JQueryScapes: customizable Java code perspectives

[Forum Demonstration Proposal]

Lloyd Markle, Kris De Volder
Department of Computer Science
University of British Columbia
Vancouver, BC, Canada 604-822-1290
{lmarkle, kdvolder}@cs.ubc.ca

ABSTRACT

JQueryScapes is a flexible, query-based source code browser, developed as an Eclipse plug-in. JQueryScapes builds on top of JQuery. Like with JQuery a JQueryScapes user can define his or her own code views on-the-fly by formulating logic queries and running them against the source code. In this manner, JQuery provides the developer with a wide variety of crosscutting as well as non-crosscutting views within a single tool. Additionally, JQueryScapes allows the developer to create an unlimited number of JQuery views and link these views together into complex, multi-paned browsing perspectives called "JQuery scapes". We will demonstrate how JQueryScapes allows a developer to very quickly prototype a full-featured browsing environment tailored to visualizing and navigating crosscutting concerns based on project and situation specific structural queues such as coding idioms, naming conventions and custom Java 1.5 annotations.

1. RELEVANCE TO AOSD

IDE-based tools and code browsing tools in particular, can play an important role in the space of AOSD. This is because the IDE by and large controls how a developer can navigate and view their code. This means that a browser can present a modular view on something that is not actually modularized in the source code. For example, let's say a developer is working on the implementation of a print operation. The implementation of this concern is scattered across many different print methods declared in a number of different classes. Nevertheless, there is an implicit connection between all the methods: they all share the same method name and are connected to each other via an overrides relationship. It is possible for a browser to leverage this implicit structure to provide the developer with a single view presenting iconic representations of all the print methods organized in tree based on how they are related through inheritance. This view modularizes the print operation concern and lets a developer quickly navigate to different parts of the concern's implementation despite the fact that these parts are scattered in the source code.

Our demonstration will show how JQueryScapes is flexible enough to allow a user to very quickly prototype customized browsing perspectives, "JQuery scapes", that are tailored to visualizing and navigating crosscutting concerns based on project and situation specific structural queues such as coding idioms, naming conventions and custom Java 1.5 annotations.

2. PROBLEMS ADDRESSED

Although IDE-based code browsing and searching tools have great potential to help developers visualize and navigate scattered and tangled code, we believe that browsing and code searching tools in modern IDEs suffer from a number of shortcomings. In this section we will discuss these shortcomings. We will focus our discussion on Java and the Eclipse [6] JDT programming environment. However, JDT is a representative state-of-the-art Java development environment and similar arguments would apply to other modern IDEs.

A first problem is the inflexibility of built-in browser tools. Eclipse JDT comes with a number of built-in browsing views, but these views are too inflexible to let developers leverage implicit structure that is specific to their particular code-base. For example, Eclipse JDT comes with built-in a package explorer, a type-hierarchy viewer, and a call-graph viewer amongst others. The functionality of these browsers is hard-coded and therefore difficult to customize. For example, the functionality of a package explorer—what types of information it displays and what kind of hierarchical structure is shown—is hard-coded into its implementation. What we need, in order to be able to support browsing many different types of (crosscutting) concerns, is the flexibility to define browsers that organize code structure in many different ways, leveraging different kinds of structural information that may be present implicitly in a particular code base. As examples of possible sources of such structural information, consider naming conventions and project-specific Java annotations. Browsing tools such as a package explorer, type hierarchy view or call-graph browser are too narrowly targeted on visualizing a specific type of structural information and are therefore ill-equipped to help developers leverage the implicit structure induced by project specific naming conventions or Java annotations. As a case in point the Eclipse environment at this time provides very limited support for browsing and navigating code based on Java annotations

present in the code.

A second problem is that code searching tools lack expressiveness. Search tools have great potential in helping developers locate and work with precisely those scattered code elements they are interested in. However Eclipse's search tool often lacks expressiveness to be able to express a precise enough search, even if the developer may be able to formulate precisely what they are interested in. For example a query like "Find all references to `System.out` and `System.err` that occur outside of methods named `print`" can not be expressed.

A third problem is difficulty to integrate information from separate views. By providing a collection of specialized, separate tools to browse different types of information IDEs force a developer to manually integrate information across views. For example, Eclipse's package browser shows code structure in terms of Java packages. Eclipse type-hierarchy shows code-structure in terms of inheritance relationships. However, because these are completely separate tools, it is hard for a developer to correlate the information from these two views to answer a question like "Which subclasses of this class are in another package".

JQueryScapes addresses these issues by providing an expressive query language and a flexible browsing environment that allows users to set-up highly customized browser configurations to visualize and combine structural information in many different ways. Not only can we produce replacements for most standard Eclipse Java browser views with little effort, we can customize these views quickly at runtime. This allows us to quickly prototype and experiment with integrated browsing configurations that make use of project specific information, such as naming conventions or Java annotations, to visualize and navigate code structure.

3. DESIGN AND IMPLEMENTATION

3.1 Idea: Queries are everywhere

JQueryScapes is built on top of JQuery. The spark of insight that inspired the development of JQuery was a rather obvious observation that a major portion of an IDEs functionality consists of different GUI views displaying different types of information about program structure.

For example, Eclipse provides a multitude of browser-style views that display different information inside of a tree-widget. Superficially these views (package explorer view, type hierarchy view, outline view, etc.) differ only in the type of information that is being displayed in them. Essentially, this means we could regard these views as not much more than convenient GUIs for displaying and navigating the result of different types of program queries.

3.2 JQueryScapes: a generic browser environment

JQueryScapes exploits the program query idea to develop an "all-in-one" generic browsing environment. JQuery consists of a program database and a generic GUI browser tool for displaying and navigating query results. Users can define their own browser views simply by formulating queries. Alternatively they can invoke queries from a predefined set accessible through point-and-click menus. These menus and their associated queries are also completely configurable by the user. JQueryScapes ads to that the ability to create unlimited JQuery views and link them together into complex, "immersive" browsing perspectives.

A testament to the flexibility of this model is that it allows JQueryScapes to subsume most of the functionality provided by different Eclipse tools, such as the Java search tool, the package explorer, the type-hierarchy view, the call hierarchy view, the task/bookmarks/errors views etc.

JQueryScapes offers three major advantages over conventional browsers and code searching tools such as those provide by Eclipse.

Expressiveness JQueryScapes is built around an expressive query language which allows developers to formulate queries that are more precise than the Eclipse built-in search tool. For example, a query like "Find all references to `System.out` and `System.err` that occur outside of methods named `print`" can be formulated in this query language.

Extensibility (see [8]) In contrast with highly-specific tools, JQueryScapes is highly extensible and configurable. For example, the functionality of a package explorer—what types of information it displays and in what kind of hierarchical structure is shown—is hard-coded into its implementation. Consequently it is relatively hard to change or extend the package explorer functionality in a non-trivial way.

In comparison, in JQueryScapes generic browser implementation, the functionality of a basic package explorer is defined by a query expression of only three lines of code. These three lines are specified in a user-modifiable, dynamically loaded configuration file. They can also be accessed through a dialog box in the JQueryScapes GUI. Consequently, it is comparatively easy for a developer to gain access to the underlying query and understand and edit this browser definition. A similar argument also applies for the definition of new JQueryScapes browsers.

Integration (see [3]) We believe that JQueryScapes, thanks to its generic "all-in-one" browser model, provides a superior GUI design compared to the typical IDE. We claim that it more adequately supports a typical code exploration task. This is not as much because of the types of views provided, or the specific information displayed in them, because essentially all information that is present in the JQuery database is also available from standard Eclipse tools.

JQueryScapes superiority stems from having better integration in the following sense: rather than having multiple separate tools, JQueryScapes generic interface subsumes and seamlessly "blends" their functionality together into a single "integrated" tool.

The problem we see with Eclipse (and other modern IDEs) is that by providing a collection of specific, separate tools to browse different types of information

they force a developer to switch between different tool-views when performing realistic exploration. Switching views causes disorientation and forces the developer to manually integrate information across views. JQueryScapes on the other hand allows a developers to incrementally expand, refine or filter the contents of any JQuery view with different types of information without losing context.

4. IMPLEMENTATION AND TECHNOLOGY

There are two interesting points worth noting about the JQueryScapes implementation.

A first point is that JQueryScapes is modularized as a set of plugins. One plugin constitutes the JQueryScapes GUI. This plugin is built on top of an abstract API provided by a database backend plugin. The backend plugin provides an extension point to allow users to provide their own database backend and query language. Currently we have a default implementation that uses TyRuBa [1] as a query language and backend store. We have also implemented an alternate backend on top of JTransformer[4] which uses Prolog as the query language. The JTransformer provides a much more detailed view of the source code down to the byte-code level, but does not store information about code comments or other Eclipse generated code markers.

Both the JTransformer and TyRuBa backends use a Prolog-like syntax for queries. We are also working on a backend plugin that provides PDL [5] as an alternative. PDL syntax resembles AspectJ pointcut syntax and we believe it will make queries easier to write, especially for developers familiar with AOSD. For example, the query "Find all references to `System.out` and `System.err` that occur outside of methods named `print`" expressed in the TyRuBa syntax is fairly verbose:

```
class(?System),name(?System,System),
field(?System,?f),
(name(?f,out);name(?f,err)),
reads(?m,?f,?),NOT(name(?m,print))
```

In PDL this query is expressed much more concisely and intuitively as follows:

```
reads('System.out'||'System.err') &&
 !within('* print(..)')
```

A second point worth noting is that JQueryScapes views have a tight integration with each other, but also with Eclipse JDT. By defining a standard API JQueryScapes has the benefit that any view created which uses that API can effectively communicate. Whether it be through linking, drag and drop, or copy and paste methods, the views can all communicate because they share a common platform. We have also made the API compatible with Eclipse's JavaModel so that JQueryScapes GUI views work with all of Eclipse's current views.

5. RELATION TO OTHER INDUSTRIAL OR RESEARCH EFFORTS

We have already discussed in detail how JQueryScapes relates to mainstream Java code browsing and searching tools such as those found in the Eclipse IDE. In this section we discuss some other related work.

SemmleCode [7] is a code querying tool similar to JQueryScapes. Compared to JQueryScapes, SemmleCode has focused more on developing a user-friendly query language and a scalable query backend. JQueryScapes on the other hand has focused more on the flexibility and composabilty of its graphical user interface components, allowing developers to build-up highly customized browsing perspectives, "JQuery scapes", made up of multiple linked-together query views. We believe it would be an interesting idea to try and use SemmleCode with its .ql query language and scalable implementation as an alternative backend for JQueryScapes.

JQueryScapes derives from JQuery [2, 3]. JQueryScapes offers several major new features compared to JQuery. On the one hand we have made the JQueryScapes backend into a separate plugin. This opens paths for others to build their own backend, or to use our backend to other query-based development tools. The JQuery GUI itself has also been augmented. Most notable is that JQueryScapes supports creation of complex multi-paned "JQuery scapes".

6. WHAT THE AUDIENCE WILL SEE

The JQueryScapes demo will consist of two parts. The first part will demonstrate some of JQueryScapes abilities to create and link views together in a simple yet powerful way. We will show how a single JQuery view can display and organize different types of information in many different ways. We will also show how multiple JQuery views can be created, organized and linked together to create an immersive browsing perspective, a "JQuery scape". An example of a JQuery scape is shown in Figure 1. Note that all of the views shown in this figure, except for the Java editor are instances of the same general purpose JQuery viewer. What makes the views different is wholly determined by the way they are dynamically configured. Some of the views display global program structure and information (e.g. the view on the left show global package structure, the view on the bottom shows all errors in the project). Other views can show temporary query results (e.g. one view shows all reference to System.out) while others display information about the current editing context (e.g. the "editor selection", "supertypes+", "incoming calls" and "outline views"). These views are "linked" with the editor which means they are updated automatically to display query results relative to the current selection in the editor window. We will show how JQuery scapes like this can be created from scratch in minutes through the JQueryScapes GUI.

The second portion of the demo will show how we can leverage this flexibility to quickly tailor a browser perspective. We will show how, simply by putting together customized JQuery views, it is possible to quickly create a full-featured browsing environment that supports viewing code in terms of crosscutting feature implementations that are marked by

project specific annotations. A screen shot of this customized perspective is shown in Figure 2. On the left there is "Features" view. It shows the names of all features as indicated by annotations scattered around the code. The "Contents" view shows the scattered elements of the code that belong with the feature selected in the "Features" view. This information is also derived from Java annotations in the code that mark these elements specifically. On the right we have customized versions of an outline and outgoing calls view which organizes elements in terms of features as well.

7. HARDWARE AND PRESENTATION REQUIREMENTS

The JQueryScapes demonstration does not require any special hardware other than a projector and a screen. The presenter will come with a laptop prepared for the talk.

8. TYPE OF DEMONSTRATION

Forum.

9. REFERENCES

[1] K. De Volder. *Type-Oriented Logic Meta Programming*. PhD thesis, Vrije Universiteit Brussel, Programming Technology Laboratory, June 1998.

[2] A. Eisenberg and K. D. Volder. Jquery: Finding your way through tangled code. Demonstration, 2004.

[3] D. Janzen and K. D. Volder. Navigating and querying code without getting lost. In *Aspect-Oriented Software Engineering*, pages 178–187. ACM, 2003.

[4] G. Kniesel, J. Hannemann, and T. Rho. A comparison of logic-based infrastructures for concern detection and extraction. In *LATE '07: Proceedings of the 3rd workshop on Linking aspect technology and evolution*, page 6, New York, NY, USA, 2007. ACM.

[5] C. Morgan, K. D. Volder, and E. Wohlstadter. A static aspect language for checking design rules. In *AOSD '07: Proceedings of the 6th international conference on Aspect-oriented software development*, pages 63–72, New York, NY, USA, 2007. ACM.

[6] Eclipse website. http://www.eclipse.org/, 2001.

[7] Semmlecode website. http://semmle.com/, 2007.

[8] K. D. Volder. Jquery: A generic code browser with a declarative configuration language. In P. V. Hentenryck, editor, *PADL*, volume 3819 of *Lecture Notes in Computer Science*, pages 88–102. Springer, 2006.

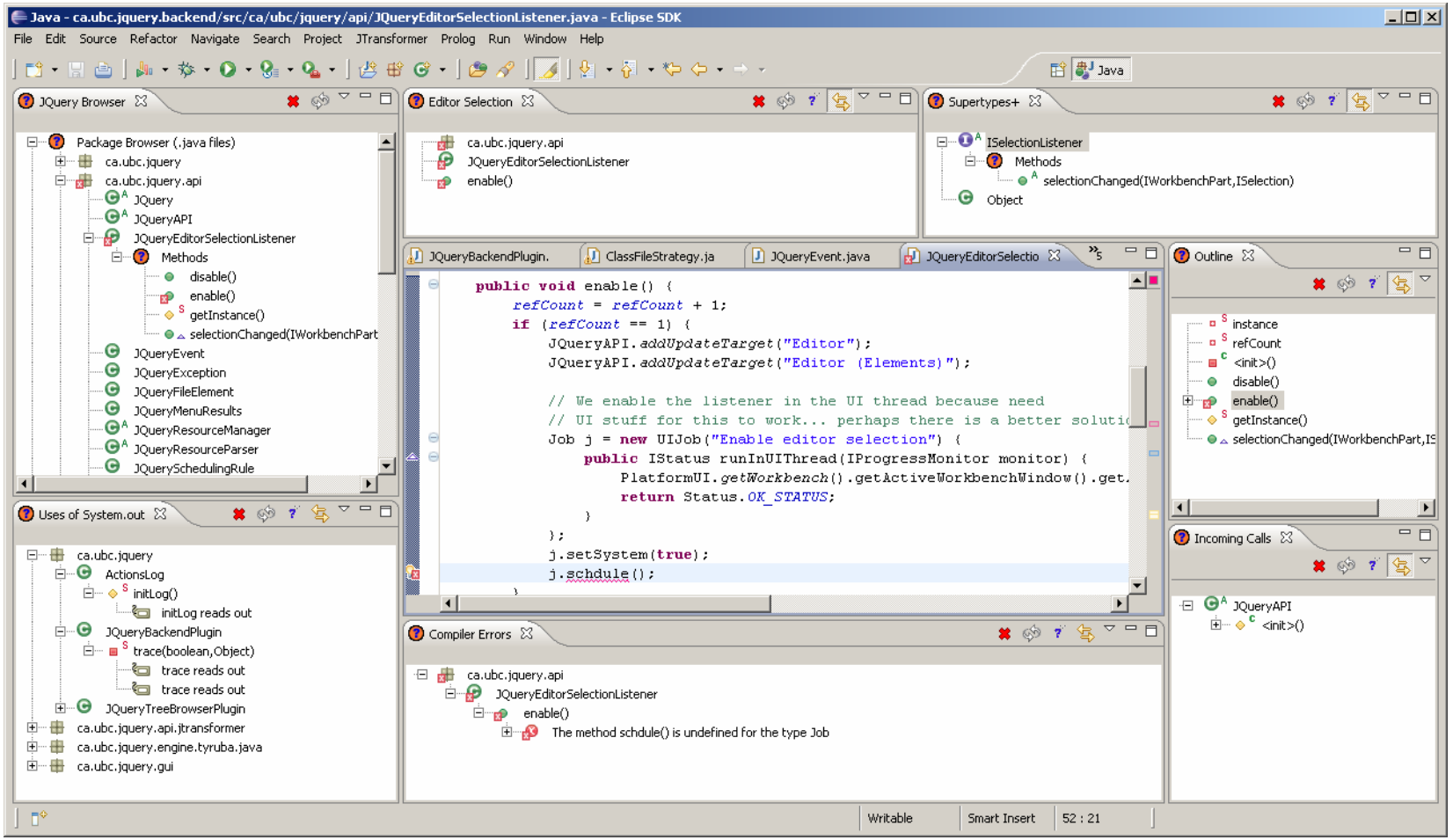

Figure 1: One of many possible JQuery perspectives.

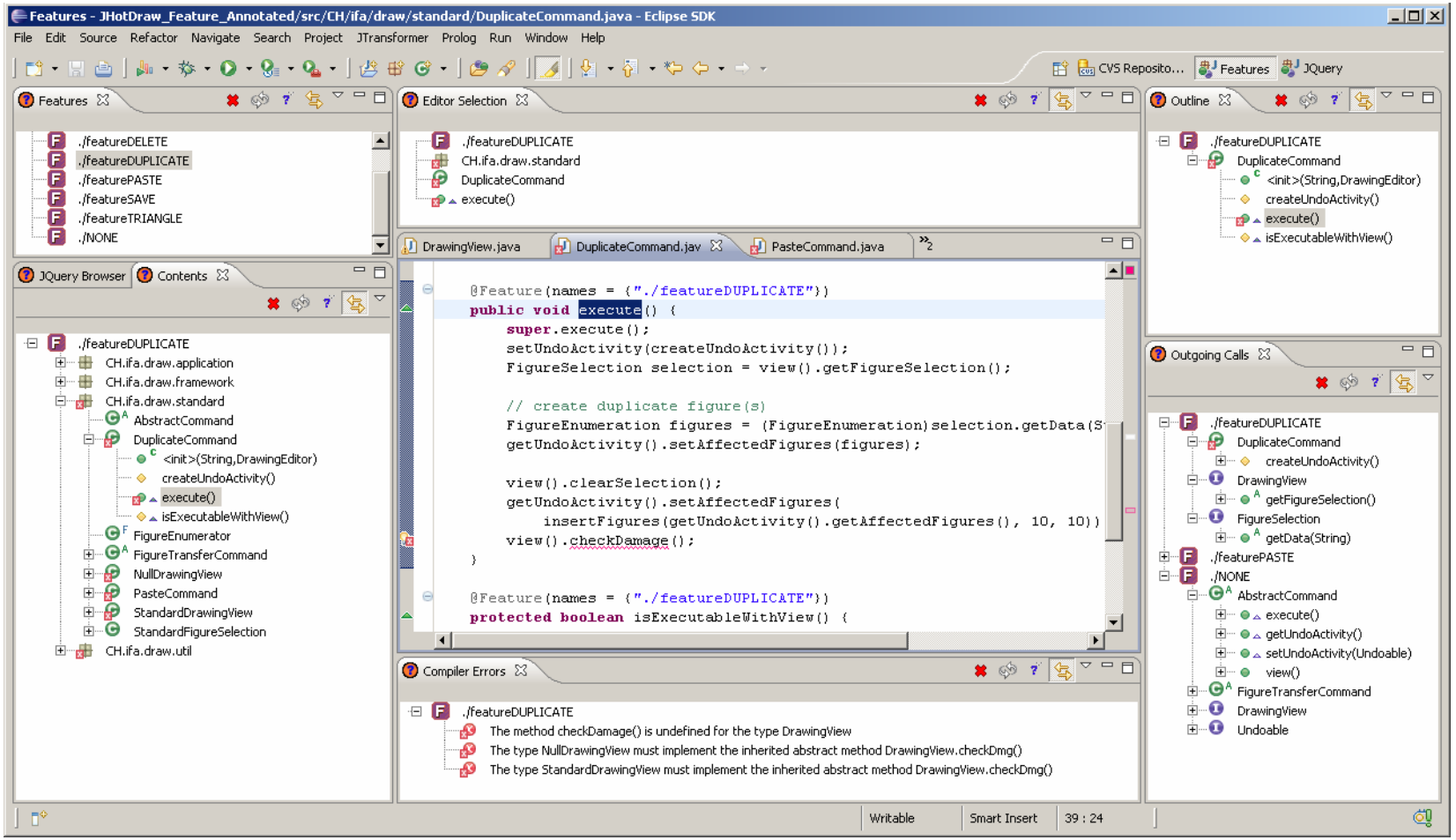

Figure 2: A customized perspective for browsing scattered features.