JQuery with Multiple Views

Edward McCormick
Computer Science
University of British Columbia
Vancouver, BC, Canada
emccormi@cs.ubc.ca

General Terms
Navigation, Visualization, Browser, Query Engine

Keywords
Object Oriented Programming, Aspect Oriented Programming, Modular Programming, Concern Separation, Tangling, Synchronized Views

ABSTRACT
In this paper we argue that JQuery's current tree hierarchy visualization for exploring query results does not provide enough contextual information for each node. Due to the unrestrictive nature of query-based exploration, the developer may often find himself viewing a set of code that does not pertain to his task. Should this occur, he must re-trace his steps until he finds the point in the query path where a 'wrong turn' was made. We propose that by providing non-queryable browsers side-by-side and sychronized with the query browser, the developer will gain useful information on how the element he is currently exploring relates to the rest of the system. We argue that use of our visualization will result in fewer missed opportunities for more effective queries as well as less disorientation during the re-tracing process. To support our claims, we have built a prototype which we call JQuery with Multiple Views.

1. INTRODUCTION
Many of todays IDE's, such as the Java Browser of Eclipse[1], allow the developer simultaneous use of multiple structural views such as method call hierarchies and type hierarchies as well as organizational views such as the outline view or package explorer. JQuery was built because although there may be a need for each of these views, continually switching from one view to the next can place a great cognitive burden on the developer. This is because each time a developer switches views during an exploration task, he must spend a moment re-orienting himself to the new view. This can lead to disorientation and confusion, delaying or even impeding the developer's progress.

This phenomenom, which the authors of [2] call **Explosion of the Browsers**, is addressed in the JQuery tool by allowing the developer to perform his exploration of the system entirely within a single browser. To use the tool, the developer first creates a browser by writing a recursive query describing the elements of the system he is interested in exploring. The query engine executes the query and returns the result as a tree hierarchy. The user is then able to browse this hierarchy, and execute context specific queries on any node of interest. Each query results with the tree updating the queried node with child nodes representing the results of the query. In this way, a great deal of information can be viewed within a single view, combatting the issues associated with an 'explosion of browsers'.

JQuery is a query-based source code browser for Java. Using JQuery, a developer is able to create a hierarchical browser of an Object Oriented system by writing a query. He is then able to perform successive queries on the nodes of the browser to form horizontal query paths. Although we find JQuery to be a highly useful tool, we believe that presenting the results of a developer's contextual queries as a series of child/parent relationships branching out from a structured browser is unsatisfactory. Following a query based exploration of the code, the developer is presented with a tree hierarchy whose elements relate to other elements of the code in more ways than are visible. We argue that this limitation may become a great hinderance when the developer must utilize the information to affect a change in the code. For example, perhaps the developer unknowingly visited the same node twice, causing an unnoticed recursion in his query path. Or perhaps there are relationships between explored elements which are absent from the path because the developer did not query for that information directly. In each situation, the developer may become confused and create an error in his code. We believe that such situations reveal a limitation inherent in trying to fit a multi-dimensional graph into a two dimentional tree structure.

In this paper we address these issues by presenting an alternative visualization for JQuery. In our visualization, which we call **JQuery with Multiple Views**, the developer will focus mainly on the query path in order to escape the drawbacks associated with an *explosion of browsers*, but will also be given the option of creating non-queryable, well-

structured browsers of the system to mirror his current position in the query path. Well-structured views are views in which all parent/child relationships are of the same type. These optional views present a means by which the developer can quickly re-orient himself within the system as a whole, while gathering useful information about the currently selected node. We show that by revealing contextual information for nodes in the query path, the developer may quickly gather how the current node fits into the system as a whole.

2. JQUERY

2.1 Design Motivations

Prior to redesigning JQuery, an evaluation of the current version was performed in order gain a better understanding of the motivations behind its current design. Through our own experimentations and regular interviews with the lead developer of JQuery[1], we were able to identify the following three high priority features.

One: The browser must take up as little screen real-estate as possible, in order to leave room for a code editor. We observed that an expert user of JQuery set up his workspace as shown in Figure 1. This layout appears to allow ample room for both the code editor and the JQuery view. However, as the query path begins to grow, we found that the developer needed to scroll left to right in the query window in order to view the entire query path.

Two: Queries that do not reveal useful information must be easily pruned from a query path. It is often the case that the developer performs a series of queries that do not result in nodes pertinent to his task. If these results are left in the query path they may cause confusion later, should the developer need to retrace his steps.

Three: The chronology of a query path must be easily intuited from the visualization. With a tree hierarchy the developer is able to discern query nodes from query results by the color of the icon next to the node. Query result nodes are always displayed as child nodes of query nodes, and query nodes are always displayed as children of the node that the query was performed on.

2.2 Typical Use

We observe in [2, 5] that the typical layout chosen for Eclipse when using JQuery is the *Java Perspective*. Figure 1 depicts a typical scenario, with the JQuery view vertically alligned with the left-hand side of the screen and the remainder of the screen used by the code editor. As described in [2], there is a distinction between a *view* and a *browser* in JQuery. In the figure, the *view* is labeled **Tree 1: spacewar**, and contains a query window, labeled **Active Query** and a console, which provides a means to relay error and log messages. JQuery provides the developer with vertical and horizontal scroll bars because, as shown in the figure, the length of a query path often grows exceeds the width of the view. Each view can contain multiple browsers. The **Active Query** view contains a browser called **Package Browser**, which has been opened to reveal all classes contained within the *spacewar* package. Beneath the class *Player*, there are two sub-queries named *Fields* and *Inverted Hierarchy*. In this example, the user has become interested in *Pilot*, a superclass of *Player*, which was returned as a result of the *Inverted Hierarchy* query. By double-clicking on any of the nodes in the browser, the developer can view the selected program element in a code editor.

[1]Kris DeVolder was a lead developer on the project and continues to use JQuery for all his development tasks

3. JQUERY WITH MULTIPLE VIEWS

A developer may choose to open as many views as he wishes, but there must always be one Active Query view, within which he may make context specific queryies on nodes. To add or remove a view, he may select a menu option. This may be done in the middle of a navigation if necessary; the new view will automatically synchronize itself with the other views.

In our prototype, we allow the developer to synchronize the views by selecting a particular node of interest in any of the views. If the selected node appears in any of the views, it is revealed and highlighted with a particular color. If there is more than one copy of the node in a view, the highest node up in the view will be revealed. Copies of the node existing even within the query path itself would be highlighted, in order to alert the developer that he has already come across this node previously in his exploration and to keep him redundant exploration. The user is able to synchronize views by up to five nodes at any time. When a node is selected, the icon next to it and any nodes that match it changes to one of 5 colors *Blue, Green, Grey, Magenta* and *Yellow*. If the user wishes to clear the five selections, he may select that option from a menu and all nodes will revert back to their original color.

The typical Eclipse layout used by a JQuery user places the JQuery window in a vertical position side-by-side with the code editor. Using this layout, when queries are performed on nodes, the user must scroll left-to-right to examine his full query path. We suppose that this particular layout is preferred due to its similarity to the Eclipse Package Explorer that most Eclipse users are used to[2]. Our first decision in designing JQuery with Multiple Views was to model it more on the *Java Browser* perspective of Eclipse, which is similar to the SmallTalk Browser[4]. Using this perspective, structural views are presented side-by-side at the top of the screen above the code editor, which fills the lower half of the screen. We believe this layout is better suited to the horizontal query paths of JQuery.

4. USE CASES

The following scenarios highlight the usefulness of using multiple views with JQuery. Each case was built from real-world experience the author had in adapting a game called Spacewar to meet his needs.

4.1 Querying with Multiple Views

Tom recently downloaded the source code for a game called Spacewar. He wants to achieve a high score, and does not have time to practice. To achieve his goal, Tom decides

[2]The layout decision was not mentioned in any of the JQuery literature

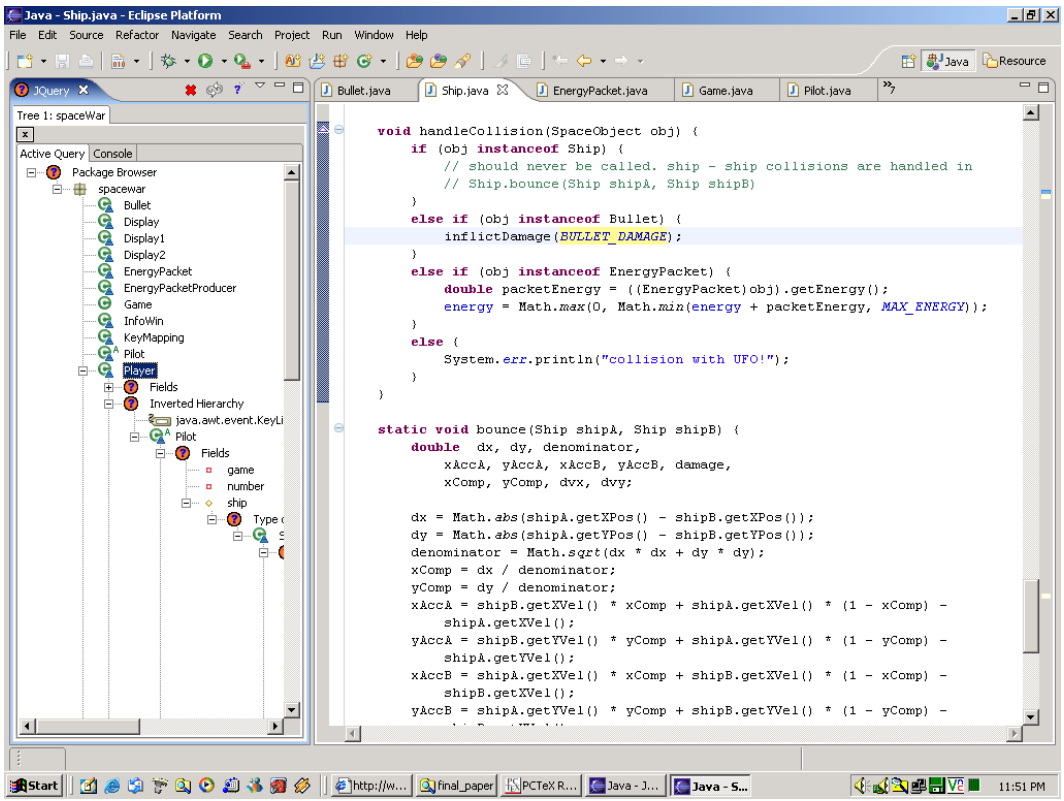

Figure 1: A Typical Query Setup

to manipulate the code so that his player will not lose energy when hit with bullets from the enemy. Tom decides that JQuery with Multiple Views will help him in this task, so he opens up an Active Query window and two other browsers. In the Active Query view he opens a Package Explorer Browser. He then creates a Call Hierarchy browser in one of the views and a Field Access browser in the other. The Call Hierarchy browser lists all methods in the Spacewar program, and the child nodes of each node are the methods that call it. The Field Access browser contains three levels: it lists all fields in the Spacewar program with the locations in the source code where these fields are written or read as child nodes, and the method where that source location exists as the third level. Tom then begins his task of figuring out how to keep his energy from being lowered during play. The browser created by Tom is shown in Figure 2.

Tom begins his task by examining the package explorer. He finds a class called *Player* there and decides he has found a good starting point. He queries for the fields of *Player* first, hoping to find a field that might be connected with an energy field. Since that does not exist, he queries for superclasses of *Player*. He finds that a super class exists, called *Pilot*. He queries for *Pilot*'s fields and finds one called energy. He double-clicks on this field and the node energy is revealed in Green in the field accesses browser in a separate view. He searches this browser for locations in Spacewar where this field is written or read and finds a method called **handleCollision**. He double-clicks on this method and the node turns light grey. As a side effect, the method call hierarchy browser is updated to reveal multiple instances of grey nodes - signifying that **handleCollision** contains multiple call sites. One of these call sites refers to a method called **inflictDamage**. Tom double clicks on this node and decides to read the associated code in the code editor. He finds that this method, contained in the class *Ship* does in fact decrease the amount of 'energy' the ship contains. Tom decides that he now has enough information to complete his task. He opens the **handleCollision** method and places an *instanceof* check to see if *Pilot* is a *Player*. If this check returns true, he does not allow a call to **inflictDamage**. Tom runs his code and finds that it works as he had hoped.

4.2 Building a Multi-dimensional Browser

Edward has just downloaded the Spacewar source code. He is a long time fan of the game, but he believes it would be a lot more fun with sound effects. He decides he would like to add sound to the game, and he wishes to start by playing an audio file every time he fires his gun at the enemy. He is an experienced user of JQuery with Multiple Views, so he begins be creating a few browsers he feels will be helpful for his task. He decides that the browser he will use as an Active Query View will be a Package Explorer. He then creates a Class Members browser, which lists all the classes by package and all of their members (methods and fields) as child nodes. Next he creates a Class Creation browser, which lists all classes that instantiate other classes, with the instantiated classes ordered by the method within which they are instantiated. Lastly a Type Hierarchies browser is created. This browser lists all types that have subtypes. The

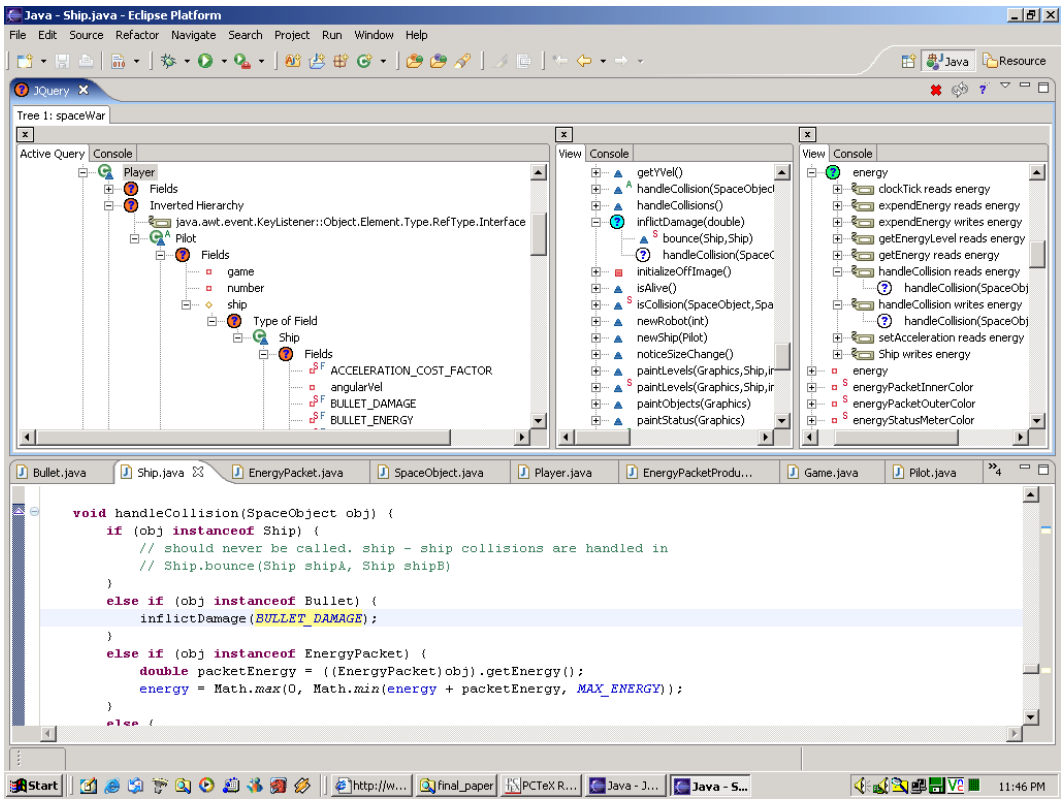

Figure 2: Tom's Browser

subtypes are child nodes of the parent types. The browsers Edward created are shown in Figure 3.

Edward begins his task by selecting *Bullet* in the Package Explorer. The *Bullet* node turns grey, and Edward notices that *Bullet* has shown up in the other three browsers, but most interestingly in the Class Creation browser. There, he finds that *Bullet* classes are instantiated in the *Ship* class from within the method **fire**. Edward is now interested in *Ship*, so he selects that node. He finds that *Ship* is revealed in all of the browsers, but once again he is drawn to the Class Creation browser. There, he finds that *Ship* objects are created in the *Game* class in a method called **newShip(Pilot)**. He looks at the code for this method and notices that the object of type *Pilot* is passed into the *Ship* constructor as an argument. He browses the Class Members browser under *Ship* and finds that it has a field of type *Pilot*. He feels that perhaps the argument to this method, which is of type *Pilot* may also be useful to learn about, so he finds *Pilot* in the Package Explorer and selects it. He finds this time that his attention is drawn to the Type Hierarchies browser. He notices that *Pilot* is an abstract class with two subclasses: *Player* and *Robot*. He feels that he has now successfully created a mental map of his task: when the field *pilot* of class *Ship* is of type *Player* and the method **fire** is called, the sound should be played. Edward inserts a method call at this point, using an instanceof test for *Player*, and then plays his game.

5. IMPLEMENTATION

JQuery was developed as an Eclipse plugin. It uses the TyRuBa[7] query language as a back-end for parsing Java files, building menu options and querying/updating the database. JQuery is written in Java and comprises 45 classes and about 3500 lines of code. Approximately 600 lines of code were modified or added to version 1.02 of JQuery to implement the mulitple views. To synchronize the views, JQuery with Multiple Views generates and performs a query on each of the views. Each node that is returned is then found in the tree hierarchy and updated to the current color.

We felt that it was important to involve the query engine rather than simply scanning a tree for items with the same name as the chosen node for two reasons. Firstly, using this method we are assured to retrieve only nodes that match in identity as well as name. If we were scanning the tree alone there is a possiblity of incorrectly selecting a node with the same name but a different identity than the current selection. Secondly, by using TyRuBa as a back end for these views we have allowed space for future modifications of the tool. Some of these will be discussed in the future works section.

6. RELATED WORK

There are currently many tools that use linked or filtered views to protect the developer from the disorientation caused by an information overflow. This section will focus on the most relevent of these and compare the methods they use to JQuery with Multiple Views.

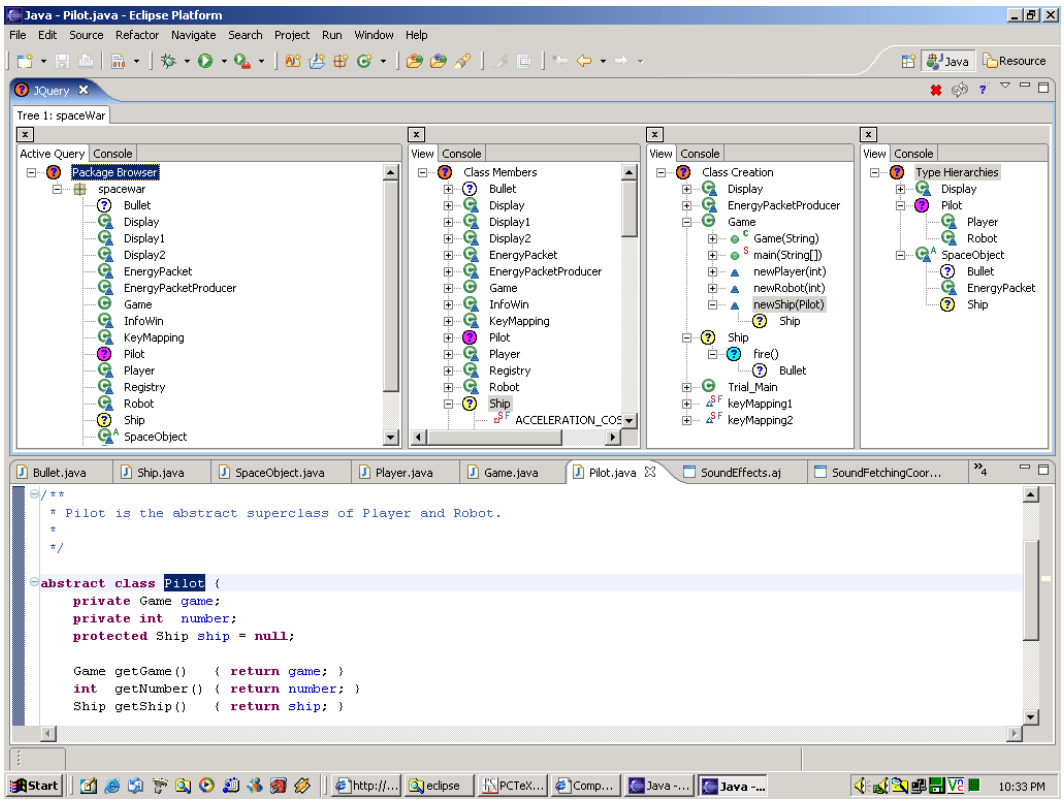

Figure 3: Edward's Browser

The FEAT tool[6] allows the developer to iteratively define the set of all code elements relevent to a feature the developer is interested in. FEAT allows the developer to use a set of pre-defined linked views - the most dominant of which are the *Participants* and *Projections*, to incrementally map out all the code relating to a specific concern. The developer begins with empty views and must explore the code using *fan-in* and *fan-out* queries available in the Projections view. Each time the developer is able to identify an element pertinant to the concern, he can add it to the Participants view. When the developer has fully mapped out the concern in these views, he can use the information contained within them to make changes to the code. JQuery with Multiple views differs from FEAT in the way that the developer uses the tool. JQuery with Multiple views does not build an explicit map of the concern. Rather, our tool is meant for exporatoring the code and making incremental changes as the user explores. Our tool therefore allows fully adaptable views, whereas the FEAT tools were designed and pre-defined for the purpose of the concern mapping.

Mylar[3] is similar to JQuery with multiple views because of it's use of color to link views and the dynamic quality of its browsers. Unlike JQuery with MV however, the browsers react as a side-effect to the developers exploration of the code. As the developer explores code using any of the views (including searching for specific code elements or viewing code errors), a Domain of Interest model is built and maintained, and the views are synchronized to this model. Using JQuery with MV, this task is more explicit. The code the developer is interested in is shown in the Active Query view as query results, and the contextual views react to the set of elements the developer selects in this view.

The Java Browser perspective of Eclipse has a very effective set of linked views. This perspective uses a set of pre-defined linked views that give the developer a structural understanding of the code he is working with. The method that the perspective employs is pure filtering. The views presented to the developer are the projects view, the package view, the type view and the members view. Since the information contained within the views is hierarchical across views, if the developer selects an element in any of the views, all child views are updated to reveal the members contained within that element. For example, a developer selecting a type in the type view will find that the members view is updated to show all members contained within the selected type. Although the nature of this set of browsers is similar to that of JQuery with Multiple Views, the user is unable to create new browsers, or to query the elements contained within any of the browsers.

7. LESSONS LEARNED

The central lesson we learned through our experience developing JQuery with Multiple Views was that color was not an appropriate method of linking the views. Because a user-defined browser can contain many occurances of the same node, it was often the case that a selected node matched too many nodes in the other views for the developer to quickly gain any useful information. The effect of so many nodes

changing color simultaneously across the screen appeared disorienting as well.

We believe the most useful manor of addressing this issue would be to link the views by filtering the amount of information shown in each view as they are synchronized. As an intermediate step that would require no additional coding, we believe our choice of colors could be greatly improved. We believe that less jarring colors would allow the developer to see the highlighted nodes without distracting him from his current task. A currently available tool that offers a customizable combination of shading and highlighting with color is Mylar.

Our project would have been more successful if we had used a more iterative, user-centered design process. Although we were able to consult with both a JQuery developer and an expert user, we were unable to show either of them a low or high fidelity prototype until we had reached our development deadline.

8. EVALUATION

We have successfuly built an Eclipse 3.0 compatible version of JQuery with Multiple Views. Due to some limitations we discuss in the following section, we do not believe the current version of our tool is a practical alternative for most developers. However, we do believe it may be very useful as a research tool. Many software tools offer synchronized browsers to assist the developer, but none allow the developer such a great deal of freedom in creating browsers. As we discuss in our future work section, we plan to place the tool in the hands of experienced JQuery developers and see what types of multi-dimensional browsers they invent to fit their various needs.

8.1 Strengths

JQuery with Multiple Views has leveraged the powerful back-end database of JQuery to allow the developer a freedom in creating browsers that no other tool allows. The process of creating and linking views using our tool was designed to be intuitive to a JQuery user. Additionally, the interface for creating a browser was taken from the original version of JQuery, which involves writing a query and ordering the bound variables. To link the browsers the developer simply needs to double click on any node.

The inherently adaptable qualities of JQuery are carried over to JQuery with Multiple Views. Using our tool, a developer is able to create browsers that span any number of dimensions he wishes. For expample, using the Eclipse Java Browser, the user is able to make use of a *Types* browser and a *members* browser. A developer exploring a system with JQuery with Multiple Views may wish to compound these views to create a single browser that contains all types with child nodes representing all members of a selected type. This may become useful when a developer who is using the Active Query window to explor type relationships in the system also wishes to see the members of any type he finds.

With these characterisics, we believe that JQuery with Multiple views may be very useful in many domain specific tasks. For example, when exploring a distributed system, the developer may wish to browse through co-located code, but to be informed (via a browser in a separate view) when a method he is exporing contains calls to middleware or across the network. Further research must be conducted before we can fully comprehend the practical applications for our tool.

8.2 Weaknesses

Although JQuery with Multiple views appears to be quite useful, we believe it can be improved upon. We noticed two major weaknesses with the current design. Firstly, using color to synchronize views does not appear to be effective enough. Although the colors to appear to grab the developers attention, it may often be disorienting when the same color appears in multiple places both within browsers and across views. We believe that a new synchronization method may be necessary. One possible solution to this problem involves filtering the nodes displayed in each browser. This idea is further explained in the future work section.

Secondly, we believe that the user interface for the views should be more adaptable to fit the developers needs. The developer can currently control the layout of the views by adjusting the number of views, the size of the views, and what browsers are contained within the views. The operations available to the developer are to synchronize the views by up to five nodes, to de-synchronize all of the nodes in all of the views at once, and to query on any node in the Active Query browser. We believe that although these are all useful features, we must add to them before JQuery with Multiple Views can become a practical alternative for developers. We plan to allow the developer more freedom in controling the layout of the views - through a drag and drop mechanism that is currently supported in many other Eclipse views. Additionally, we noticed that JQuery with Multiple Views tends to use a great deal of screen real-estate. We believe a minimize option for views may help the developer free up some space for the code editor. We also felt a weakness in our design to be that we only allow the developer to desynchronize all the nodes at once. This feature does not appear to match well with the way developers use JQuery - making educated guesses via queries that often do not yield useful results. We believe that it would have been a better design to allow the developer a way to de-synchronize each node separately from the others.

9. FUTURE WORK

There are three areas of our JQuery with Multiple Views project that we believe need future work. Firstly, we believe it is necessary to conduct some more formal user studies on the tool. For our purposes, it would be most interesting to see if the developer utilized the multiple views feature to complete a task, as well as what types of browsers were created. To keep our study focused on the information visualization and not the query language, we believe the test subjects would need to have a working knowledge of JQuery and Java. As a secondary objective of the study, it would be interesting to note how many views the deveolopers keep open concurrently, and the size and positioning of the views.

Currently, JQuery with Multiple Views is still succeptible to the negative effects of Explosion of Browsers, including loss of context and disorientation while switching between views. To further combat this issue, we plan to draw further from principles of the Java Browser perspective. With this

perspective, when a selection in made in one browser, the other browsers are filtered to show only information directly related to that selection. For example, if a class is selected in the class browser, the method browser is updated to show only methods contained within that class. This helps to retain context for the user while browsing, since no information not directly related to that class is shown. Something similar to this feature could be built directly into JQuery with Mulitple views by allowing a view to base the information shown in its browser on a pointer to the currently selected node in any of the other views. With this new feature, a set of browsers could be created within JQuery with MV that match exactly with the Java Browser perspective. Additionally, the user could build these browsers to show information indirectly related to the selected node. For example: each time the user selects a class in the Active Query view, a browser in a separate view could reveal all subclasses of that class that override one of it's methods. Such information is extremely difficult to find using the browsers of the Java Browser perspective of eclipse and could be extremely useful to the developer in many routine maintenence tasks such as tracking down a bug.

And thirdly, although we have not discussed the issue of building queries in this paper, we believe a great deal of work must be put into finding a more suitable user interface for this vital part of JQuery. Currently, the novice user can use JQuery without writing queries because the tool comes prepackaged with many queries that the developers of JQuery thought would be useful. If the user wants to express new queries, however, it is necessary for him to open the source of JQuery and learn by example from the queries that currently exist. This often tedius and error prone process could possibly turn users away from JQuery with Multiple Views.

10. CONCLUSION

In this paper we have introduced JQuery with Multiple Views, a version of JQuery which we implemented to address the lack of contextual information present in a query path. Our tool allows the developer to create multiple views which present additional information to the developer on how a selected node fits into the system. We leveraged the logic language TyRuBa as a means of creating the additional views and also synchronizing the views.

We used color as a means for synchronizing the views. Our tool provides five colors *Blue, Green, Grey, Magenta* and *Yellow* which we felt provided enough contrast to one another to allow the developer to quickly view all matching nodes. Any node selected by the developer (by double-clicking) will take on one of these colors, and any node in the other views that matches it will automatically be revealed and highlighted by the same color.

As we discussed, we do not believe our tool is ready for practical use in development but we feel that it may serve as a starting point for experimenting with domain specific browsers. As a starting point, we will experiment in the distributed sytems domain, where functional and systemic code can be developed separately yet simultaneously. As for the immediate future, we believe it will be necessary to reimplement the synchronization mechanism to work by filtering uninteresting nodes rather than coloring the interesting ones.

11. REFERENCES

[1] Eclipse home page. http://eclipse.org/.

[2] D. Janzen and K. DeVolder. Navigating and querying code without getting lost. In *AOSD*, Boston, MA, USA, 2003. UBC, ACM.

[3] M. Kersten and G. C. Murphy. Mylar: a degree-of-interest model for ides. In *AOSD '05: Proceedings of the 4th international conference on Aspect-oriented software development*, pages 159–168, New York, NY, USA, 2005. ACM Press.

[4] G. E. Krasner and S. T. Pope. A cookbook for using the model view controller user interface paradigm in smalltalk-80. In *Journal of Object-Orientated Programming*, volume 1, pages 26–49, August/September 1988.

[5] E. McCormick and K. D. Volder. Jquery: finding your way through tangled code. In *OOPSLA '04: Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, pages 9–10, New York, NY, USA, 2004. ACM Press.

[6] M. P. Robillard and G. C. Murphy. Feat: a tool for locating, describing, and analyzing concerns in source code. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pages 822–823, Washington, DC, USA, 2003. IEEE Computer Society.

[7] TyRuBa home page. http://tyruba.sourceforge.net/.