Apatite: A New Interface for Exploring APIs

Daniel S. Eisenberg, Jeffery Stylos, Brad A. Myers
School of Computer Science
Carnegie Mellon University
dse@andrew.cmu.edu, jsstylos@cs.cmu.edu, bam@cs.cmu.edu
http://www.cs.cmu.edu/~apatite

ABSTRACT
We present Apatite, a new tool that aids users in learning and understanding a complex API by visualizing the common associations between its various components. Current object-oriented API documentation is usually navigated in a fixed tree structure, starting with a package and then filtering by a specific class. For large APIs, this scheme is overly restrictive, because it prevents users from locating a particular action without first knowing which class it belongs to. Apatite's design instead enables users to search across any level of an API's hierarchy. This is made possible by the introduction of a novel interaction technique that presents popular items from multiple categories simultaneously, determining their relevance by approximating the strength of their association using search engine data. The design of Apatite was refined through iterative usability testing, and it has been released publicly as a web application.

Author Keywords
API Documentation, Search tools, Browsing, Visualizations, Web applications.

ACM Classification Keywords
D.2.7 [Software Engineering]: Documentation; D.2.2 [Programming Languages]: Design Tools and Techniques – Software libraries.

General Terms: Documentation, Human Factors.

INTRODUCTION

Modern software engineers spend much of their development time using existing libraries, frameworks, and other Application Programming Interfaces (APIs). Over time, these APIs have greatly increased in number and in size. APIs like the Microsoft .NET Framework and the Java SDK contain thousands of classes and tens- to hundreds of thousands of methods and grow larger with every new version.

CHI 2010, April 10–15, 2010, Atlanta, Georgia, USA.

Exploring these new APIs and learning how to use them has become a daunting task. Many APIs are so large that no one person can be familiar with the entire system, so even experienced developers are often faced with the task of finding and understanding an unfamiliar part of an API. Programmers will often use web search engines like Google to help find which parts of the APIs to explore in more depth; however, studies have found this to be a powerful but imperfect strategy [6]. The web contains many relevant programming discussions and examples, but it also contains many irrelevant and unhelpful pages.

A popular strategy for learning APIs is to explore the official documentation, usually in a standardized format like Javadoc. Javadoc displays a hierarchy of all of the packages, classes, and methods of an API, but usually provides few cues besides the alphabetically listed package and class names to help users decide which classes to investigate. Studies of existing API documentation [1,4] have suggested that these hierarchies are insufficient when programmers have in mind a desired action but possess no knowledge of which class it might belong to. For example, programmers attempting to read from a file often begin by looking for a `read()` method. However, the `File` class does not contain any such method, and Javadoc provides no cues to indicate which of the many other classes in `java.io` are capable of performing this action.

This suggests that documentation interfaces can be improved by offering an alternative to the top-down, encapsulation-based browsing hierarchy that is usually employed. Apatite (**A**ssociative **P**erusal of **A**PIs **T**hat **I**dentifies **T**argets **E**asily) was designed with this idea in mind, and incorporates a number of novel features and interaction techniques. First, it provides all information in a graph, not a hierarchy, so users can go from one item to all associated items. Second, the items shown are filtered by popularity of usage, focusing the user's attention on the most likely answers. Third, given a selection of any item, Apatite initially shows the most popular related items in multiple categories. This quickly gives users an overview of the items, and provides affordances for drilling into any category. Fourth, users can explore an API by starting from methods and actions (the verbs) as well as properties, not just starting from classes or packages (the nouns) as in Javadoc.

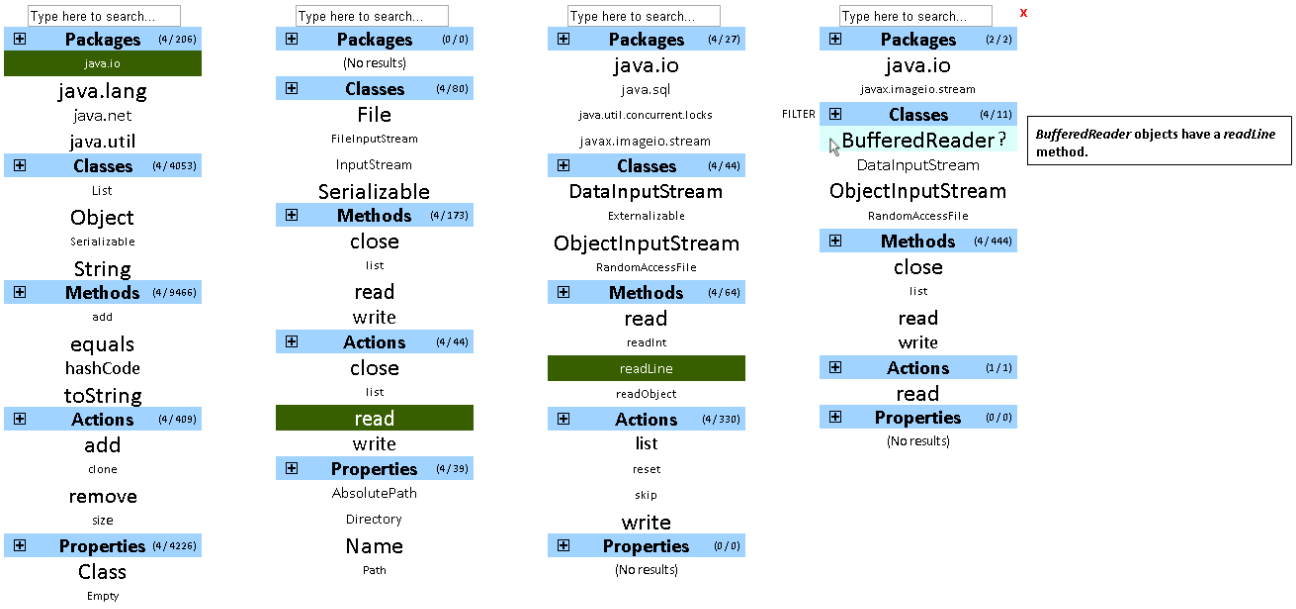

Figure 1. An Apatite use case demonstrating its core functionality. The user is looking for a class that is capable of reading lines from an external source. Initially, only the column on the left is visible. When the user clicks an item, a new column appears to its right containing elements that are associated with that selection.

DESIGN INSPIRATION

Apatite is heavily inspired by the associative browsing tool Feldspar [2]. Feldspar provides an interface to browse personal information like email, contacts, and events based on association rather than textual search. Employing a multi-column browsing interface, Feldspar enables users to filter a list by arbitrary dimensions in an arbitrary order (e.g., *people* mentioned in an *email* about an *event* last *year*), eliminating the need to remember specific details or enter any text queries.

The complication in adapting Feldspar to browsing a large API like the Standard Java 6 API is the sheer number of classes and methods to choose from; as noted above, users benefit from cues that differentiate between components that are commonly used and those that are obscure and/or irrelevant. The Jadeite Java documentation system [7] and other new work [3] provide inspiration for addressing this issue. Jadeite uses font sizes as they are employed in "tag clouds" by representing popular items with larger text. This feature proved to be very effective in the context of Javadoc-like documentation in Jadeite's user study. This idea, combined with Feldspar's interaction style, are the primary elements in Apatite's design.

INTERFACE AND USAGE

Figure 1 provides an overview of Apatite's interface when configured to browse the Standard Java API. When the tool is first initiated, only the column on the far left is visible. The column presents five categories, each of which represents a different "type" of API item that the user may want to start the search from: packages, classes, methods, actions, and properties. Under each header of this initial column, the four most popular items are listed alphabetically. A larger font size indicates a more popular item. For example, from the first column in Figure 1 we can see that `Object` is a more popular class than `List` because it is printed in larger text. If the user does not see an item of interest, the "+" sign can be clicked to expand a section. Alternatively, the user can type into the search box at the top if part of a component's name is already known.

When the user selects an item, a new column is generated immediately to the right. The categories here are the same, but now all of the items are ranked according to how strongly they are *associated* with the initial selection. For example, in the figure, the `Name` property is strongly associated with the `java.io` package. (See the following section for an overview of how these associations are calculated.)

The user can continue on from here, iteratively generating columns of associated items until the desired component is found. Figure 1 illustrates how a programmer might investigate how to read a line of data, progressing from the `read` action to the `readLine` method and finally to the `BufferedReader` class. Associations apply over multiple columns, so results that are related to *all* of the previous few selections will be ranked higher than results that are related to only *some* of the previous few selections. In Figure 1, `BufferedReader` is highly correlated with all of `java.io`, `read`, and `readLine`.

This example also demonstrates several advantages that Apatite has over traditional documentation schemes. In addition to providing the ability to start browsing from any place in the API hierarchy, Apatite exposes the user to

many other useful components associated with any selected item, thereby fostering a better understanding of how the API fits together. For instance, from Figure 1 we can see that the `close` method is often called near `readLine` and that the `java.sql` package also contains methods for reading data.

Apatite also allows users to easily backtrack along a specific search path and explore other interesting options. Clicking on an item in a previous column clears away all subsequent columns and continues the associative browsing from that point. Apatite automatically scrolls the page horizontally as the number of columns becomes large, allowing for arbitrarily long search paths while preserving access to all previous selections.

Extended Accordion Interaction

One novelty of Apatite's interface is its mechanism for drilling down into particular categories. As mentioned earlier, clicking the "+" in a category header expands that section's list to show the top 15 items. Doing so collapses the remaining categories, focusing the interface on the single extended list (see second column of Figure 2). The selected category can be expanded further, showing the top 30, 60, etc. items. Clicking the "-" in the header, or clicking on a different header returns the column to its initial state.

This interaction resembles typical "accordion"-style controls commonly found in web applications; however, it is unique in that it allows the user to initially view a few elements from *all* categories simultaneously. This feature gives users an overview of relevant items from a variety of categories and also provides an affordance for easily browsing through associations without being restricted to walking down specific paths of the hierarchy (such as the Package→Class→Method path provided by Javadoc).

This interaction technique also allows users to easily go up, down and across the API hierarchy. For example, the user can select an action, see what methods implement that action, see what classes contain that method, and then see what other methods are often used in each class.

Additional User-Inspired Features

Throughout Apatite's development, we performed a number of user studies to gauge the tool's potential and discover which aspects were in need of improvement. The current design is the result of an iterative design process that has addressed many issues that were identified by preliminary users.

The ability to search a column for particular keywords was added since it was a highly requested feature, allowing users to quickly jump to a particular item if they already have one or more words in mind. Filtering occurs instantaneously – the column's content adjusts with each additional keystroke, providing continuous results as the user refines the query. The search mechanism also handles multiple keywords by modifying the ranking system to reflect how

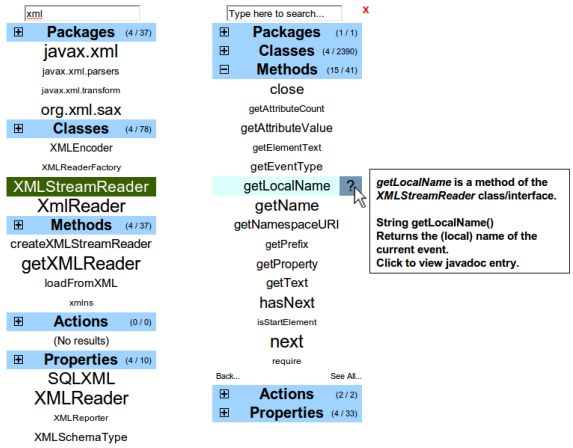

Figure 2. Demonstration of accordion expansion, text searching and informational popups.

many of the keywords each item matches. For example, given a search query of "data stream," classes containing *both* the "data" and "stream" keywords are printed in larger text than classes containing only a single keyword, although the relative popularity of each item is still reflected by text size within each classification.

Apatite also helps users learn about a particular API component in more depth. When the user hovers over a particular item, a "?" link appears to the right of its name. Hovering over the "?" displays a description of that item, with details varying depending on its category. For example, for a method in a class, the popup will contain its signature and Javadoc description (see Figure 2). Clicking on the "?" brings the user directly to the full Javadoc documentation. When a method or class is not unique, such as methods with multiple signatures, or when a method is selected where the class is ambiguous, then the popup shows how many different implementations exist, and clicking on the "?" brings the user to an intermediate page which provides links to each Javadoc entry that may be relevant.

Early users also wanted a better understanding of the relationships that Apatite tracks, so in the current version, rolling over an item provides a brief textual description of its relationship to the previous selection. This functionality is shown in Figure 1 – rolling over `BufferedReader` provides the explanation that it contains an implementation `readLine`, which had been selected in the previous column.

We have also provided a mechanism for explicitly choosing which types of relationships are displayed whenever multiple types of associations are represented in a single list. For example, the final column of Figure 1 is displaying classes that fall into three different categories: object types that are *returned* by `readLine`, object types that are *arguments* of `readLine`, and object types that *implement* the `readLine` method. If the user wants to view only a particular type of relationship, the "FILTER" link can be clicked to specify this in the resulting dialog box.

IMPLEMENTATION

A comprehensive description of Apatite's implementation is beyond the scope of this note (see [5] for a full explanation). However, the technique by which we determine the strength of association between pairs of API items is especially notable since it is fundamental to Apatite's overall effectiveness. The reason for designing Apatite as an *associative* search tool is to help guide the user towards API components that are commonly used together. Therefore, all of the associations have been derived from actual usage data collected from various public search engines.

The overall popularity of each API item, which is used by the initial column to display the most commonly used packages, classes, and methods, are determined by comparing the number of total Google results when searching for each item's fully qualified name. This technique had previously been used for the Jadeite tool, with effective results. The popularity of each *action* and *property* is simply the sum over all popularities of its respective methods (*e.g.* Popularity(*read*) = Popularity(`readInt()`) + Popularity(`readLine()`) + …). A method is categorized as an action if its first camel-case word is a verb.

To estimate the strength of association between two API items, we used a new heuristic. For each possible pair of methods, we counted the number of times one method's name appeared in the first 100 Yahoo! search results of the *other* method. The result is an approximation of how often the methods are used together in actual code.

All other association strengths are calculated from these two heuristics. When one item is encapsulated within another, only the overall popularity is used (e.g., when the `ArrayList` class is selected, the *method* category in the next column displays all of its methods ranked by their overall popularity). In all other cases, each item is decomposed into its implemented methods and the second heuristic described above is summed over each possible pair; this is then combined with the compared item's overall popularity to determine a final value. For example, to calculate how strongly the `StringBuffer` class is associated with the `BufferedReader` class, the association values between the methods that they implement are summed together, and then this number is weighted by the overall popularity of `StringBuffer`.

DEPLOYMENT

In June 2009, Apatite was released publicly as a web application and began receiving an increasing number of users. Among repeat users, the most popular feature has been the search function; users are commonly using Apatite to quickly determine whether a class or method exists with a name containing a particular keyword.

The public application is periodically updated with new data and features. Please try out Apatite for yourself: http://www.cs.cmu.edu/~apatite.

CONCLUSION AND FUTURE WORK

We believe Apatite has the potential to provide many additional contributions to the field of API usability. If deployed on a large scale, Apatite can be used to track common association patterns that are navigated by users. This could be used to aid API designers by revealing common use cases and helping them think ahead of time about which classes and methods will likely be used together.

Apatite's interface can be used to display any set of items, and there are many possible datasets that we feel would be interesting to browse in an associative way.

Finally, although we initially have targeted users who already know how to program, we believe Apatite might also be useful to those just learning how to program. A modified version for that audience might use *nouns* and *verbs* as the primary search categories and help learners build a clear mental model of how an API works. We are also exploring how to better extract verbs from Javadoc method descriptions and adding these words to the *actions* category.

ACKNOWLEDGEMENTS

This work was funded in part by a grant from SAP, and in part under NSF grants CCF-0811610 and CCR-0324770. Any opinions, findings and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect those of the NSF.

REFERENCES

1. Beaton, J., Jeong, S.Y., Xie, Y., Stylos, J. and Myers, B.A. "Usability Challenges for Enterprise Service-Oriented Architecture APIs," *VL/HCC'08*. Sept, 2008, Herrsching am Ammersee, Germany. pp. 193-196.
2. Chau, D.H. and Myers, B. "What to Do When Search Fails: Finding Information by Association", *Proceedings CHI'08*. Florence, Italy, April, 2008. pp. 999-1008.
3. Holmes, R. and Walker, R.J. "A Newbie's Guide to Eclipse APIs", *International Working Conference on Mining Software Repositories*. pp. 149-152.
4. Jeong, S.Y., Xie, Y., Beaton, J., Myers, B.A., Stylos, J., Ehret, R., Karstens, J., Efeoglu, A., and Busse, D.K. "Improving Documentation for eSOA APIs Through User Studies", *Second International Symposium on End User Development (IS-EUD'09)*, March, 2009. Siegen, Germany. Springer-Verlag, LNCS 5435, pp. 86-105.
5. Stylos, J. "Making APIs More Usable with Improved API Designs, Documentation and Tools." Ph.D. thesis, Carnegie Mellon University, May 2009.
6. Stylos, J. and Myers, B.A. "Mica: A Programming Web-Search Aid", *VL/HCC'06.* Sept, 2006, Brighton, UK. pp. 195-202.
7. Stylos, J., Faulring, A., Yang, Z., and Myers, B.A. "Improving API Documentation Using API Usage Information", *VL/HCC'09*, Sept, 2009. Corvallis, Oregon. pp. 119-126.