Exploiting On-line Data Sources as the Basis of Programming Projects

Peter DePasquale
The College of New Jersey
2000 Pennington Road
Ewing, New Jersey, USA
depasqua@tcnj.edu

ABSTRACT

The Internet is an exciting place to find real, interesting, and interactive sources of data for use in the classroom. This data (often real world), can provide the basis of interesting projects for CS1 and CS2 courses. Additionally, the source or exchange protocol can be used as a gentle introduction for novice computer science majors to the myriad of data formats and emerging technologies available today. Presented here are a number of interesting data sources used during the 2004-2005 academic year in our own CS2 data structures course.

Categories and Subject Descriptors

K.3.2 [**Computer and Information Science Education**]: Computer Science Education

General Terms

Design, Experimentation, Languages

Keywords

CS1/CS2, Programming projects, On-line data sources

1. INTRODUCTION

The Computer Science Department at The College of New Jersey typically offers one section of our data structures course each semester – a required course for our major and minor students which is taken within the first year of entrance into the program. During the semester, students are asked to complete six take-home programming projects as well as weekly closed-laboratory assignments (smaller programming assignments). Both the take-home and in-laboratory assignments are designed to strengthen familiarity and depth of knowledge of material covered during in-class lectures. While preparing for the Spring 2005 offering of the course, the author looked for ways to include emerging technologies in the course materials in some fashion. It was believed that the students were likely to be cognizant (or curious) of the emerging technologies such as news feeds, web services, and the like.

The decision was made to incorporate these technologies into students' take-home programming assignments as a vehicle for introduction. Rather than providing a fixed data input source (input file) as had been typically done in the past, real world data sources were tapped as the basis of the projects' underlying data.

The remainder of this paper will present the data sources used during the course offering – discussing the strengths and weaknesses of each. The manner in which the students integrated these input sources into their projects is also provided. Two additional sources (which will be exploited in the near term) are presented. Concluding thoughts and student comments are also included in the paper.

2. YAHOO!'S HISTORICAL STOCK QUOTES

One robust data source from which instructors can draw is Yahoo!'s historical stock quotes [7]. Provided as part of their financial coverage, Yahoo! allows unrestricted searching of historical stock, mutual fund, and indices quotes from financial markets around the world.

The basic stock quote interface will return a summary page of the most recent quote information, a chart of the recent performance of the stock and other informational links (options, charts, company profiles, etc.) Included among these links is a "Historical prices" link that takes users to a form from which they can further query the historical prices of a particular stock. By default this query interface will list the most recent 66 daily quotes. However, users can vary the query's starting and ending date, as well as request daily, weekly, or monthly data as the basis of the returned data set.

Once the desired query is generated, the interface lists a portion of the resulting data set (the first 66 results, with paging control to view the remaining data), as well as a hyperlink to download the data to a comma separated value (CSV) text file. The resulting data file contains the date, opening price, high price, low price, closing price, volume, and adjusted closing price for the stock queried over the time period requested. An example of the input file is shown in Figure 1.

```
Date,Open,High,Low,Close,Volume,Adj. Close*
8-Jul-05,37.87,38.28,37.47,38.25,10383400,38.25
7-Jul-05,36.81,37.76,36.80,37.63,13704400,37.63
6-Jul-05,37.71,38.16,37.20,37.39,14093800,37.39
5-Jul-05,36.55,38.15,36.50,37.98,16223900,37.98
1-Jul-05,36.83,36.97,36.29,36.50,8928600,36.50
30-Jun-05,36.61,37.16,36.31,36.81,14942500,36.81
29-Jun-05,37.23,37.29,36.12,36.37,16012800,36.37
28-Jun-05,37.49,37.59,37.17,37.31,12510700,37.31
27-Jun-05,36.84,38.10,36.68,37.10,21434700,37.10
24-Jun-05,39.09,39.12,37.68,37.76,14668200,37.76
23-Jun-05,38.83,39.78,38.65,38.89,24080500,38.89
22-Jun-05,38.26,38.60,38.14,38.55,15175900,38.55
21-Jun-05,37.72,38.19,37.38,37.86,13233100,37.86
20-Jun-05,37.85,38.09,37.45,37.61,11561300,37.61
17-Jun-05,38.47,38.54,37.83,38.31,21290200,38.31
16-Jun-05,37.19,38.08,36.82,37.98,19559800,37.98
15-Jun-05,36.87,37.30,36.30,37.13,20119400,37.13
14-Jun-05,35.92,36.15,35.75,36.00,12423100,36.00
13-Jun-05,35.89,36.61,35.82,35.90,15563300,35.90
10-Jun-05,37.40,37.40,35.52,35.81,24247600,35.81
```

Figure 1: Sample stock quotes in a comma separated value file.

This CSV file was used in the data structures course as the basis of input for an early project. Students were required to correctly parse the input file, and create a StockData object for each line of input (except for the first header line) that contained all of the information present on the input line. Each resulting object was then stored in a singly linked list in the order processed from the input file (the first item in the file was the first item in the list).

Following the population of the linked list, the students were required to process a series of commands from a second input file. A small sample of the commands include:

- printing a specific data field (open, high, etc.) for a given date,
- locating the object representing data on a specified date and calculating the change in the closing price for the three subsequent dates, and
- printing all of the dates (if they exist), where the closing price of the stock was within 2% of the closing price from the preceding day.

From the students' perspective, this data set is less complex to parse and integrate into a program than the Slashdot data discussed below. It does provide an interesting source for pedagogical purposes. For example, each line of input includes a trading date. Due to market closings (weekends, holidays, and catastrophic events) the dates are not contiguous. The data combined with Java's Gregorian Calendar and Calendar classes can be used to calculate which weekdays the market was closed or on which day of the week the input line refers.

The plethora of financial data can also be used to exercise student's output formatting of monetary amounts. Generally speaking, students will read the financial information as a floating-point value and store it as such. However, when outputting the same information (or a derivative such as the difference in the daily range of the trading price), we can introduce the NumberFormat and Locale classes (which will format a floating-point value as a currency amount based upon a particular locale).

One drawback to the Yahoo! historical stock price data source is that it is more easily exploited as a text file and not as a source directly from the web. This may permit the input file to be used as a data source for a CS1 course project (possibly with some data removed from the file to reduce the complexity of the object the students create to store the data).

3. GOOGLE'S WEB API

Google.com provides a freely accessible web service API to software developers who wish to integrate Google's search capabilities into their own applications [1]. This API utilizes Simple Object Access Protocol (SOAP [5]) and Web Service Description Language to query the Google search engine via a Java, Perl, or Visual Studio .NET application.

One project assigned in our Spring 2005 data structures course required the integration of this API into a larger application that prompted the user to repeatedly enter search terms (keywords) that were in turn sent to the search engine via the API. Search results returned from Google were placed in a simple list data structure in the order they were received. Following the construction of the list of returned search results, the students were required to perform a sequence of operations upon the list typical of a first data structures course (remove, insert, find, etc.). Students also handled simple error cases typical of list operations.

In order to accomplish this project, the students were required to obtain, or master each of the following.

- Each student was required to register for a free Google Web API account and license key. This provides each student's application the ability to issue search engine queries. Since queries to the search engine are limited to 1,000 per day per license key, each student was asked to obtain his or her own key and was not provided a single key from the instructor.
- Each student downloaded the Google Web API containing libraries in a number of languages, including the Java jar file that the students utilized for the project. The students used the API's jar file by including its location in the CLASSPATH variable during program compilation and execution.
- The students' applications then utilized the provided API classes to establish a connection to the server, provide the license key for the query being executed, issue the query request, and obtain a query result from the server.
- Queries issued to the search engine resulted in a GoogleSearchResult object being returned. These objects acted as collections of GoogleSearchResultElements and provided various meta-data regarding the query itself. Such data included the amount of time required to complete the query, if any filtering was performed on the result set, indexing information, etc.

Individual search result objects (GoogleSearchResultElement) contained specific information about a search result "hit". A summary of the data accessible in these objects is shown in Figure 2 for a single result "hit" on a query regarding "TCNJ Department of Computer Science".

For each query to Google's search engine, only ten results are returned at a time. Subsequent queries may be issued with incremented starting index numbers to obtain results beyond the first ten items.

```
URL  = "http://cs.tcnj.edu/"

Title = "The College of New Jersey <b>Computer</b>
<b>Science</b> Department"

Snippet = "<b>...</b> <b>computer</b> <b>science</b> and its
applications, the Department of <b>Computer</b> <b>Science</b>
at <br> the College <b>...</b> Phone: 609.771.2268 Fax:
609.637.5190  E-mail: cs @ <b>tcnj</b>.edu <b>...</b>"

Directory Category = {SE="", FVN="Top/Computers/
Computer_Science/Academic_Departments/North_America/
United_States/New_Jersey"}
  Directory Title = "College of New Jersey, The (<b>TCNJ</b>)"
  Summary = "Ewing, New Jersey."
  Cached Size = "13k"
  Related information present = true
  Host Name = ""
```

Figure 2: Example of data returned from the Google API on the query of "TCNJ Department of Computer Science".

One point worth noting in using Google as a data source is that there are discrepancies when comparing search results from the traditional web interface (`http://www.google.com`) and those obtained through the API. Specifically, each service provides an estimate of the number of search results available to the user. Thus, as students validate their results from an API query with Google's web interface questions may arise regarding the discrepancy of the number and order of returned search results.

As with the standard web interface to the Google search engine, the API search capabilities permit the use of a robust query language. The language supports a number of features including:

- site restriction – limits the query results to those hosted on a specified web site (e.g. site:www.tcnj.edu),
- filtering by file types – limits the result set to include only files containing a specified extension (e.g. .doc or .xls),
- Boolean OR – queries can include the logical OR operator to broaden a search, and
- filtering by links – limits the result set to only those pages which contain links to a specified site.

This programming project can easily be altered to take advantage of a relatively large amount of data or a different data structure. With the ability to submit subsequent queries to the search engine (with only the starting index value modified), a larger data set can be obtained. Instructors may find great value in passing significantly large data set to their students (as an input challenge to ensure that the program has been sufficiently implemented to handle truly dynamic volumes of data). Additionally, the project can be easily adapted to require the use of other data structures such as heaps or hash tables for storage.

In addition to being an interesting data source, the Google Web API provided an opportunity to provide coverage for a topic the students had not seen previously – the use of an external API via a jar file. While the students had become quite comfortable with the use of the Java API provided by Sun (java.io, java.util, java.lang, etc.); the Java API is essentially integrated into the programming environment. By using the Google API, the students were exposed to the issues of modifying their CLASSPATH to include packages from a third-party source at both compilation and run-time. Furthermore, the students were asked not to submit the Google API jar file as part of their deliverable, forcing them to follow the packaging and delivery specifications carefully.

The Google API is not difficult for the average CS2 student to master. Only one student (of fifteen) failed to complete the project. Contained in the Web API download is a sample Java program that the students were encouraged to examine closely. This sample program demonstrated creating a simple command line interface to the API and elaborated the sequence of steps required to query the search engine successfully. Their programs mimicked this behavior in addition to performing the required list manipulations also present in their project.

4. SLASHDOT'S SYNDICATION FEED

With the emergence of Really Simple Syndication (RSS), the task of obtaining a publicly available syndication (content) feed has become a relatively trivial exercise. These feeds are most often used to incorporate content from one web site upon another site. For example, web sites can easily incorporate headlines and links from the BBC's web site by obtaining the RSS from the BBC and reformatting the content and links to suit the design of the site. Additionally, feed aggregator applications can collect feeds for users wishing to scan the most recent content of a large number of sites quickly.

The RSS feed is a web-deployed Extensible Markup Language (XML) formatted file that is fairly straightforward to parse. In the final project of the semester, the students were asked to use the Scanner class (java.util.Scanner[1]) to establish a connection to, and read directly from, the feed available on Slashdot [3]. By obtaining their input data in this fashion, the students' applications no longer read from a static text input file, but rather from the existing data source in real time. The real time nature of their applications led to an in-class discussion of the need to thoroughly test the students' implementations – the news feed is completely dynamic and can vary as the news on Slashdot changes. This dynamic nature is in contrast to the relatively static data sequence returned from querying Google via the API mentioned earlier.

Once the feed stream was established, students were required to disregard the first portion of the file containing various meta data about the feed itself and extract various data regarding the web site's most recent news postings (the remainder of the feed's contents). Within the remaining data stream an `<item>` tag defines each news item and contains multiple child elements which provide additional data related to the specific news item. An example of an `<item>` tag is shown in Figure 3.

Once the source data was parsed, the students were required to place selected data from each `<item>` tag into a user-designed object and insert the object into a binary search tree based on the title of the news item. Following the completion of building the binary search tree, the students were then required to perform a number of operations on the tree, which ensured that their implementation

[1]The Scanner object, introduced in the 1.5 release of the Java language easily supports reading and parsing data from the keyboard, input files, and an InputStream object (which can be created from a valid URL).

```
<item rdf:about="http://games.slashdot.org/article.pl?
sid=05/06/11/1320233&from=rss">
<title>Mame Working on the PSP</title>
<link>http://games.slashdot.org/article.pl?sid=05/06/11/1320233&
from=rss</link>
<description>An anonymous reader writes "The PSP Wiki site has
posted a release of Mame the arcade emulator for the Sony PSP,
heres an english site with Screenshots of the Emulator in
action." I guess I won't need to go to such great lengths to
play Pac-Man next time.</description>
<dc:creator>CmdrTaco</dc:creator>
<dc:date>2005-06-11T14:10:00+00:00</dc:date>
<dc:subject>emulation</dc:subject>
<slash:department>oh-glorious-day</slash:department>
<slash:section>games</slash:section>
<slash:hitparade>4,4,3,1,1,0,0</slash:hitparade>
<slash:comments>4</slash:comments>
</item>
```

Figure 3: A sample Slashdot news item in RSS form.

of the binary search tree was correct. Such operations included correctly finding the minimum and maximum nodes in the tree, removal of a specific node in the tree, testing the presence of a specific node in the tree, and printing each node in the tree via one of several tree traversals (pre-order, post-order, etc.)

The use of the Slashdot RSS feed (or any feed for that matter) provides a nice introduction to XML during the CS2 course. Since XML files are essentially containers for data that provide structure to the data, the relationship to the topic of data structures ties in nicely to the traditional CS2 course. XML is rapidly gaining popularity in industry as a method for application configuration, interfacing with web services, and the transfer of data from one source to another. By incorporating XML into this portion of the CS2 course, the students gained initial exposure to this important emerging technology that may not be covered elsewhere in the curriculum.

The incorporation of reading an XML data source in a programming project provides an additional interesting option for instructors. There are multiple approaches for processing the XML source. My students utilized the Scanner class to parse the source file on a tag-by-tag basis. As a permissible alternative approach, advanced students were permitted to utilize the Simple API for XML (SAX) or Document Object Model (DOM) parsers provided in the Java API to parse their data source. (None took me up on the offer.) Thus, depending on the rigor of the project/course, one of several approaches could be undertaken to handle data extraction from the input source into the students program. The latter approaches (SAX and DOM) require additional instruction on the operation of each of the parsers, as well as the use of additional packages and classes from the API.

5. FORTHCOMING EXPLOITS

This author has begun to explore additional on-line data sources to exploit in upcoming course sections. Two sources have been identified: real time earthquake data and a cleaning house for open web services that may be queried. These two sources are discussed below.

5.1 USGS Earthquake data

The United States Geological Survey (USGS) Earthquake Hazards Program web site [4] provides a near real time map of earthquakes around the world. A series of publicly available RSS feeds continually list quakes larger than 2.5 magnitude over the last five and seven days, as well as quakes larger than 5.0 over the last seven days.

The earthquake data includes a magnitude value, date and time stamp in Coordinated Universal Time (UTC), latitude and longitude coordinates for the earthquake's rupture point, the depth of the quake, as well as a text description of the region of the quake.

The earthquake data source will be utilized as the basis of a project which will require students to convert the UTC value to the corresponding date/time stamp for their local time zone, as well as determine which of quakes listed were within a proximity of a predefined point. To accomplish the more complex second task, the students will need to examine each latitude/longitude coordinate pair and determine the distance to a given latitude/longitude coordinate pair. By using the Haversine formula, students can calculate the distance between two points on the earth through simple trigonometric calculations.

5.2 Going Further with SOAP

Lim [2] argues for the incorporation of web services into the CS1/2 curriculum and that such an integration is inevitable. We can begin to approach such integration and explore the impact of the coverage of these topics by first utilizing the available data sources web services provide.

Custom SOAP messages can be created within an application to query a large range of freely available web services. Xmethods.net [6] provides an extensive listing of third party web services one can utilize to obtain varied data ranging from a list of prime numbers, tips of the day, stock market data, as well as ZIP code-related queries. In conjunction with Sun's Enterprise Edition software, the SOAP with Attachments API for Java (SAAJ) provides a simplified API for authoring and parsing SOAP messages.

Figure 4 shows a portion of Java source code detailing the creation of a SOAP message used to query a random quote server. The corresponding SOAP response message is displayed in Figure 5.

```
// Create the connection
SOAPConnectionFactory scf = SOAPConnectionFactory.newInstance();
SOAPConnection connection = scf.createConnection();
SOAPFactory sf = SOAPFactory.newInstance();

// Create the SOAP message
MessageFactory mf = MessageFactory.newInstance();
SOAPMessage message = mf.createMessage();

// Create objects for the message parts
SOAPPart soapPart = message.getSOAPPart();
SOAPEnvelope envelope = soapPart.getEnvelope();
SOAPBody body = envelope.getBody();

// Populate the body of the SOAP message
Name bodyName = sf.createName("getQuote",
                "mrns0", "http://comp");
SOAPBodyElement bodyElement = body.addBodyElement(bodyName);
Name childName = sf.createName("HTMLformat");
SOAPElement textChild = bodyElement.addChildElement(childName);
textChild.addTextNode("true");

// Set destination and send the message
URL endpoint = new URL(
          "http://www.boyzoid.com/comp/randomQuote.cfc");
SOAPMessage response = connection.call(message, endpoint);
```

Figure 4: Creating a SOAP message in a Java application.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/
soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<soapenv:Body>
  <ns1:getQuoteResponse xmlns:ns1="http://comp"
  soapenv:encodingStyle="http://schemas.xmlsoap.org/
  soap/encoding/">
    <getQuoteReturn xsi:type="xsd:string">
      &lt;em&gt;"Some days you are the dog, some days you are
the tree."&lt;/em&gt;&lt;br&gt;&lt;strong&gt;Unknown
&lt;/strong&gt;
    </getQuoteReturn>
  </ns1:getQuoteResponse>
</soapenv:Body>

</soapenv:Envelope>
```

Figure 5: A SOAP response message to a quote server.

Beyond the XML coverage required to introduce the concept of the message format to students (not previously exposed to XML), we need only to define how each message is constructed. Each SOAP message contains three core components: the destination of the message, the method name and namespace of the query, and optional parameters to the method, if required.

The SAAJ API contains classes and methods to build and extract the SOAP request/reply messages from within an application. By using SAAJ, SOAP queries can easily be integrated into CS1 or CS2 programming projects thereby offering another method of introduction to SOAP and web services (as compared to the Google introduction presented in Section 3).

This approach to on-line data exploitation clearly requires additional programming beyond that seen in the Google-base project. The solution tested by this author however required inclusion of six additional jar files from the J2EE release and the SAAJ API into the compilation and execution CLASSPATH environment variable. While this added programming may not be suitable for all situations and students, it likely can be introduced to accelerated students or course sections as a follow up to the Google API project. The multitude of required jar files can also lead to an ancillary discussion of modular decomposition and deployment approaches.

6. STUDENT REACTIONS AND CONCLUSIONS

6.1 Student Reactions

The inclusion of these resources as data sources for the programming projects seemed to be well received by the students. Initially, the reaction by the students of the data sources was one of caution – clearly, they would need to become familiar with several technologies external to the core language and data structure concepts covered by the lecture material. However, after providing brief overviews of XML, RSS, and how the Google API operated (including demonstrating the sample application Google packaged with their API), the reactions were noticeably more upbeat.

The students from the course were emailed by the author to solicit comments (both positive and negative welcomed) regarding their use of the on-line data sources in conjunction with their programming projects. The most detailed response included the following:

> *"I thought, especially with the Google API, that using Java to extract real world data was very useful. I especially liked using the API, because it exposed us to using other people's pre-written classes, and made us try to figure out how they worked. Though I really didn't learn anything about how XML worked, I did like the fact that we were doing exercises that were based in the real world, rather than taking a big text file and picking out every time the word 'the' was present. Overall I enjoyed the experience of using Java in some realistic concept."*

6.2 Conclusions

The computer science curriculum landscape is becoming increasingly crowded. As our field and the technologies that support it continue to evolve, we will likely need to find creative ways of introducing some of these new technologies into the classroom without abandoning those topics of which we already provide coverage.

By utilizing selected on-line data sources in conjunction with CS1/2 programming projects, we can begin to gently introduce emerging technologies into the curriculum in the first year of study. Students respond well to this style of integration of topics of which they are likely to need at least a cursory knowledge upon graduation. Simple introductions to topics such as XML and RSS also provide a reinforcement of the use of structured data in real world applications. Such introductions can additionally be used to motivate students to further examine the finer points of these emerging technologies as areas of future study.

7. REFERENCES

[1] Google Inc. Google Web API (beta). Internet URL: `http://www.google.com/apis/index.html`, last accessed 9/9/2005.

[2] B. B. L. Lim, C. Jong, and P. Mahatanankoon. On Integrating Web Services From the Ground Up Into CS1/CS2. In *SIGCSE '05: Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*, pages 241–245, New York, NY, USA, 2005. ACM Press.

[3] Open Source Technology Group Inc. Slashdot RSS feed. Internet URL: `http://slashdot.org/rss/index.rss`, last accessed 9/9/2005.

[4] U.S. Department of the Interior, U.S. Geological Survey. Earthquake Hazards Program RSS feed. Internet URL: `http://earthquake.usgs.gov/recenteqs/`, last accessed 9/9/2005.

[5] World Wide Web Consortium. SOAP Version 1.2 Specification. Internet URL: `http://www.w3.org/2000/xp/Group/`, last accessed 9/9/2005.

[6] XMethods, Inc. List of publically available web services. Internet URL: `http://www.xmethods.com/`, last accessed 9/9/2005.

[7] Yahoo Inc. Yahoo! Finance Historical Stock Quotes. Internet URL: `http://finance.yahoo.com`, last accessed 9/9/2005.