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

Avigdor Gal
Technion - Israel Institute of Technology

Giovanni Modica
Mississippi State University

Hasan Jamil
Wayne State University

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 sources' content. The OntoBuilder project [10, 5] supports the extraction of ontologies from Web search interfaces, ranging from simple Search Engine forms to multiple-pages, complex reservation systems. Ontologies from similar domains are then consolidated into an ever improving single ontology with which a domain can be queried, either automatically or semi-automatically.

As an example, consider Figure 1. The figure presents partial screen shots of two forms in the domain of matchmaking. The forms require similar information to be gathered by the system, yet may use different formats to gather the information. For example, while one form asks explicitly for the "Level of Education," another form may ask for it implicitly, using a label "You are a." The similarity of the two fields can be observed only when considering the possible values to be filled. To be able to automatically match heterogeneous forms, a system must be equipped with semantic understanding of the domain, available through such ontological constructs as composition.

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), according to the mapping suggested by OntoBuilder.

Unlike systems such as Protégé [4] and Lixto [2], OntoBuilder enables fully-automatic ontology matching, and therefore fall within the same category as Cupid [7] and GLUE [3]. 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 ordering of concepts within forms (termed *precedence*).

2 Overview of OntoBuilder

OntoBuilder was developed using Java, which makes it portable to various platforms and operating system environments. OntoBuilder also provides an applet version with the same features as the standalone version and the added functionality that allows users to access and use it within a Web client. The tool also runs under the Java Web Start technology. 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. The two types of relationships OntoBuilder is specifically equipped to deal with are composition and precedence, to be discussed in Section 2.2.

OntoBuilder is a generic tool and serves as a module for several projects, both at the Technion and at MSU. For example, we have designed a framework for evaluating automatic schema matching algorithms [1, 6], 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"

Figure 1: Heterogeneous forms example

ontologies. Our empirical results with OntoBuilder show that its algorithms satisfy one of the forms of monotonicity we present in [6]. OntoBuilder is also envisioned to serve as an information integration tool in the EthoSource [8] public data repository. Finally, algorithms from OntoBuilder are being employed in an agent negotiation protocol for trading information goods.

The rest of this section presents the main features and highlights of OntoBuilder. The detailed description can be found in [10, 5, 9]. The process of ontology extraction and matching is divided into four phases, as depicted in Figure 2. The input to the system is an HTML page representing a Web site main page (*e.g.*, `http://www.avis.com`). In phase 1, the HTML page is parsed using a library for HTML/XML documents. All form elements and their labels are identified in phase 2. In phase 3, the system produces an initial version of global (target) ontology and local (candidate) ontologies. Later, in phase 4, the ontologies are matched in an iterative manner to produce a refined global ontology. We next focus on the extraction and matching processes.

2.1 Ontology extraction

Ontology extraction begins with accessing each Web source by the system browser and parsing each page into an ordered tree, called DOM tree (short for Document Object Model), which identifies page elements. This W3C standard can be used in a fairly straightforward manner to identify form elements, labels, input elements, *etc.* OntoBuilder performs suitable "cleaning" and filtering, *e.g.*, elimination of superfluous tags and removal of formatting and scripting tags, to overcome incorrect specification of the source HTML code.

The diversity of layout techniques and principles in Web design complicates the label identification process for input elements even in a well-structured DOM tree. In order to overcome this diversity we have created a set of *extraction rules*, learned from a representative set of HTML documents in different domains, to recognize an HTML page layout. The extendable set depicts all table and non-table input layouts we have encountered. Examples of input layout include text and image labels for input elements

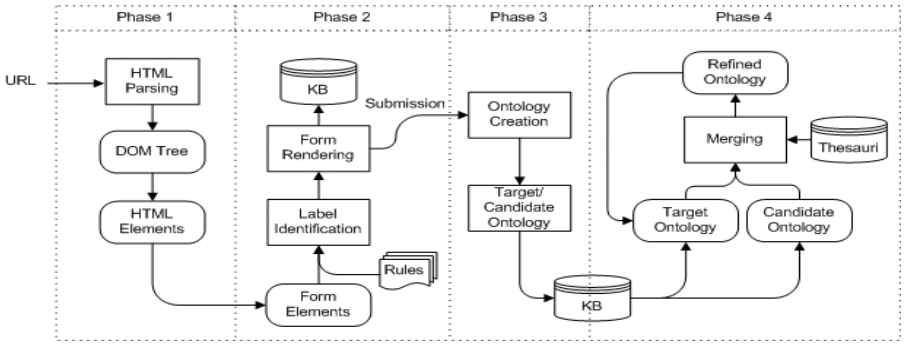

Figure 2: Ontology creation and matching process in OntoBuilder

(or forms), and table type and row type label and input field forms. The extraction process end result is an XML document containing the extracted terminology of the Web source (a dictionary).

The label identification algorithm employed in OntoBuilder is able to identify a high percentage of elements and their associated labels. Our experiments (reported in [9]) show that on average, the label identification algorithm achieves more than 90% effectiveness. These results do not include hidden fields, buttons, and images. Although these elements are used in the ontology extraction, they usually do not have an associated label (*e.g.*, hidden fields are not even shown to the user). With the release of the W3C standard for XHTML (basically well-formed HTML), Web application developers can have a solid foundation to make HTML pages easier to parse, assisting further in the task of ontology extraction. We plan on extending OntoBuilder capabilities to support XHTML as well.

Once terms are extracted, OntoBuilder analyzes the relationships among them to identify ontological structures of composition and precedence. **Composition** in Web forms is constructed through three techniques, namely *multiple term association*, *name similarity*, and *domain normalization*. *Multiple term association* involves the association of multiple terms with the same label, in which case all terms are named and grouped under that label. As an example, consider the American Airlines Web site presented in Figure 3, where the label `Departure Date:` relates to three different fields of month, day, and time. *Name similarity* groups entry labels that share identical prefix. *Domain normalization* involves the splitting of a term into subterms through recognition of known domains (such as day and time). Therefore, a time domain will be split into subterms, representing hours, minutes and AM/PM information. **Precedence** determines the order of terms in the application according to their relative order within a page and among pages. For example, car rental forms will present pickup information before return information. Also, airline reservation systems will introduce departure information before return information. 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. For a concrete example, see Section 2.2.

2.2 Ontology matching

Ontology matching aims at refining domain information by mapping various ontologies **within the same domain**. OntoBuilder supports an array of matching and filtering algorithms. There are four main algorithms that form the core of OntoBuilder matching process, namely *word similarity, string matching, value normalization*, and *value matching*. Additional algorithms can be implemented and added to the tool as plug-ins. All algorithms are extensions of an abstract algorithm interface. The interface describes the signature (methods and functions) that matching algorithms must implement in order to be used in the tool. 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

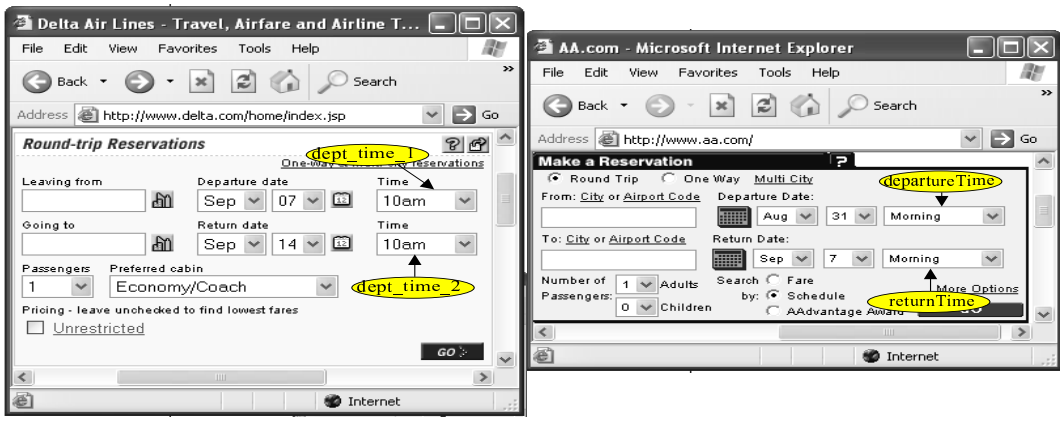

Figure 3: AA versus Delta

well-known algorithms such as *stoplists* and *dehyphenation*. It also supports automatic domain recognition and normalization to enhance the matching.

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 3. 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 3 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.

2.3 Additional features

OntoBuilder supports the use of wizards, easy-to-use scripts. The *Ontology Creation Wizard* assists the user in extracting ontologies from HTML pages. The *Ontology Merging Wizard* supports the matching and merging of ontologies.

OntoBuilder provides an easy to use environment for ontology authoring. Therefore, it can be used to build ontologies from scratch or refine extracted ontologies. It also provides conversion capabilities to a variety of ontology formats, including the BizTalk schema format from Microsoft. 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 4 provides a snapshot of OntoBuilder's user interface.

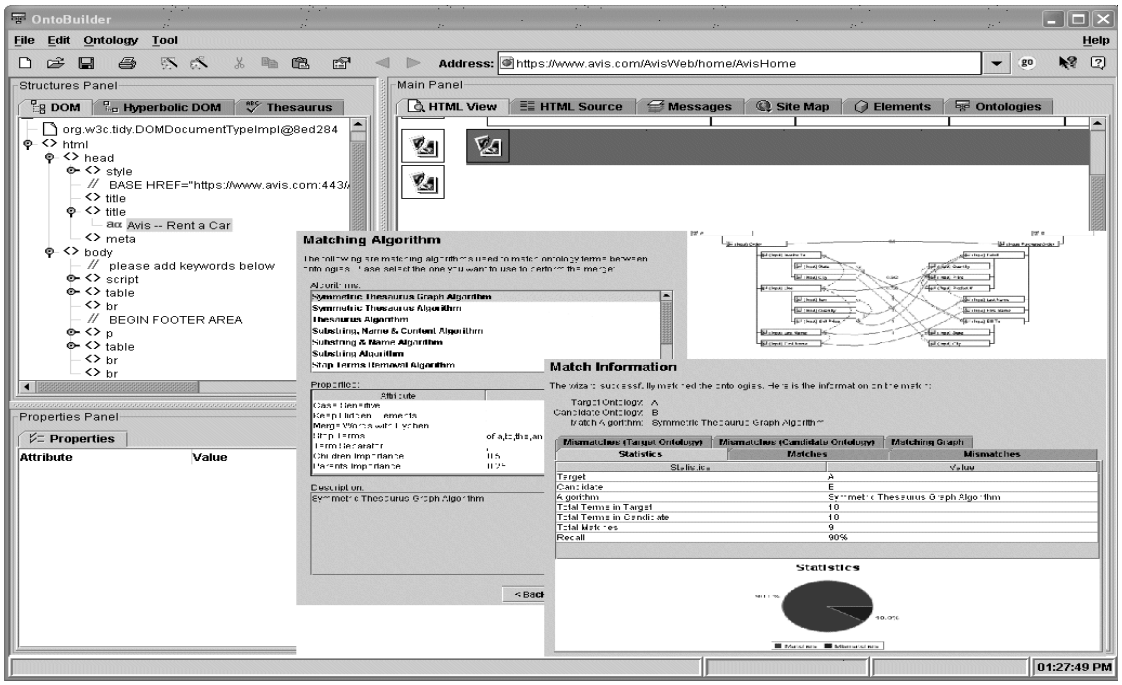

Figure 4: The OntoBuilder user interface

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. OntoBuilder is available at `http://www.cs.msstate.edu/~gmodica/Education/OntoBuilder`.

References

[1] 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.

[2] R. Baumgartner, S. Flesca, and G. Gottlob. Supervised wrapper generation with lixto. In *Proceedings of the International conference on very Large Data Bases (VLDB)*, pages 715–716, 2001.

[3] 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.

[4] 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.

[5] A. Gal, G. Modica, and H.M. Jamil. Improving web search with automatic ontology matching. Submitted for publication. Available upon request from avigal@ie.technion.ac.il, 2003.

[6] 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.

[7] 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.

[8] Emilia Martins and Hasan Jamil. Ethosource. Internet Address: http://sunflower.bio.indiana.edu/~bbleakle/.

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

[10] G. Modica, A. Gal, and H. Jamil. The use of machine-generated ontologies in dynamic information seeking. 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 433–448. Springer, 2001.