SEMOLE:
A ROBUST FRAMEWORK FOR GATHERING INFORMATION FROM THE WORLD WIDE WEB[1]

Hyung-Jin Kim and Lee Hetherington

Spoken Language Systems Group
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139 USA
http://www.sls.lcs.mit.edu
{indigo, ilh}@sls.lcs.mit.edu

ABSTRACT

This paper describes seMole (*se*mantic *Mole*), a robust framework for harvesting information from the World Wide Web. Unlike commercially available harvesting programs that use absolute addressing, seMole uses a semantic addressing scheme to gather information from HTML pages. Instead of relying on the HTML structure to locate data, semantic addressing relies on the relative position of key/value pairs to locate data. This scheme abstracts away from the underlying HTML structure of Web pages, allowing information gathering to only depend on the content of pages, which in large part does not change over time. We use this framework to gather information from various data sources including Boston Sidewalk and the CNN Weather Site. Through these experiments we find that seMole is more robust to changes in the Web sites and it is simpler to use and maintain than systems that use absolute addressing.

1. INTRODUCTION

In 1994, the MIT Spoken Language Systems Group introduced GALAXY, a client-server architecture for accessing on-line information using spoken dialogue [1]. Since then, GALAXY has served as the testbed for our research and development of human language technologies, resulting in systems with expertise in various *domains* (A domain is one type of information such as airline information or weather information). In 1996, we made our first significant architectural redesign and introduced WEBGALAXY, which had a Java GUI to permit universal access via any web browser [2]. While the underlying human language technologies were the same across all domains, the information access/retrieval mechanisms for WEBGALAXY domains were developed separately and therefore lacked several key features:

- *Code re-use*: Because each domain was written with a different computer language, the work used to develop one domain could not be re-used to develop a new domain.
- *Robustness*: All information was gathered using an absolute addressing scheme similar to the addressing scheme of the WIDL language developed by WebMethods [3]. Data was gathered from each Web page by using the absolute HTML position of the data on the page. Therefore, if the page changed even slightly, the data retrieval would likely fail.

[1]This research was supported by DARPA under contract N66001-96-C-8526, monitored through Naval Command, Control and Ocean Surveillance Center and a research contract from Bell Atlantic.

SeMole (*se*mantic *Mole*) is an information retrieval framework that addresses these problems. Since seMole is a server-based framework in which all code resides on the server, it allows code to be re-used by anyone using the seMole framework. For example, a ski information harvester that needs information about the current weather can also invoke a weather harvester that resides on the server.

With seMole, users define *semantic templates*, XML-based scripts that use *semantic addressing* to gather information from a Web page. Many of today's commercial Web harvesting packages (i.e., WebMethod's B2B [3] and AgentSoft's LiveAgent Pro [4]) use an *absolute addressing* scheme to gather information from Web pages. An example absolute address for a hypothetical Web page is "The third text element matching the regular expression 'high forecast' under the second table element in the HTML structure." These addresses are very susceptible to small changes in the HTML structure. If, for example, another table were inserted at the beginning of our hypothetical Web page, then the absolute address would be pointing to the wrong data. For absolute addressing schemes to be reliable, the position of all pieces of data in the HTML structure must be consistent from one version of the page to the next. Unfortunately, since Web pages are usually updated often (to update the appearance or to change data on the page), these absolute addressing schemes tend to fail on a regular basis.

Semantic addressing, on the other hand, relies on the content of the page to find data rather than the HTML structure by itself. For example, to find information about the Wednesday high temperature forecast on a weather page, a semantic address might be "the integer nearest the word 'high' nearest the word 'Wednesday'." This scheme is entirely independent of the HTML structure and only depends on how key/value pairs are placed in the two-dimensional HTML space. Further, notice that no matter how much the Web page changes, the content of the page will remain the same: there will always be a Wednesday high forecast. Therefore, even if the façade of this weather site changes, semantic addressing will still retrieve the correct data.

2. SYSTEM DESCRIPTION

We used two frameworks to compare semantic addressing to absolute addressing: seMole and absMole. SeMole and absMole are very similar: they both share components to interface to the Web and they are both Java Server servlets (the Java Web Server uses Java applications called servlets. Similar to cgi-scripts, they are fast and easy to prototype).

```
<getCNNWeather>
   <near>
      <fetchpage>
        "http://www.cnn.com/WEATHER/html/
        CambridgeMA.html"
      </fetchpage>
      "Wed.|Wednesday";
      "HIGH";
      "[0-9]*";
   </near>
</getCNNWeather>
```

Figure 1: A sample template to gather information from the CNN Weather site. The "fetchpage" runs first. It requests the Web parser to retrieve a page from www.cnn.com; its result is one of four arguments passed to the command "near." The "near" command then requests the semantic parser to find the high temperature forecast for Wednesday on the CNN Web page (i.e., integer nearest the word HIGH nearest the word Wednesday).

2.1. seMole

SeMole has three main components: the Web parser, the seMole language, and the semantic engine. The Web parser is responsible for interfacing seMole to the World Wide Web. It extends an XML parser developed by Microsoft [5]. This parser has been modified to parse HTML pages as well as XML pages. This allows seMole to gather data from any current page on the Web as well as any future pages written in XML.

The seMole language is the user interface into the seMole framework. Based on XML, the seMole language is a high-level scripting language that allows users to quickly develop semantic templates to gather information from Web sites. These templates map information found on a Web page to an XML reply. Each tag within the template represents a command. Control flows bottom-up (from child to parent) in each template (see Figure 1).

A template not only defines how information is gathered from a site, but it also defines how the information is returned to the user. After the completion of a command, the result of that command takes the place of the command originally in the template structure. Once completed, the template is returned to the user with only the results (see Figure 2).

The semantic engine is the heart of seMole: it is responsible for robustly finding information retrieved from the Web. The semantic engine relies on *semantic keys* to find information on Web pages. A semantic key is a *token* (a text tag in the HTML structure) that matches a word or a phrase in a frame. This word or phrase labels a *value* on the frame. In Figure 1, the "near" command uses the regular expression "HIGH" to find the semantic key "HIGH" in the HTML structure. This key, in turn, labels a value: in this case, an integer representing the high forecast. In most cases, a regular expression will match multiple semantic keys on a Web page. To find the best fit for this regular expression, various heuristics are applied to the Web page to find the best match to the regular expression. These heuristics include:

- How the key is formatted relative to the value. In most cases, a key has a "higher" typing than a value (e.g., a key will have a bigger font or it may be bold).
- How well the regular expression matches the keys.
- If there are multiple keys matched to one value, the distance between the keys and value determine which is the proper key. If there are multiple keys with multiple values, then *semantic groups* are used.

```
<getCNNWeather>
   <near_result>
      95
   </near_result>
</getCNNWeather>
```

Figure 2: The result of the template in figure x. Notice that the result of the "near" command ("near_result") took the place of the original "near" command.

To find groups of information, (e.g., multiple flight schedules) the parser makes use of *semantic groups*. While semantic keys define a relationship between a key and a value within a page, semantic groups define a relationship amongst groups of key/value pairs. For example, in Figure 3 we have a page with information about a multi-leg flight. One flight leg is one semantic group. Using seMole, we can define one template to find data on one leg. When this template is applied to the page, the semantic engine will apply the template two times without overlapping pieces of data across groups.

The Web parser, the seMole language, and the semantic engine allow a user to find information independent of the underlying HTML structure. Therefore, even if the same data is presented in a completely new HTML façade, seMole will still be able to find the data with the same template.

2.2. absMole

The absMole is an absolute addressing framework that we developed to test seMole's performance. It shares all the components of seMole, except for one: the semantic engine. In its place is an *absolute engine* modeled after the WIDL language developed by WebMethods. As specified in the WIDL whitepaper, WIDL uses a mix of regular expression matching and absolute addressing to find data [3]. Our absolute engine uses a group of absMole language commands to only allow for regular expression matching and absolute addressing. Although the language may differ from WIDL, the fundamental mechanism for finding data remains the same.

3. RESULTS

We compare seMole's semantic engine to absMole's absolute engine by creating templates for various Web sites. A *Web site* refers to any source of on-line information that presents HTML information about one domain (e.g., stock information). Each Web site that we considered had more than one page of information, with each page describing its own *target* (e.g., Microsoft stock, or Ascend stock). Each of these target pages is called a *frame* of data.

Each Web site was categorized in terms of topology (persistent or transient) and data (persistent or transient). "Topology" refers to how the data is presented in a frame. If a site has persistent topology, then the data is in the same place across all frames and each frame's topology is consistent through time. If the data is located in different places from one frame to the next, the topology is said to be transient. "Data" refers to the amount of data present on the frame. When the amount of data is the same across all frames, the site is said to have persistent data. If there is a discrepancy in the amount of data from one frame to the next or across time (the key, value, or both may be missing), the site is said to have transient data.

Flight #1 8/12/98

Departs: Houston, TX (IAH)	**Arrives:** Honolulu, Oahu, HI (HNL)
Gate: C-14	**Gate:** 10
Scheduled Time: 9:45 am	**Scheduled Time:** 12:45 pm
Actual Time: 11:10 am	**Estimated Time:** 1:45 pm
Status: In Flight	

Departs: Honolulu, Oahu, HI (HNL)	**Arrives:** Agana, Guam (GUM)
Gate: 11	**Gate:** 09
Scheduled Time: 1:45 pm	**Scheduled Time:** 5:25 pm
Estimated Time: 2:30 pm	**Estimated Time:** 6:10 pm
Status: Running Late	

Figure 3: A flight schedule for Continental flight #1 with two legs.

3.1. CNN Weather Site: Persistent Topology, Persistent Data

One of the first sites that we considered was the CNN Weather Site [6]. This site has over 6,100 frames of data for various cities worldwide. These frames largely have persistent topology as well as persistent data. We attempted to gather weather data using seMole and the absMole. Templates created for seMole were approximately half as long as the absMole templates. Because there were fewer commands to parse, the seMole templates usually ran twice as fast as the corresponding absMole templates.

Because the topology and data were persistent, both seMole and absMole consistently gathered the same data. The only discrepancy between the two frameworks occurred when the CNN Weather Web site changed its façade in March of 1998. Banners were added and tables were rearranged to update the appearance of the site. Further, important key/value pairs of data (e.g., "HIGH" and "72") were tagged with different fonts and typing. This, in turn, rearranged the HTML hierarchy that absMole relied on to gather information from the frame. Because of this, the absMole's template failed entirely. To fix the template, all references to the HTML structure had to be relabeled so that the template queried the proper parts of the HTML structure.

Although absMole's template failed, seMole's template continued to properly gather information from the site. This is largely because the semantic keys remained. Although they now had different fonts and typing, keys such as "HIGH", "LOW", and "WIND", still stayed relatively near the data that they labeled.

3.2. Cool Travel Assistant: Transient Topology, Persistent Data

Another Web site that we gathered data from is CO.O.L. Travel Assistant [7]. This site provided flight information for various airlines. Each frame presented on this site held information about one flight (see Figure 3). Each of these frames contained a variable number of legs for each flight (each flight had at least one leg). For each leg of the flight, there existed one table describing where the flight originated, ended, its estimated flight time, and its current status. Data was persistent within each leg, allowing us to predict what kind of data each leg had. Unfortunately, there was no information on each frame to determine how many legs to expect on each frame. This information had to be determined automatically.

We first attempted to gather data from this site using absMole. Because each leg of each flight was presented using the same table structure, we were able to define a simple subroutine template to gather information from one leg. However, to apply this subroutine to a variable number of legs, we needed to rely on the structure encapsulating the legs. From this structure, we directly determined the number of legs presented on the frame, and applied the subroutine to each leg's table. Unfortunately, this approach is vulnerable to many possible inconsistencies of the frames. For example, if banners or advertisements were added between the legs, this template would incorrectly determine the number of legs per flight. Further, we relied on the fact that one HTML node is a parent of the tables representing each leg. Therefore, if the parent node moved in the frame, or if each leg were presented using more than one table, this template would again fail.

We then gathered the same information using seMole. As with absMole, we first created a simple "subroutine" template to gather information from one leg on the frame. Like other seMole templates, this subroutine template used semantic tags to find airline information. Therefore, even if the information for one leg of a flight were spread over more than one table, this subroutine would still be able to properly group the leg's data. To capture all the legs within the frame, we took advantage of the seMole's semantic grouping ability. Instead of explicitly finding the number of frames by looking at the HTML structure, we simply created another template that applied the subroutine template across the entire frame. Since all the information for one leg was always grouped together, seMole was able to differentiate between two different legs and independently apply the subroutine template to each leg. This was accomplished without the aid of the HTML substructure. Since this template was created without any dependence on the HTML structure, it was very robust to any changes in the topology of the frames.

3.3. Boston Sidewalk: Persistent Topology, Transient Data

Another domain that we attempted to harvest is restaurant information. For this domain, we used the Boston Sidewalk site [8]. This site contains a repository of restaurants in the Boston area. The frames are organized under various categories (Chinese, Italian, etc.) and provide information about hours, payment options, parking availability, as well as reviews by Boston Sidewalk. Although the frames are topologically similar, not all the data is available for all restaurants. For instance, some frames lacked information about parking and therefore lacked a key/value pair for parking. Further, the Sidewalk review on each page lacked a key, making it difficult to locate.

Again, we first attempted to harvest each frame using absMole. The sidewalk review was always located in a certain location within the HTML structure, so retrieving the review was straightforward. However, retrieving the transient data (e.g., payment options and parking) proved to be more problematic.

	Robust-ness	Faster template prototyping	Running time
Persistent data	seMole	absMole	even
Transient data	seMole	seMole	seMole
Persistent Topology	seMole	absMole	even
Transient topology	seMole	seMole (semantic groups)	seMole
Many semantic keys	seMole	seMole	seMole
Few semantic keys	absMole	absMole	seMole

Table 1: A comparison between seMole and absMole for our three experiments. Each table element describes which framework (seMole or absMole) is better for that category. Note that the running time of absMole does not necessarily reflect running time of other commercial harvester applications.

To tackle this problem, we used several conditional statements to search for the transient data. Unfortunately, these conditional statements, as well as the statements used to retrieve the review, depended heavily on the structure of the HTML. The conditional statements queried a certain portion of the HTML structure. If the HTML structure changed even slightly, the queries would return garbage data.

Retrieving the transient data proved to be simpler with seMole. Since seMole only returns data that match the template, we created a strict template that never returned false data. For example, for parking availability, we queried the entire frame for the key "parking" followed by the regular expression "yes|no|none|street|garage". When the data was not present, this key/value pair was so strict that no false matches were returned. However, since there was no semantic key labeling the review, fetching the review from the frame proved to be difficult with seMole. Since we lacked a generic solution to finding the review on the page, we used an absolute addressing approach and fetched a certain part of the HTML structure that we expected to be the review.

4. DISCUSSION AND FUTURE WORK

This paper describes our efforts to develop a semantic addressing system for our WEBGALAXY architecture. Our experiments have shown that our semantic addressing system, seMole, is more robust and simpler to use than our absolute addressing system, absMole (see Table 1). This is because semantic addressing is able to abstract away from the HTML structure, allowing it to capture data when either the data or the topology is highly transient. In comparison, absolute addressing has proved to be cumbersome mechanism when dealing with transient topology or transient data because it is heavily dependent on the structure of HTML pages.

The only case where the absMole was more efficient to use was when the data was very persistent from one frame to the next. In these cases, the overhead required to do a semantic search through the HTML space was superfluous. Further, seMole was incapable of dealing with frames of data that have very few semantic keys. This is because almost all of the commands in seMole language rely on the user knowing what semantic keys to search for to find data. For example, in the Boston Sidewalk experiment, the review lacked a label. Without anything to identify the review, seMole could not robustly find the data on the page without resorting to absolute addressing.

Currently, seMole requires a large amount of user intervention to harvest data from the Web: users must study the target Web page, create a template, and test the template across many frames to check for robustness. In an effort to make harvesting Web pages simpler, future iterations of seMole will include a *dispatcher* which will automatically apply templates to a large pool of Web sites with only minimal user intervention. The dispatcher will take advantage of the fact that many of the templates that we have developed can easily be applied to various Web sites under one domain. For example, our CNN weather forecast template can also be applied to any other weather site that has day to day forecasts for US cities. By spreading information retrieval across multiple web sites, the dispatcher aims to make the seMole even more robust.

At an abstract level, a seMole template maps HTML to XML. XML is a burgeoning standard that will use HTML-like tags to label information on Web page [9]. XML is an attempt by the W3 Consortium to accomplish what HTML was originally intended for: to label data with semantic tags. Since seMole uses XML as its interface, seMole acts a link between HTML and XML. With its explosive growth, the Web will inevitably become a universally available, time-critical database. However, since many Web pages are still being written with HTML, these pages lack any structure to find and manipulate data. We believe that seMole is a bridge that overcomes this problem.

5. ACKNOWLEDGEMENTS

We would like to thank Giovanni Flammia and Victor Zue for their inspiration to develop absMole and seMole. We would also like to thank Raymond Lau, Stephanie Seneff, and Philipp Schmid for testing seMole and for developing templates for WEBGALAXY.

6. REFERENCES

[1] D. Goddeau, E. Brill, J. Glass, C. Pao, M. Phillips, J. Polifroni, S. Seneff, and V. Zue, "Galaxy: A Human-Language Interface to On-Line Travel Information," *Proc. International Conference on Spoken Language Processing*, pp. 707-710, Yokohama, Japan, Sep. 1994.

[2] R. Lau, G. Flammia, C. Pao, V. Zue, "WebGALAXY - Integrating Spoken Language and Hypertext Navigation," in *Proc. Eurospeech 1997*, pp. 883-886, Rhodes, Greece, Sep. 1997.

[3] C. Allen, "WIDL: Application Integration with XML," in *World Wide Web Journal*, Vol.2, Issue 4, Fall 1997. (http://www.agentsoft.com/whit.html).

[4] "AgentSoft and Automation Technology White Paper—Description and Comparison," AgentSoft Ltd., 1998. (http://www.agentsoft.com/compare.html).

[5] "Extensible Markup Language (XML)," Microsoft, 1998. (http://www.microsoft.com/workshop/c-frame.htm#/xml/default.asp).

[6] "CNN - Weather," Cable News Network, 1998. (http://www.cnn.com/WEATHER).

[7] "CO.O.L. Travel Assistant," Continental Airlines, 1998. (http://cooltravelassistant.com).

[8] "Boston Sidewalk," Microsoft Corporation, 1998. (http://boston.sidewalk.com).

[9] "Extensible Markup Language," World Wide Web Consoritium, 1998. (http://www.w3.org/XML/).