OntoBuilder: Fully Automatic Extraction and Consolidation of Ontologies from Web Sources using Sequence Semantics

Haggai Roitman and Avigdor Gal

Technion – Israel Institute of Technology
Technion City, Haifa 32000, Israel

1 Introduction

Ontologies, formal specifications of domains, have evolved in recent years as a leading tool in representing and interpreting Web data. The inherent heterogeneity of Web resources, the vast amount of information on the Web, and its non-specific nature requires a semantically rich tool for extracting the essence of Web source content. The OntoBuilder project [5] supports the extraction of ontologies from Web interfaces, ranging from simple Search Engine forms to multiple-pages, complex reservation systems. Ontologies from similar domains are then matched to identify ontology mappings.

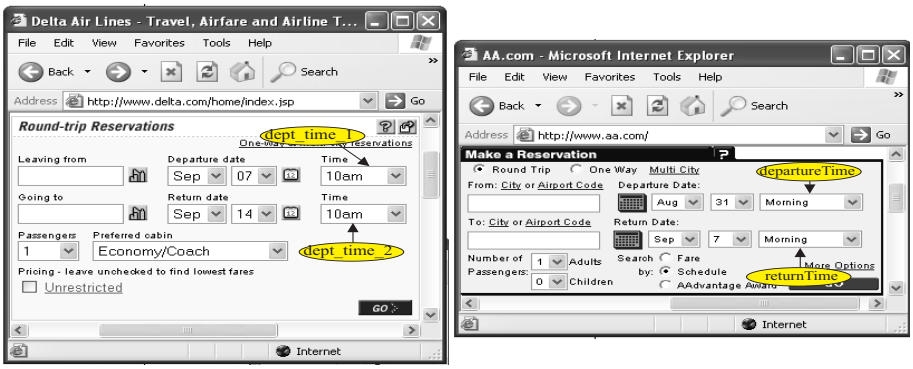
Given a sample form, filled by the user, and given a new form, from another Web site, OntoBuilder finds the best mapping between the two forms. This, in turn, can serve a system in automatically filling the fields, a sort of a query rewriting.

Unlike systems such as Protégé [2] OntoBuilder enables fully-automatic ontology matching, and therefore falls within the same category as GLUE [1]. The use of ontologies, as opposed to relational schema or XML, as an underlying data model allows a flexible representation of metadata, that can be tailored to many different types of applications. OntoBuilder contains several unique matching algorithms, that can match concepts (terms) by their data types, constraints on value assignment, and above all, the sequencing of concepts within forms (termed *precedence*), capturing sequence semantics that reflect business rules.

2 Overview of OntoBuilder

OntoBuilder was developed using Java, which makes it portable to various platforms and operating system environments. OntoBuilder generates dictionary of terms by extracting labels and field names from Web forms, and then it recognizes unique relationships among terms, and utilize them in its matching algorithms. There are two types of relationships OntoBuilder is specifically equipped to deal with, namely composition and precedence. The latter is discussed later in this section.

OntoBuilder is a generic tool and serves as a module for several projects at the Technion. For example, we have designed a framework for evaluating automatic schema matching algorithms [4], and we use OntoBuilder both for evaluation and for improving our methodology. This framework provides a sufficient condition (we term *monotonicity*) for a matching algorithm to generate “good” ontologies. Our empirical results



(a) (b)

Fig. 1. AA versus Delta

with OntoBuilder show that its algorithms satisfy one of the forms of monotonicity we present in [4]. Also, algorithms from OntoBuilder are being employed in an agent negotiation protocol for trading information goods [6]. Finally, OntoBuilder is used as a testbed for experimenting with simulataneous top-K mapping evaluation [3].

The rest of this section presents the main features and highlights of OntoBuilder, focusing on the sequence semantics. The detailed description can be found in [5, 7]. The process of ontology extraction and matching is divided into four phases. The input to the system is an HTML page representing a Web site main page. First, the HTML page is parsed and all form elements and their labels are identified. Next, the system produces an initial version of global (target) ontology and local (candidate) ontologies. Finally, the ontologies are matched to produce amapping.

Ontology matching aims at refining domain information by mapping various ontologies **within the same domain**. OntoBuilder supports an array of matching and filtering algorithms. Additional algorithms can be implemented and added to the tool as plug-ins. Algorithm parameters (such as weights) are specified using an XML configuration file which can be edited using a user-friendly interface.

Ontology matching is based on term and value matching, the former compares labels and field names using string matching, while the latter provides a measure of similarity among domains, as reflected by constrained data fields, such as drop-down lists and radio buttons. OntoBuilder provides several preprocessing techniques, based on Information Retrieval well-known algorithms such as *stoplists* and *dehyphenation*. It also supports automatic domain recognition and normalization to enhance the matching.

Once terms are extracted, OntoBuilder analyzes the relationships among them to identify ontological structures of composition and precedence. We focus here on the latter. **Precedence** determines the order of terms in the application according to their relative order within a page and among pages. In any 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, car rental forms will present pickup information before return information. Also, airline reser-

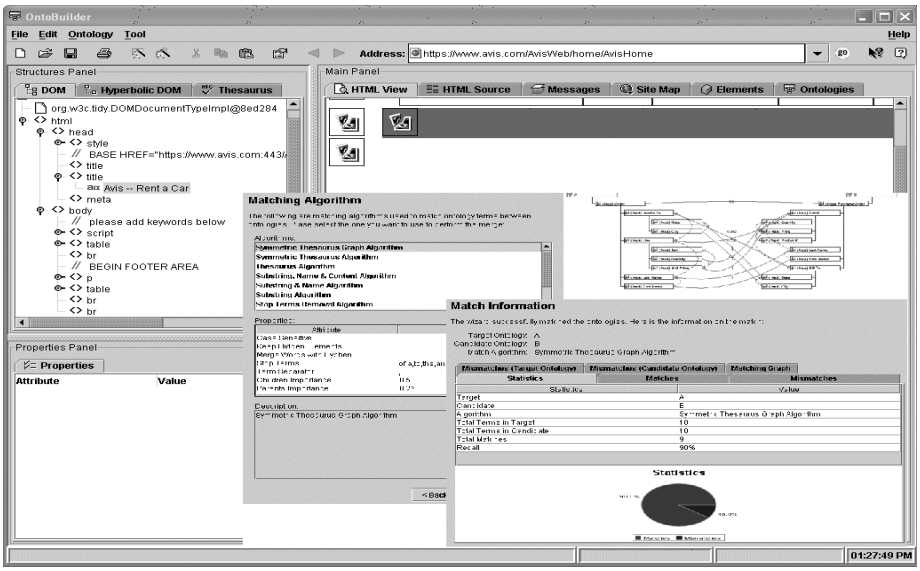


Fig. 2. The OntoBuilder user interface

vation systems will introduce departure information before return information. 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. It is our conjecture (supported by experiments) that precedence reflects time constraints of the application business rules and thus can be used to match better heterogeneous ontologies.

OntoBuilder employs unique algorithms for identifying structure similarity using **composition** and **precedence** constructs. Structure similarity is determined based on structure partitioning into subontologies, using terms as pivots, and comparison of subontologies. For example, using the precedence construct and two terms in two ontologies as pivots within their own ontology, OntoBuilder computes the similarity of subontologies that contain all terms that precede the pivots and also the subontologies that contain all terms that succeed the pivots (recall that Web forms enforce complete ordering of fields). A higher similarity among subontologies increases the similarity of the pivot terms themselves. This simple, yet powerful algorithm, has proven to be successful in a series of experiments performed with OntoBuilder on variety of Web sites. For example, consider Figure 1. The form of Delta airline reservation system 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 American Airlines reservation system (see Figure 1 on the

right), the two time fields of the latter were not labeled at all (relying on the proximity matching capabilities of an intelligent human observer), and therefore were assigned, using composition by association, with the label `Departure Date` and `Return Date`. The fields were assigned the names `departureTime` and `returnTime`. Term matching would prefer matching both `Time(dept_time_1)` and `Time(dept_time_2)` of Delta with `Return Date(returnTime)` of American Airlines (note that 'dept' and 'departure' do not match, neither as words nor as substrings). Value matching cannot differentiate the four possible combinations. Using precedence matching, OntoBuilder was able to correctly map the two time entries, since the subontologies of the predecessors of `Time(dept_time_2)` and `Return Date(returnTime)` match better than subontologies of other combinations.

OntoBuilder provides an easy to use environment for ontology authoring. Therefore, it can be used to build ontologies from scratch or refine extracted ontologies. In order to provide an intuitive interface to the user, the system implements common visualization techniques such as graph representations and hyperbolic views for ontologies, Web site maps, and document structures. Figure 2 provides a snapshot of OntoBuilder's user interface.

3 System demonstration

We will demonstrate OntoBuilder using an easy-to-follow example of matching Car rental ontologies. The system will create ontologies of car rental Web sites on-the-fly, and combine them into a global ontology. The benefits of OntoBuilder in resolving, in an automatic manner, semantic heterogeneity, including synonyms and designer errors, will be highlighted. In particular, we will focus on the use of the precedence construct in correctly identifying mappings.
OntoBuilder is available at `http://ie.technion.ac.il/OntoBuilder`.

References

1. 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.
2. 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.
3. A. Gal. Managing uncertainty in schema matching with top-k schema mappings. *Journal of Data Semantics*, (6):90–114, 2006.
4. A. Gal, A. Anaby-Tavor, A. Trombetta, and D. Montesi. A framework for modeling and evaluating automatic semantic reconciliation. *VLDB Journal*, 14(1):50–67, 2005.
5. A. Gal, G. Modica, H.M. Jamil, and A. Eyal. Automatic ontology matching using application semantics. *AI Magazine*, 26(1), 2005.
6. G. Koifman, O. Shehory, and A. Gal. Negotiation-based price discrimination for information goods. In *Proc. of the Third International Conference on Autonomous Agents & Multi Agent Systems*, NY, NY, 2004.
7. G. Modica. A framework for automatic ontology generation from autonomous web applications. Master's thesis, Mississippi State University, July 2002.