Wrapper Induction for End-User Semantic Content Development

Andrew Hogue
MIT CSAIL
Cambridge, MA 02139
ahogue@theory.lcs.mit.edu

David Karger
MIT CSAIL
Cambridge, MA 02139
karger@theory.lcs.mit.edu

ABSTRACT

The transition from existing World Wide Web content to the Semantic Web relies on the labeling and classification of existing information before it is useful to end-users and their agents. This paper presents a wrapper induction system designed to allow end-users to create, modify, and utilize semantic patterns on unlabeled World Wide Web documents. These patterns allow users to overlay documents with RDF classes and properties, and then to interact with this labeled content within a larger Semantic Web application, such as Haystack.

Categories and Subject Descriptors

H.3.5 [**Information Storage and Retrieval**]: Online Information Services; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces

Keywords

Semantic Web, Haystack, RDF, wrapper induction

1. INTRODUCTION

The Semantic Web promises to "bring structure to the meaningful content of Web pages, creating an environment where software agents roaming from page to page can readily carry out sophisticated tasks for users." [1] Information which is currently prepared only for humans to read will be richly labeled, classified, and indexed to allow intelligent agents to schedule our appointments, perform more accurate searches, and generally interact more effectively with the sea of data on the Web.

These advances, however, rely on the accurate semantic labeling of data that currently exists only in human-readable format on the World Wide Web. Normally, this labeling would be a task for content providers, as they have easy access to the relational data which makes up the pages, as well as the ability to alter the existing published content.

In this work our goal is to provide a tool which allows *end-users*, rather than content providers, to author and utilize their own semantic labels for existing content. By giving users control over semantic content, we hope to reduce the reliance of the Semantic Web on content providers and speed its adoption.

Our system allows users to create semantic patterns, also called *wrappers*, by simply highlighting relevant features of a web page in a browser. From these positive examples, a flexible, reusable pattern is induced. The user may then give the wrapper semantic meaning by overlaying it with RDF statements about the classes and properties it represents.

On subsequent visits to the same page (or to similar pages), the predefined pattern is reevaluated. When matches are found, instances of the semantic classes represented by the pattern are created on the fly. These objects are overlaid on the browser, allowing us to reify the matched items into first-class objects, rather than flat text.

By integrating this tool into the Haystack [10] information management environment, these semantic overlays create a rich environment in which the user may interact with content on the Web. For instance, a pattern defined on a page containing upcoming seminars would allow the user to right-click and add an interesting talk to their calendar. Patterns defined on news sites would allow the user to create, modify, and subscribe to their own RSS feeds.

We begin in Section 2 by surveying existing work on wrapper induction, as well as current user interfaces for labeling and manipulating semantic information on the Web. Section 3 describes the user interface which allows for pattern induction as well as interaction with existing patterns. In Section 4 we develop our underlying pattern induction algorithm, which utilizes the hierarchical nature of HTML to create accurate, reusable patterns. Finally, in Section 6, we conclude with a discussion of future directions and improvements.

2. RELATED WORK

Existing Semantic Web projects have tended to focus on enabling content-providers to produce semantic content, and end-users to consume it, whereas few have allowed end-users to create their own metadata.

The Annotea project [4] is a Web-based system which allows users to add descriptive metadata to individual pages. Im-

Figure 1: Creating a pattern on a talk announcement page.

plemented using current web standards, users of an Annotea-enabled browser may create annotations and associate them with text within a given document, or with the page as a whole. Although annotations are described using RDF, and could, in theory, be used to create semantically labeled content, each annotation applies to a specific node in a specific page, and there is no way to generalize them to be applied to repetitive content. Similarly, the concept of Sticky Notes for the Semantic Web [5] has been proposed, but does not allow the application of existing ontologies to Web data.

One interactive system which enables users to iteratively develop reusable patterns on the Web is *LAPIS* [8]. Patterns are developed using a language called *text constraints*, specifying operators such as *before*, *after*, *contains*, and *starts-with*. By using a pre-defined library of parsers which tokenize and label the document, users can create patterns of arbitrary complexity, or allow the system to infer them from examples. Currently, users may utilize LAPIS to perform such tasks as simultaneous editing and outlier finding.

Wrapper induction is defined by Kushmerick [7] as the task of learning a procedure for extracting tuples from a particular information source from examples provided by the user. Kushmerick defined the HLRT class of wrappers. These wrappers were restricted to locating information which is delimited by four types of flags, the "head," "left," "right," and "tail." Subsequent work on wrapper induction involves hierarchical structure [9], probabilistic models [2, 11], and hierarchical classifiers [12].

3. USER INTERFACE

Giving users the ability to effectively label and categorize semantic content on the Web requires an easy, intuitive user interface. Our wrapper induction algorithm has been implemented within the Haystack information management client, which gives users a rich set of contextual tools for creating and manipulating semantic data.

When a user wishes to wrap a page containing semantic content, they browse to the page, highlight the content, right-click, and choose "Create a Wrapper" from the context menu. They are then asked to choose a semantic class which describes the selected content. For example, Figure 1 shows a wrapper being created for a page containing data with the `Talk Announcement` semantic class.

Once the user has specified a class for the wrapper, it is generated using the algorithms described in the next section. The user is then provided with visual feedback by highlighting current matches within the browser. If the current wrapper is not general enough, the user may specify additional examples by simply highlighting the relevant text and choosing "Add to an Existing Wrapper" from the context menu.

When the user is satisfied with the current wrapper, they may add additional semantic properties to it. For example, `Talk Announcement`s have properties such as `title`, `speaker`, `date`, and `time`. The user simply highlights the data representing one of these properties and chooses "Add a Property" from the context menu. This action associates RDF predicates with certain "slots" in the pattern. For example, by selecting just the text "Effectively computing the landscape..." from the talk announcement above, the user could assign that part of the pattern the RDF predicate `title`.

Later, when a user visits a page with existing wrappers, they are matched against the content. When a match for a pattern is found, a new instance of the semantic class represented by the pattern is created and overlaid on the page. Because the Haystack environment provides content-specific context menus for semantic data, the user may now interact with semantic content on the web as if it were a first-class RDF object.

For example, in Figure 2, the user has right-clicked on one of the events on the talk announcement calendar. Because a `Talk Announcement` wrapper has been defined for this page, the user is presented with a context menu relevant to that class, including items such as "Add to My Calendar" and "Create Talk RSS Feed." Also note that certain properties of the `Talk Announcement` have been filled in, such as its `title`.

Figure 2: Utilizing a pattern on a talk announcement page.

4. WRAPPER INDUCTION

The user interface described in the previous section makes it easy for users to label semantic data on the Web, and later revisit that data and interact with it. The ability to perform these actions is enabled by a powerful wrapper induction algorithm which takes advantage of the hierarchical structure of the DOM of a web page.

Because of the way HTML is rendered in a browser, when a user selects a contiguous block of a web page, they are, in effect, selecting a subtree from the page's DOM[1]. Thus, by providing several highlighted examples in the browser, the user is indicating that several subtrees in the DOM have both similar syntactic structure and similar semantic meaning.

Our wrapper induction algorithm takes advantage of these structural similarities by finding the *best mapping* between the given example subtrees. The best mapping is defined as the mapping between the nodes of two trees with the lowest-cost *tree edit distance* [13].

Once we have found this best mapping, we create a pattern by removing any nodes that are unmapped. This creates a "skeleton" tree which contains only the common nodes among all examples. Figure 3 gives an example of the best mapping between two `Talk Announcement` subtrees, and the resulting pattern. We note that each example has the same basic structure, but different content. After the best mapping is found, unmapped nodes are removed and replaced with wildcards. If the user associates RDF predicates with the pattern, they are bound to these wildcard nodes. For instance, in Figure 3 the user has bound the wildcards in the generated pattern to predicates from the `talk` and `dc` schemas.

[1]In fact, the selection may represent several sibling subtrees, but similar edit-distance heuristics apply. See Section 5, as well as the first author's thesis [3], for more details.

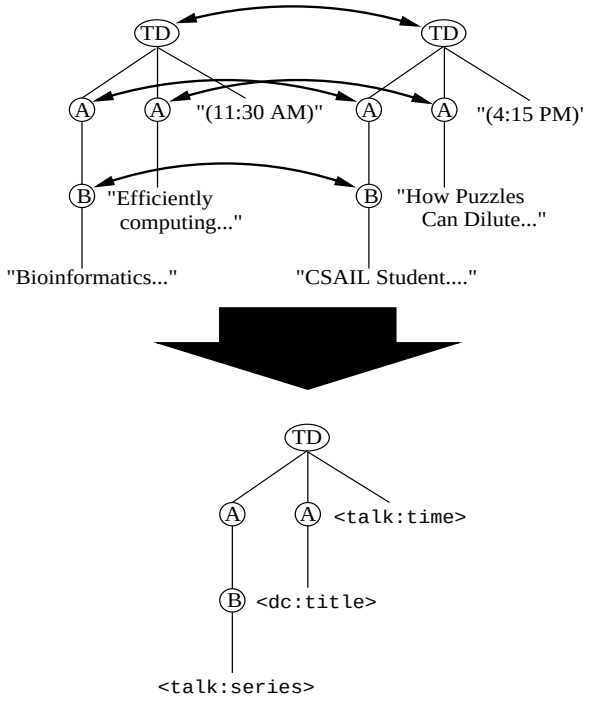

Figure 3: Generating a wrapper from the best mapping between two example trees.

Once the general pattern is created, we can match it against the document by simply looking for subtrees on the page which have the same structure. First, we find nodes in the document with the same tag as the root of the pattern. For each of those nodes, we attempt to match the children of the document node against the children of the pattern. This process continues recursively until every node in the pattern is bound to a node in the document, resulting in a match, or we fail to find a binding.

For each match that is found, a new instance of the RDF class associated with the wrapper is created. The wildcards in the pattern (which resulted from removing unmapped nodes) are bound to nodes in the document. If the user has labeled these slots with RDF predicates, the matched text is used to create the properties of the new instance. This instance may then be supplied to the browser or to an agent to be used as first-class semantic content.

In addition to the simple removal of unmapped nodes, additional heuristics have allowed us to reduce the number of examples usually needed to form an effective wrapper. One important heuristic is the *recursive collapse* of similar nodes in the pattern. When neighboring subtrees in the pattern have the same structure, we simply "collapse" them into a single subtree. Later, when we match the pattern, we allow the collapsed subtree to greedily match more than one node in the document.

By applying this heuristic recursively through the pattern tree, we can create patterns which can match variable-length lists of semantic items without forming a pattern for each possible list-length. An example is shown in Figure 4. Here,

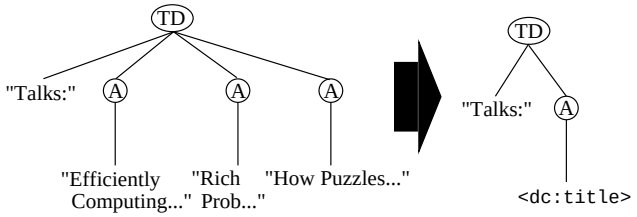

Figure 4: Collapsing nodes to form a "greedy" wrapper

the user has formed a wrapper to match a list of links to upcoming talks. By collapsing the `A` nodes into a single pattern node, we can match any number of talks listed on the page without forming a new wrapper.

5. EXPERIMENTAL RESULTS

While development is still ongoing, initial experiments in creating and using these wrappers have shown them to be widely applicable. Table 1 shows the number of examples necessary to form a wrapper for several common web sites.

The most common failure mode for our wrapper induction algorithm deals with semantic classes which span several neighboring subtrees. If these subtrees do not include all children of the parent node, it becomes difficult to effectivly calculate edit distance and perform matching. This causes failures, for example, with wrapping the `Story` class on the New York Times (http://nytimes.com). We are investigating the use of supplementary tools, such as LAPIS [8] to alleviate this issue.

6. FUTURE WORK

This interface and algorithm are based on an ongoing Master's thesis [3], and as such they are both very much works in progress. In addition to the ideas presented here, several other future improvements have suggested themselves:

Document-level Classes Many times an entire web page represents a single semantic class, with items on the page detailing the properties of that class. For example, each page from the Internet Movie Database (http://imdb.com) represents the `Movie` class, with items on the page representing properties of that class. We would like to allow for applying RDF statements which tie predicates on a page to the page-level class.

Labeling Across Pages Much semantic information transcends page boundaries. For instance, on the CSAIL events calendar, only the talk's `series`, `title` and `time` are listed on the calendar page, while more information is available by clicking on the title link. We would like to develop a system which allows semantic classes and properties to span multiple pages.

Wrapper Verification Web pages are constantly in flux, and methods for validating wrappers are important [6]. We would like to develop an efficient way to verify that the semantic content being returned by the wrappers is still accurate.

URL	Semantic Class	Examples
http://google.com/search	`SearchResult`	3
http://imdb.com/title	`Actor`	1
http://imdb.com/title	`Director`	2
http://slashdot.org	`Story`	2
http://www.csail.mit.edu/biographies/PI/biolist.php	`Person`	2
http://www.csail.mit.edu/events/eventcalendar/calendar.php	`Talk Announcement`	1

Table 1: Number of examples necessary to form a wrapper.

Wrapper Sharing The nature of the wrappers defined here allows them to be easily stored and retrieved. Once semantic patterns have been created for a page, they may be shared between users. One can imagine downloading a full set of wrappers for a given site and instantly enabling a full Semantic Web experience for users without the need for each user to author their own wrappers.

7. CONCLUSIONS

The ability to effectively and meaningfully label information on the World Wide Web is vital to the advancement of the Semantic Web. Normally, this responsibility lies with content-providers, but this paper has outlined another approach. By allowing end-users to participate create semantic wrappers, we can give anyone with a browser the ability to develop and interact with semantic content.

The key features of the wrappers presented here are that they are flexible, reusable, and serializable. Once defined, they transform a standard web page into a rich Semantic Web environment. Through integration with a rich environment such as Haystack, we have created a Semantic Web environment where the *user* is in control.

8. REFERENCES

[1] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284(5):35, May 2001.

[2] D. Freitag and A. McCallum. Information extraction with HMM structures learned by stochastic optimization. In *AAAI/IAAI*, pages 584–589, 2000.

[3] A. Hogue. Tree pattern inference and matching for wrapper induction on the World Wide Web. Master's thesis, Massachusetts Institute of Technology, forthcoming.

[4] J. Kahan and M.-R. Koivunen. Annotea: an open RDF infrastructure for shared web annotations. In *World Wide Web*, pages 623–632, 2001.

[5] D. Karger, B. Katz, J. Lin, and D. Quan. Sticky notes for the semantic web. In *Proceedings of the 8th International Conference on Intelligent User Interfaces*, pages 254–256, 2003.

[6] N. Kushmerick. Wrapper verification. *World Wide Web*, 3(2):79–94, 2000.

[7] N. Kushmerick, D. S. Weld, and R. B. Doorenbos. Wrapper induction for information extraction. In *Intl. Joint Conference on Artificial Intelligence (IJCAI)*, pages 729–737, 1997.

[8] R. C. Miller and B. A. Meyers. Lightweight structured text processing. In *Proc. of USENIX 1999 Annual Technical Conference*, pages 131–144, Monterey, CA, USA, June 1999.

[9] I. Muslea, S. Minton, and C. Knoblock. A hierarchical approach to wrapper induction. In O. Etzioni, J. P. Müller, and J. M. Bradshaw, editors, *Proc. of the Third International Conference on Autonomous Agents*, pages 190–197, Seattle, WA, USA, 1999. ACM Press.

[10] D. Quan, D. Huynh, and D. R. Karger. Haystack: A platform for authoring end user semantic web applications. In *Proc. 2nd International Semantic Web Conference*, 2003.

[11] K. Seymore, A. McCallum, and R. Rosenfeld. Learning hidden Markov model structure for information extraction. In *AAAI 99 Workshop on Machine Learning for Information Extraction*, 1999.

[12] L. Shih and D. Karger. Learning classes correlated to a hierarchy. Technical report, MIT AI Lab, 2001.

[13] K.-C. Tai. The tree–to–tree correction problem. *J. Association of Computing Machinery*, 26(3):422–433, July 1979.