noting that some of these terms are, in themselves, composite terms. For example, `Pick-Up Time` is a group of three entries, one for the hour, another for the minutes, and the third for either *AM* or *PM*.

Precedence: The last construct we consider is the precedence relationship among terms. In any

[1]The choice of words to describe ontological constructs in Bunge's work had to be general enough to cover any application. We feel that the use of *thing*, which may be reasonable in a general framework, can be misleading in this context. Therefore, we have decided to replace it with the more concrete description of *term*.

interactive process, the order in which data are provided may be important. In particular, data given at an earlier stage may restrict the availability of options for a later entry. For example, the Avis Web site determines which car groups are available for a given session, using the information given regarding the pick-up location and time. Therefore, once those entries are filled in, the information is sent back to the server and the next form is brought up. Such precedence relationships can usually be identified by the activation of a script, such as (but not limited to) the one associated with a SUBMIT button. It is worth noting that the precedence construct rarely appears as part of basic ontology constructs. This can be attributed to the view of ontologies as static entities whose existence is independent of temporal constraints. We anticipate that contemporary applications, such as the one presented in this paper, will need to embed temporal reasoning in ontology construction.

The main difference between the first three constructs on the one hand, and the third category on the other, is that the equivalence of the construct in the data model is given explicitly in the former but is only implicit in the latter. In our example, terms are explicitly available as labels and entry names, and values are explicitly available as value-sets. Composition is explicitly available in XML definitions through its hierarchical structure.[2] The precedence construct, on the other hand, is only implicitly given, through the process of form submission.

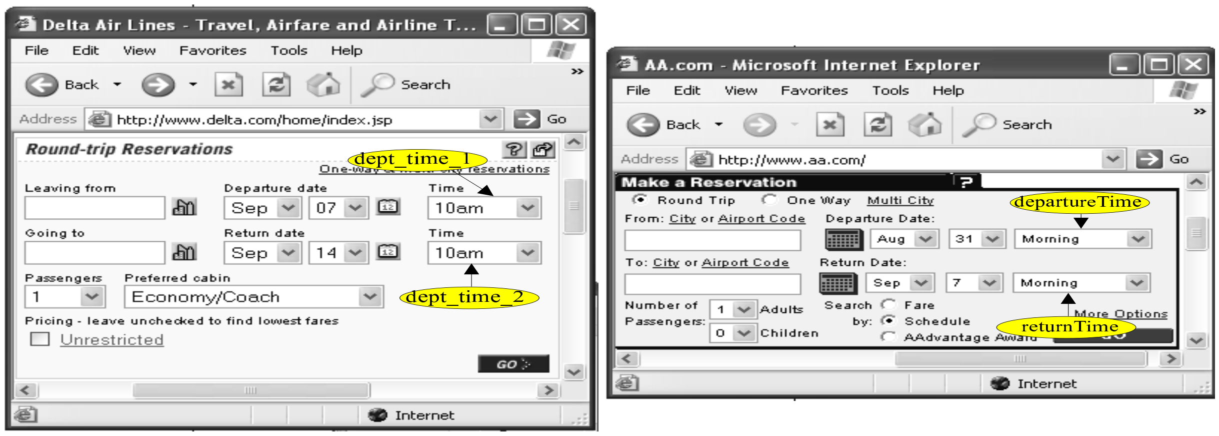
It is worth noting that the recognition of useful ontological constructs is independent of the algorithms that are utilized to perform the reconciliation process. In the ensuing discussion we shall demonstrate the usefulness of precedence in identifying correct mappings, yet discuss the difficulty of generating a good matching algorithm that avoids false positives and false negatives in the process.

3 Ontology matching

In the matching process, a mapping is determined between two ontologies. To illustrate the complexity of the process, consider first the following example.

Example 1 (Ontology matching) *Consider the Delta and American Airlines reservation systems (see Figure 1). Figure 1(a) presents a form that contains two time fields, one for departure and the other for return. Due to bad design (or designer's error), the departure time entry is named* `dept_time_1` *while return time is named* `dept_time_2`*. Both terms carry an identical label,* `Time`*, since the context can be easily determined (by a human observer of course) from the positioning of the time entry with respect to the date entry. For the American Airlines reservation system (Figure 1(b)), the two time fields of the latter were not labeled at all (counting on the proximity matching capabilities of an intelligent human being), and therefore were assigned, using composition by multiple term association, with the label* `Departure Date` *and* `Return Date`*. The fields were assigned the names* `departureTime` *and* `returnTime`*. Term matching would incur problems*

[2]Forms are given in HTML, which does not have a composition construct per se. Yet, our methodology transforms the HTML code into an XML definition, to be utilized in the reconciliation process.


Delta Air Lines - Travel, Airfare and Airline T...
File Edit View Favorites Tools Help
Back Search
Address http://www.delta.com/home/index.jsp Go
dept_time_1
dept_time_2
Internet
AA.com - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search
Address http://www.aa.com/ Go
departureTime
returnTime
Internet
(a)
(b)

Round-trip Reservations
One-way & multi-city reservations
Leaving from
Departure date Sep 07
Time 10am
Going to
Return date Sep 14
Time 10am
Passengers 1
Preferred cabin Economy/Coach
Pricing - leave unchecked to find lowest fares
Unrestricted
GO

Make a Reservation
Round Trip   One Way   Multi City
From: City or Airport Code
Departure Date: Aug 31 Morning
To: City or Airport Code
Return Date: Sep 7 Morning
Number of Passengers: 1 Adults 0 Children
Search by: Fare [✓ Schedule] AAdvantage Award
More Options

Figure 1: Delta versus AA

in differentiating the four terms (note that 'dept' and 'departure' do not match, neither as words nor as substrings). □

We denote by *Web resource dictionary* the set of terms extracted from a Web resource (typically composed of several Web pages within a single Web site). Let $V = \{v_1, v_2, ..., v_n\}$ and $U = \{u_1, u_2, ..., u_m\}$ be two Web resource dictionaries. The general matching process is conducted in two steps. First, pair-wise matching yields a similarity measure μ_{v_i,u_j} for all pairs $v_i \in \{v_1, v_2, ..., v_n\}$ and $u_j \in \{u_1, u_2, ..., u_m\}$. Next, a subset of the $n \times m$ pair-wise matching (dubbed a *mapping*) is selected as the "best" mapping between the two ontologies. Such a mapping may utilize some variation of a weighted bipartite graph matching [13], if the required mapping is of a $1:1$ nature. For a matching process that yields $1:n$ mappings, a simpler algorithm may be applied, in which a term v_i in one dictionary is mapped into a term u_j in another dictionary to which its similarity is maximized. Such an algorithm enables duplicate entries in one dictionary, yet does not allow the partition of a single value to several values.

The process of ontology matching is formalized and discussed in depth in [11]. In particular, we have shown in [11] that the specific methodology described herein is well suited to identifying the exact mapping (as perceived by a human observer) as the mapping with the highest sum (or average) of similarity measures of the selected term pairs.

This following sections focus on three methods for pair-wise matching, namely term, value, and precedence matching. We omit the discussion of the composition matching, for the sake of brevity. A detailed algorithm is available in [22].

4 Syntactic matching

4.1 Term matching

Term matching compares labels (verbal description of a form entry) and names (entry name as being sent to the server) to identify syntactically similar terms. To achieve better performance, terms are preprocessed using several techniques originating in IR research, including capitalization-based separation, ignorable character removal, de-hyphenation, and stop-term removal.

We have applied two separate methods for term matching based on string comparison as follows:

Word matching: Two terms are matched and the number of common words is identified. The similarity of two terms t_1 and t_2 using word matching (dubbed $\mu^W_{v_i,u_j}$) is defined as the ratio between the number of common words in t_1 and t_2 and the total number of unique words in terms t_1 and t_2, providing a symmetric measure of the similarity of these two terms. The more common words the terms share, the more similar they are considered to be. For example, consider the terms t_1=`Pickup Location` and t_2=`Pick-up location code`. The revised terms after preprocessing are $\bar{t}_1$=`pickup location` and $\bar{t}_2$=`pickup location code`. The term similarity, using word matching, is computed as

$$\mu^W_{t_1,t_2} = \frac{2\ (\texttt{pickup},\ \texttt{location})}{3\ (\texttt{pickup},\ \texttt{location},\ \texttt{code})} = 66\%$$

Two words $w_1 \in t_1$ and $w_2 \in t_2$ are considered to be common if they are spell the same, sound the same (soundex), or are considered synonyms, using a publicly available thesaurus such as WordNet.[3] Mismatched terms can be presented to the user for manual matching. Every manual match identified by the user is accepted as a synonym, and expands and enriches the thesaurus.

String matching: We find the maximum common substring between two terms whose words have been concatenated by removing white spaces. The similarity of two terms using string matching (dubbed $\mu^S_{v_i,u_j}$) is computed as the length of the maximum common substring as a percentage of the length of the longest of the two terms. As an example, consider the terms `airline information` and `flight airline info`, which after concatenating and removing white spaces become `airlineinformation` and `flightairlineinfo`, respectively. The maximum common substring is `airlineinfo`, and the effectiveness of the match is $\frac{length(\texttt{airlineinfo})}{length(\texttt{airlineinfomation})} = \frac{11}{18} = 61\%$.

We define a threshold (t^T) to identify a reasonable match. Any match with less than t^T is discarded. This threshold can be adjusted by the user.

For each pair, we compute four figures (two for labels, $\mu^{W,L}_{v_i,u_j}$ and $\mu^{S,L}_{v_i,u_j}$, and two for names, $\mu^{W,N}_{v_i,u_j}$ and $\mu^{S,N}_{v_i,u_j}$). We combine the figures into one figure, representing the strength of the match.

[3] http://www.cogsci.princeton.edu/~wn/

Therefore, the similarity measure of a term v_i with a term u_j is computed as the weighted average

$$\mu^T_{v_i,u_j} = \omega^{W,L}\mu^{W,L}_{v_i,u_j} + \omega^{S,L}\mu^{S,L}_{v_i,u_j} + \omega^{W,N}\mu^{W,N}_{v_i,u_j} + \omega^{S,N}\mu^{S,N}_{v_i,u_j} \tag{1}$$

where $\omega^{W,L}$, $\omega^{S,L}$, $\omega^{W,N}$, and $\omega^{S,N}$ are positive weights such that $\omega^{W,L} + \omega^{S,L} + \omega^{W,N} + \omega^{S,N} = 1$.

4.1.1 Experiments

We have conducted experiments to evaluate the performance of the term algorithm using two metrics, namely precision and relative precision. Let V and U be Web resource dictionaries. U partitions V into two subsets V_1 and V_2, such that V_1 is the set of all matchable terms and V_2 contains all those terms that cannot be matched with any term in U. Let M be a set of cardinality m, representing the set of all attributes in V that were matched by the algorithm. *Precision* (P) is the fraction of all found matches that are correct. It is computed as

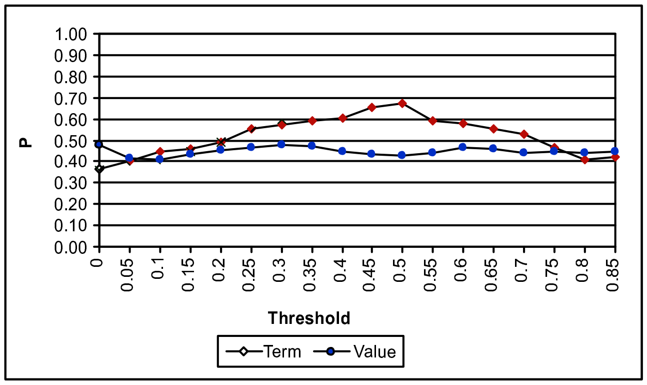
$$P = \frac{|V_1 \cap M|}{m}$$

Relative precision is concerned with the ability of an algorithm to avoid false positives. Let t be a threshold. $V_1(t)$ is the set of all matchable terms among those terms for which the algorithm has given a similarity measure higher then t. *Relative precision* (RP) is computed to be

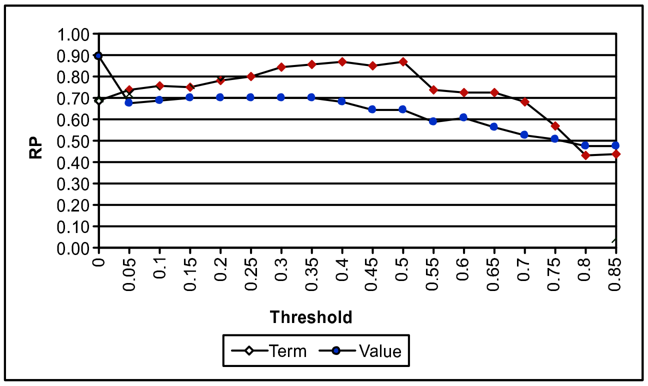
$$RP\,(t) = \frac{|V_1 \cap M\,(t)|}{|V_1(t)|}$$

The higher RP gets, the more efficient is the algorithm (at a given threshold) in avoiding false positives. It is worth noting that for $t = 0$, $V_1(t) = V_1$, and $RP\,(0)$ becomes recall, which is computed (using our terminology) as

$$\frac{|V_1 \cap M|}{|V_1|}$$



(a)



(b)

Figure 2: Precision and Relative Precision vs. Threshold

Figure 2 illustrates the performance of the term algorithm. Its performance varies from precision of 0.35 to 0.7. Its relative precision varies from 0.7 to 0.9. These results are good and can be attributed to the descriptive nature of labels in Web forms. However, even at its peak, the term algorithm identifies 0.1 of the matches incorrectly. Example 1 has illustrated one such case, that serves as a motivation to the presentation of the precedence construct. In Example 1, the Term algorithm prefers matching both `Time(dept_time_1)` and `Time(dept_time_2)` of Delta with `Return Date(returnTime)` of American Airlines.

4.2 Value matching

Value matching utilizes domain constraints to compute similarity measure among terms. Whenever constrained value-sets are present, we can enhance our knowledge of the domain, since such constraints become valuable when comparing two terms that do not exactly match through their labels. For example, the label corresponding to Avis' `Return Date` in Alamo's Web site is `Dropoff Date`. The labels only partially match, and the words `Return` and `Dropoff` do not appear to be synonymic in general-purpose thesauri (dropoff is not even considered a word in English, according to the Oxford English Dictionary [1]). Nevertheless, our matching algorithm matches these terms using their value-sets, since the term `Dropoff Date` has a value-set of {*(Select), 1, 2, ..., 31*} and the `Return Date` of Avis is associated with the value-set {*Day, 1, 2, ..., 31*}.

It is our belief that designers would prefer constraining field domains as much as possible, to minimize the effort of writing exception modules. Therefore, it is less likely (although known to happen occasionally) that a field with a dropdown list in one form will be designed as a text field in another form. In the case of a small-sized domain, alternative designs may exist (*e.g.*, *AM/PM* may be represented as either a dropdown list or radio buttons). Since the extraction algorithm represents domains in a unified abstract manner, the end result is independent of the specific form of presentation.

Fields with select, radio and check box options are processed using their value-sets. Therefore, different design methods act as no barrier in extracting the actual value sets. Value sets are preprocessed to result in generic domains. By recognizing separators in well-known data types, such as '/', '-', and '.' in date structures, ':' in time structures, '()' in telephone numbers, '@' in e-mail addresses, and 'http://' in URLs, domains can be partitioned into basic components, creating a compound term. The name of each new subterm is constructed as a concatenation of the existing name and the recognized domain type (*e.g.*, day). For example, the term `Pickup Date (pick_date)`, which is recognized as a *date* field based on its domain entries, is further decomposed into three subterms: `Pickup Date (pick_date_day)`, `Pickup Date (pick_date_month)`, and `Pickup Date (pick_date_year)`. It is worth noting that such preprocessing also affects term matching by generating additional terms, and therefore is performed prior to term matching.

Similarity is calculated as the ratio between the number of common values in the two value sets and the total number of unique values in them. For example, suppose that t_1=`Return time` and t_2=`Dropoff time` with values {*10:00am,10:30am,11:00am*} and {*10:00am,10:15am,10:30am,10:45am,11:00am*}, respectively. Preprocessing separates the domains into hour values ({*10,11*} versus {*10,11*}), minutes values ({*00,30*} versus {*00,15,30,45*}), and the value {*am*} (identical in both schemata). There is a perfect match in the hour domain, yet the minutes domains share two values (*00* and *30*) out of four (*00*, *15*, *30*, and *45*). Thus, the similarity is calculated as $\frac{2}{4} = 50\%$. The power of value matching can be further highlighted using the case of `Dropoff Date` in Alamo and `Return Date` in Avis. These two terms have associated value sets $\{$*(Select),1,2,...,31*$\}$ and $\{$*(Day),1,2,...,31*$\}$ respectively, and thus their content-based similarity is $\frac{31}{33} = 94\%$, which improves significantly over their term similarity $\left(\frac{\texttt{1(Date)}}{\texttt{3(Dropoff, Date, Return)}} = 33\%\right)$.

The domain recognition component can overcome differences of representation within the same domain. For example, we can apply transformations, such as converting a 24-hours representation into one of 12 hours. Thus, a domain $\{$*10:00, 11:00, 12:00, 13:00*$\}$ in a 24-hours representation can be transformed into three domains $\{$*1, 10, 11, 12*$\}$, $\{$*00*$\}$, and $\{$*am, pm*$\}$ in a 12-hours representation.

Figure 2 illustrates the performance of the value algorithm, as a function of the threshold. The reasonable performance of the Value algorithm is evident. What is not evident from this graph is that Value performance varies much more than that of other algorithms. Clearly, for ontologies with many different data types Value has good prediction capabilities (better than Term), while for onotologies in which many terms share the same domain, Value will find it much harder to predict correct mappings. The analysis of RP in Figure 2(b) shows a repetition of the patterns in Figure 2(a). An interesting phenomenon is the ability of the Value algorithm to outperform the term algorithm for a 0 threshold, with an average relative precision of 90%. This analysis can also serve in identifying optimal thresholds for various algorithms (in order to minimize false positives). Therefore, Value performs best at 0 threshold, while Term performs well in $[0.3, 0.5]$.

Returning to Example 1, it is worth noting that value matching cannot differentiate the four possible combinations, since they share the same time domain. Therefore, other alternatives, that exploit better the application semantics, should be considered.

5 Precedence matching

Let u_i and u_j be atomic terms in a Web resource dictionary. u_i *precedes* u_j if one of the following two conditions is satisfied:

1. u_i and u_j are associated with the same Web page and u_i physically precedes u_j in the page.
2. u_i and u_j are associated with two separate Web pages, U_i and U_j, respectively, and U_i is presented to the user before U_j.

Evaluating the first condition is easily achieved when the page is extracted into a DOM tree (short for Document Object Model), a W3C standard that can be used in a fairly straightforward manner to identify form elements, labels, and input elements. The properties of the precedence relation are summarized in the following proposition.

Proposition 1 *The precedence relation is irreflexive, antisymmetric, and transitive.*

The precedence relationship, as presented in this paper, serves as an estimation of the actual time constraints of a business process. For example, car rental companies would be likely to inquire about pick-up information before return information. As yet another example, consider the advance search Web pages of Lycos and Yahoo. The Term algorithm has had difficulties in matching `member name (m_u)` with `yahoo i_d (login)`, giving it a score of 0.01. Instead, it preferred matching `member name (m_u)` with `list my new yahoo mail address free (mail directory)`, with a much higher score of 0.2. Precedence, on the other hand, indicates that login information precedes other terms in this category of Web forms, putting it at the very beginning of the form.

Nevertheless, not all terms share precedence relationships. For example, there is no reason why either shipping address or invoice address should take precedence in a purchase order. To evaluate the difficulty of crafting a good matching algorithm, utilizing precedence, we have tested a simple algorithm, using a technique we term *graph pivoting*. Given an atomic term v_i in a Web resource dictionary V, we can compute the following two sets:

- $precede(v_i) = \{v_j \in V | v_j \text{ precedes } v_i\}$
- $succeed(v_i) = \{v_j \in V | v_i \text{ precedes } v_j\}$

It is worth noting that, following Proposition 1, $precede(v_i) \cap succeed(v_i) = \emptyset$. Given two terms, v and u, from two Web resource dictionaries V and U, respectively, we consider u and v to be pivots within their own ontologies. Therefore, we compute the similarity measure of matching $precede(v)$ with $precede(u)$, and $succeed(v)$ with $succeed(u)$. This computation is based on the syntactic similarity measures of term and value. Presumably, terms will tend to match better if both those that precede them and those that succeed them do so. Our experiments show that the performance of this algorithm measures significantly lower in precision than Term (only 30%-50%). The algorithm produces many false positive errors, suggesting that such an algorithm is put to better use in refuting possible matches than in supporting them.

6 Concluding remarks

In this paper, we have proposed the use of application semantics to enhance the process of ontology matching. Application semantics involves those elements of business reasoning that affect the way in which concepts are presented to users, for example via their layout. In particular, we have introduced the *precedence* ontological construct, in which temporal constraints determine the sequence of concepts presented to the user. While the paper has suggested the extraction of ontologies from HTML forms, we consider the use of ontologies to be essential for the broad area of Web search. Current search engines (in particular Google) have applied IR techniques in matching documents with user queries. We believe that the addition of structures such as precedence to search engines, whenever suitable, would enhance the precision of the search process. We leave this as an open research question. In particular, we will explore the use of additional ontology structures to improve the effectiveness of the matching process.

It is our conjecture that using application semantics as a means for semantic reconciliation can be generalized beyond its application to HTML Web forms. For example, the relational model has little ability to represent application semantic means such as precedence. However, many relational databases are interfaced nowadays through the use of HTML forms, for which precedence (and other application semantics) can increase the success of semantic reconciliation. Also, analysis of typical queries for a given application reveals information regarding the typical use of concepts, which can be further utilized in the semantic reconciliation process. We plan on investigating the methods illustrated above in future research.

While precedence has proven itself in certain instances, a good algorithm is still needed to extract this knowledge and put it to use, as our experiments show. The conceptual framework we provide, however, opens the door to more application-semantic concepts to be introduced and used in the ontology matching process.

We aim at continually improving the proposed algorithms. For example, the use of a linear algorithm for finding the maximal substrings and superstrings of two given strings was suggested in the context of bioinformatics [24]. Embedding a variation of this algorithm in our system may reduce the complexity of string matching. Finally, we intend to research in depth the problem of complex query rewriting in a heterogeneous schemata setting, using data types identification and domain normalization. The method proposed in this work serves as a promising starting point, yet a more thorough methodology is yet to be developed.

Research complementing the present paper provides sufficient conditions for matching algorithms to identify exact mappings, as conceived by an expert. This work is reported in [2, 11].

Acknowledgement

Our thanks is given to Louiqa Raschid and AnHai Doan for useful discussions. This research was partially supported by the Fund for the Promotion of Research at the Technion (191-496) and by the Fund of the Vice President for Research at the Technion (191-507). Giovanni Modica and Hasan Jamil's research was partially supported by National Science Foundation EPSCoR Grants (EPS-0082979 and EPS-0132618), a USDA-ARS Cooperative Agreement grant (CRIS-6406-21220-005-15S) and a Southwest Mississippi Resource Conservation & Development Grant (01050412). Avigdor Gal's research was partially supported by the IBM Faculty Award (2002). We thank Ido Peled, Haggai Roitman, and the class of "Information Systems and Knowledge Engineering Seminar," Fall Semester, 2002, for their assistance in collecting and analyzing the data.

References

[1] *Concise Oxford Dictionary.* Oxford Univ. Press, 8 edition, 1991.

[2] A. Anaby-Tavor, A. Gal, and A. Trombetta. Evaluating matching algorithms: the monotonicity principle. In S. Kambhampati and Craig A. Knoblock, editors, *Proceedings of the IJCAI-03 Workshop on Information Integration on the Web*, pages 47–52, Acapulco, Mexico, August 2003.

[3] J. Berlin and A. Motro. Autoplex: Automated discovery of content for virtual databases. In C. Batini, F. Giunchiglia, P. Giorgini, and M. Mecella, editors, *Cooperative Information Systems, 9th International Conference, CoopIS 2001, Trento, Italy, September 5-7, 2001, Proceedings*, volume 2172 of *Lecture Notes in Computer Science*, pages 108–122. Springer, 2001.

[4] A. Borgida. Knowledge representation, semantic data modelling: What's the difference? In *Proceedings of the 9th International Conference on Entity-Relationship Approach (ER'90)*, pages 1–2, Lausanne, Switzerland, 1990.

[5] M. Bunge. *Treatise on Basic Philosophy: Vol. 3: Ontology I: The Furniture of the World*. D. Reidel Publishing Co., Inc., New York, NY, 1977.

[6] M. Bunge. *Treatise on Basic Philosophy: Vol. 4: Ontology II: A World of Systems*. D. Reidel Publishing Co., Inc., New York, NY, 1979.

[7] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to map between ontologies on the semantic web. In *Proceedings of the eleventh international conference on World Wide Web*, pages 662–673. ACM Press, 2002.

[8] F.M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Reasoning in description logic. In G. Brewka, editor, *Principles on Knowledge Representation, Studies in Logic, Languages and Information*, pages 193–238. CSLI Publications, 1996.

[9] N. Fridman Noy and M.A. Musen. PROMPT: Algorithm and tool for automated ontology merging and alignment. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, pages 450–455, Austin, TX, 2000.

[10] A. Gal. Semantic interoperability in information services: Experiencing with CoopWARE. *SIGMOD Record*, 28(1):68–75, 1999.

[11] A. Gal, A. Anaby-Tavor, A. Trombetta, and D. Montesi. A framework for modeling and evaluating automatic semantic reconciliation. *VLDB Journal*, 2004. to appear.

[12] A. Gal, A. Trombetta, A. Anaby-Tavor, and D. Montesi. A model for schema integration in heterogeneous databases. In *Proceedings of the 7th International Database Engineering and Application Symposium*, Hong Kong, China, July 2003.

[13] Z. Galil. Efficient algorithms for finding maximum matching in graphs. *ACM Computing Surveys*, 18(1):23–38, March 1986.

[14] T.R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[15] R. Hull. Managing semantic heterogeneity in databases: A theoretical perspective. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 51–61. ACM Press, 1997.

[16] J. Kahng and D. McLeod. Dynamic classification ontologies for discovery in cooperative federated databases. In *Proceedings of the First IFCIS International Conference on Cooperative Information Systems (CoopIS'96)*, pages 26–35, Brussels, Belgium, June 1996.

[17] M. Kifer, G. Lausen, and J. Wu. Logical foundation of object-oriented and frame-based languages. *Journal of the ACM*, 42, 1995.

[18] J. Madhavan, P.A. Bernstein, P. Domingos, and A.Y. Halevy. Representing and reasoning about mappings between domain models. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI)*, pages 80–86, 2002.

[19] J. Madhavan, P.A. Bernstein, and E. Rahm. Generic schema matching with Cupid. In *Proceedings of the International conference on very Large Data Bases (VLDB)*, pages 49–58, Rome, Italy, September 2001.

[20] D.L. McGuinness, R. Fikes, J. Rice, and S. Wilder. An environment for merging and testing large ontologies. In *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR2000)*, 2000.

[21] R.J. Miller, L.M. Haas, and M.A. Hernández. Schema mapping as query discovery. In A. El Abbadi, M.L. Brodie, S. Chakravarthy, U. Dayal, N. Kamel, G. Schlageter, and K.-Y. Whang, editors, *Proceedings of the International conference on very Large Data Bases (VLDB)*, pages 77–88. Morgan Kaufmann, 2000.

[22] G. Modica. A framework for automatic ontology generation from autonomous web applications. Master's thesis, Mississippi State University, July 2002.

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

[24] Walter L. Ruzzo and Martin Tompa. A linear time algorithm for finding all maximal scoring subsequences. In Thomas Lengauer, Reinhard Schneider, Peer Bork, Douglas L. Brutlag, Janice I. Glasgow, Hans-Werner Mewes, and Ralf Zimmer, editors, *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, pages 234–241, Heidelberg, Germany, 1999. AAAI.

[25] P. Spyns, R. Meersman, and M. Jarrar. Data modelling versus ontology engineering. *ACM SIGMOD Record*, 31(4), 2002.

[26] B.C. Vickery. *Faceted classification schemes*. Graduate School of Library Service, Rutgers, the State University, New Brunswick, N.J., 1966.