# Satisfying Programmers' Information Needs in API-Based Programming

Chandan R. Rupakheti, Daqing Hou
*Department of Electrical and Computer Engineering*
*Clarkson University*
*Potsdam, NY 13699-5722*
{*rupakhcr, dhou*}*@clarkson.edu*

*Abstract*—**Programmers encounter many difficulties in using an API to solve a programming task. To cope with these difficulties, they browse the Internet for code samples, tutorials, and API documentation. In general, it is time-consuming to find relevant help from the plethora of information on the web. While programmers can use search-based tools to help locate code snippets or applications that may be relevant to the APIs they are using, they still face the significant challenge of understanding and assessing the quality of the search results. We propose to investigate a proactive help system that is integrated in a development environment to provide contextual suggestions to the programmers as the code is being read and edited in the editor.**

*Keywords*-**API-based programming; inter-procedural analysis; abstract interpretation; rule-based inferences**

## I. Introduction

Software libraries, frameworks, and Application Programming Interfaces (APIs) tend to become more complex as they mature over time. As more features are added and evolved in a framework, more rules are introduced to govern the interaction between its components. The users of the framework then have to know and follow these numerous rules in order to complete a programming task.

In general, a reuse-oriented system may not be used in the best possible ways that it is originally designed for. Instead, users often settle with a suboptimal set of available solutions that are just enough for their current tasks [3]. To address this problem, Fischer envisions an architectural design for a development environment that encompasses such tools as visualization, explanation, recommendation, and critics, to help programmers work with the framework and APIs [3].

The kind of environments that Fischer proposes are aimed at helping bridge the gap between the *situation model* and the *system model*. The situation model is a programmer's mental model about the task or problem needed to be solved, and it is often imprecise and informal. The system model, on the other hand, is about the actual running system. The system model is precise and formal, and can be expressed by such technical artifacts as source code that a machine can understand and execute, and design documentation.

In translating a problem in the situation model to a solution in the system model, a programmer may face several challenges. Given a problem, the programmer may not know whether a solution is possible and what process he needs to follow to create the solution. When there are multiple possible solutions, he may not know which one suits his situation the best. Finally, a programmer may make mistakes in executing a complex solution procedure. While search-based systems such as [1] [6] can bring valuable examples and documentation to the programmer, he may still have the difficulties in formulating the right queries in the first place [3], or in understanding and adapting the examples to his task at hand. Ideally, what is needed would be a human expert who can assess both the mental model and the current situation of the programmer and always give sound advices, but experts are always a scarce resource.

We propose to investigate tools and techniques to compensate for this scarcity of expertise. The overall goal for our tools would be to provide useful and relevant documents to help a programmer understand, grow, and criticize the API-related features used in his code. The relevant information would be proactively presented to the programmer while he reads and writes code in the editor. Specifically, there can be three ways for our tools to provide such help:

1) They *explain* the effect of the APIs to the programmer. While IDEs provide documentation for individual APIs, they do not explain the effect of combining multiple APIs;
2) They *recommend* APIs and solutions that might be needed next, based on their relevancy to those that are already used in the code;
3) They *criticize* the improper use of APIs and issues warning and error messages.

The technical foundation of our tools will be program analysis algorithms. More specifically, the tools would work by collecting facts at each relevant control point through inter-procedural program analysis and abstract interpretation. These facts would be used to infer the status of the code and to attach relevant documentation to specific locations in the code. The inferences will be based on the formal behavioral specifications for the relevant APIs. These documents, unlike the majority of those on the web, would be more directed towards the specific problems in the programmer's code.

To provide the three forms of help, somebody needs to supply the tools with the necessary information. There are two kinds of information sources: the original API designers and third-party, expert users of the APIs. Third-party developers, based on their experience, can identify and supply such information to the tools from diverse sources such as online forums, tutorials, books, design and API reference documentation, and output of exploratory empirical studies such as [9], [10], [8]. Another possibility is to integrate the output of other tools such as [21]. In this work, we will evaluate our tools mainly as third-party developers.

Our hypothesis is that such tools are *feasible*, *general*, and can improve both *software productivity* and *quality*, by making APIs easier to work with and less error-prone.

## II. A Motivating Example

User *JarifAlimov* has posted the following question on the Java Swing Forum [1] [2] on *March 14, 2004*, which has not been replied yet:

*Hi guys, I have problem's using JFrame in my program. I use the following code:* (shown in Listing 1)

*So, when I run the program, a Frame pops up with a Button at the appropriate location, but there is no JPanel with two RadioButtons. Well the most interesting thing yet follows, when I try to reshape the Frame by simply draging it with the mouse, the panel with RadioButtons appears on the Frame according to BorderLayout.NORTH, but the button suddenly gets to occupy the rest of the Frame. Can you please give an idea of what is going on? Thanks.*

Listing 1: JFrame does not work properly.

```
1  public static void main(String[] args){
2   JFrame win=new JFrame("Beat␣Kansas");
3   win.setDefaultCloseOperation(
4    JFrame.EXIT_ON_CLOSE);
5   win.setSize(400,400);
6   win.setVisible(true);
7
8   ButtonGroup group=new ButtonGroup();
9   JPanel panel=new JPanel();
10
11  JRadioButton j1=new JRadioButton("radio",
12   true);
13  group.add(j1);
14  panel.add(j1);
15  JRadioButton j2=new JRadioButton("my␣hell");
16  group.add(j2);
17  panel.add(j2);
18
19  JButton but1=new JButton("whateva");
20  but1.setSize(90,20);
21  but1.setLocation(23,45);
22
23  win.getContentPane().add(but1);
24  win.getContentPane().add(panel,
25   BorderLayout.NORTH);
26  win.repaint();
27 }
```

[1]http://forums.oracle.com/forums/forum.jspa?forumID=950. All URLs verified on Feb. 17, 2011.

[2]Although we've presented only one example, we will address the generalizability of our approach.

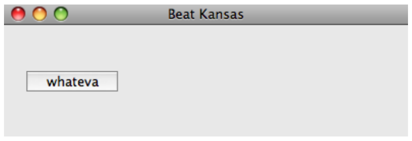

Figure 1: JFrame before being resized.

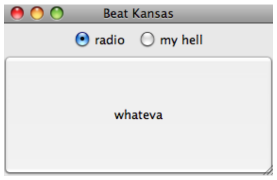

Figure 2: JFrame after being resized.

*JarifAlimov* had two problems with his program. The first is when the program executes, it creates a window, as shown in Figure 1, where only *but1* but not the other components are visible. The second problem occurs when the window is resized (Figure 2). Although all of the components are visible now, *JarifAlimov* is complaining that *but1* is occupying too much space in the middle of the window. Let's analyze the source code in Listing 1 to see how the tool could help *JarifAlimov* in three ways.

### A. *Explanation*

We envision that the tool analyzes the source code inter-procedurally, performing abstract interpretation based on the semantics of the API. Table I depicts a subset of the facts that holds after each line of code in Listing 1. In the text that follows, we illustrate how these facts play a critical role in explaining the behavior of the code, in particular, the two problems, for the user *JarifAlimov*. [3]

As shown in Table I, at line 6, *win* does not have any children in its content pane. So nothing but the frame itself is displayed. At line 26, the call to *win.repaint()* makes the *but1* visible because *but1* has *size* and *location* defined whereas *panel* and its subcomponents do not.

When the window is resized, the AWT/Swing framework triggers *JFrame* to re-compute its layout. In this case, the content pane of the frame uses *BorderLayout* as the default layout manager, which allocates all the extra space to the widget positioned in the center. The *win.getContentPane().add(but1)* method call at line 23 places *but1* in the center position of the window. These would explain why *but1* occupies the rest of the space of the frame, which confused *JarifAlimov*. In addition, the tool would also suggest that the size and location set at lines 20 and 21 will be overwritten by the layout manager, a fact that *JarifAlimov* would also like to know.

### B. *Critics*

A criticism will be produced if any of the following conditions becomes true:

[3]Note that Table I is used not only to provide explanations but also recommendations and criticisms.

Table I: Abstract interpretation of the code in Listing 1.

| Line # | Facts that hold after Line # |
|---|---|
| 2 | win.size = Unknown<br>win.visible = false<br>win.contentPane.children = <><br>BorderLayout : win.contentPane.layout<br>win.contentPane.layout.properties = <> |
| 4 | win.defaultCloseOperation = EXIT_ON_CLOSE |
| 6 | win.visible = true |
| 8 | group.members = <> |
| 9 | panel.children = <> |
| 21 | group.members = <j1, j2><br>panel.children = <j1, j2><br>j1.visible = false<br>j1.size = Unknown<br>j1.location = Unknown<br>j2.visible = false<br>j2.size = Unknown<br>j2.location = Unknown<br>but1.visible = false<br>but1.size = (90,20)<br>but1.location = (23,45) |
| 23 | win.contentPane.children = <but1><br>win.contentPane.layout.properties = <(but1, CENTER)> |
| 25 | win.contentPane.children = <but1, panel><br>win.cP.layout.props = <(but1,CENTER), (panel,NORTH)> |
| 26 | but1.visible = true |

1) In the GUI composition code, there exists a control point where a frame is visible but a descendant is not.
2) There exists a point where a frame repaints itself by calling *repaint()* and one of its descendants has an unknown size or location.
3) There exists a widget $w$ such that $w$ participates in a layout manager and the size or location of $w$ is set explicitly by calling *w.setSize()* and *w.setLocation()* in the client code.

The first problem is that initially the window does not display all of its components. This problem could have been detected by the critic rule 1. The facts accumulated on line 23 in Table I show that *win* is visible but *j1* and *j2* are not. Therefore, there is a violation of this rule and our tool would notify the user about this in some way. Furthermore, at line 26, *win.repaint()* is called but as shown in the table, sizes and locations of *j1* and *j2* are unknown. Therefore, our tool would also report a violation of rule 2.

At line 23, the button *but1* participates within *BorderLayout* of the content pane. In addition, it also has its size and location set by the two calls of *setSize()* and *setLocation()* at lines 20 and 21. Therefore, our tool would report a violation of rule 3.

### C. Recommendation

In general, our tool would offer two kinds of recommendation. The first kind is about information that is related to features or functionalities that have appeared in the code under analysis. For instance, recognizing the use of *JFrame* in the code, the tool would recommend other features that are available for a frame such as *win.setLocation()* method

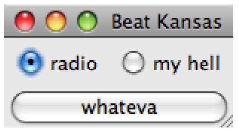

Figure 3: The final look of the *JFrame* after the tool's help.

to set the initial position of the frame. In this way, the user will conveniently find out what other relevant features are available beside the ones that he is using.

The second kind is related to the condition in the code and is triggered by the critic. For example, to fix the violation of rule 3 discussed in the previous section two alternative recommendations would be given. The first recommendation would be to use the so-called the *absolute positioning* strategy which can be achieved by calling *win.getContentPane().setLayout(null)*. The second would be to rely on a layout manager for sizing and positioning which can be done by deleting lines 5, 6, 20 and 21 and calling *win.pack()* followed by *win.setVisible(true)* at the end of the code. Our tool would present the instructions to do these and the rationales to the user. Figure 3 shows the look of the window when the user follows second recommendation.

### D. Discussion

*JarifAlimov* in our example could achieve the same result with different code as well. In general, there can be multiple solutions for a given programming task. The goal for our tool is not to find a *perfect* solution for a programming task; that is what the programmer has to do. It only aims to make the most important, API-related information easily accessible to the programmer by actively reacting to their intent as expressed through their code. By doing this, it would help reduce the information gap between the library providers and the application programmers.

## III. Planned Investigation

*Domain analysis.* As an initial set of use cases for evaluating our tools, we plan to develop support for two APIs: *JFrame* and *JTree*. We are formulating a set of API rules and relevant documents, concurrently with our manual investigation of the discussions in the Java Swing Forum.

*Program analysis algorithms.* We have built our own path-sensitive analysis framework [15] over Soot [20] to accomplish inter-procedural data flow analysis. We have used Eclipses' JDT before for type hierarchy and AST analysis [14]. We plan to use these works in this project.

*Representation of rules.* The rules can be represented in a tabular form in memory. We plan to investigate other alternatives such as SCL [7] and Prolog [4] for better representation of rules and inference engine. These alternatives would give the programmer a way to specify new rules without having to touch the core of the tools.

[4]http://www.gnu.org/software/gnuprologjava/

## IV. Evaluation Plan

After the completion of the first prototypes, user studies will be conducted to test the hypothesis on productivity and quality. A comparative study would be performed on a set of similar programming tasks, with and without the use of our tools. The *explanations*, *recommendations*, and *criticisms* provided by the tools in the process will be monitored and classified as being *useful*, *not useful*, or *irrelevant*. We will also evaluate the usability, accuracy, and the performance of the tools.

## V. Related Work

In our own previous works, we have studied the obstacles programmers may have about APIs [10], [9], [8]. Several rules from these works have been identified, which can be used as test cases for the proposed tools.

Our tools would use inter-procedural program analyses [18] [16]. We have used these techniques in our previous work on a path-sensitive static analysis tool and model checker called *EQ* [5] [14] [15].

Scaffidi discusses four kinds of challenges in learning and using APIs [17], which can be used as a conceptual guide for the kinds of challenges that our tools should address. Extensive past work has been done in searching [19] [5] [2] [1] [6], explaining and exploring examples [4] [12] [13], and understanding and debugging [11]. The main functionality of the search tools is to find code snippets and documents on the basis of key words. Our tool should complement them. By adding the elements of code pattern recognition and program understanding, our tools should be able to better predict the relevancy of a document to the code under analysis. But unlike theirs, we require the library providers and alike to write rules for API uses, which may be a burden. It has yet to be seen if well-known frameworks such as Java Swing have a manageable set of rules.

## VI. Conclusion

We explored the potential problems in software reuse that are caused by the information gap between the situation model and the system model. Such problems are often the consequence of a lack of understanding of the internal working of a framework and APIs on the part of programmers as well as other difficulties related to programmers' information needs. Although we presented only one example illustrating how our tool could help, we believe that such examples abound in practice. In fact, tens of thousands of discussions of similar problems were posted in the Swing Forum. If our tool could address a subset of these problems, it would be a valuable contribution towards bringing the much-needed help to the application programmers. We also believe that this problem is not unique to Java and Swing. A bigger research question is whether, and to what extent, such tools can be generalized to handle similar but unseen problems, and to other APIs and platforms.

[5] http://eqchecker.sourceforge.net/

## References

[1] J. Brandt, M. Dontcheva, M. Weskamp, and S. R. Klemmer, "Example-centric programming: integrating web search into the development environment," in *CHI*, 2010, pp. 513–522.

[2] B. Dagenais and H. Ossher, "Automatically locating framework extension examples," in *FSE*, 2008, pp. 203–213.

[3] G. Fischer, "Cognitive view of reuse and redesign," *IEEE Softw.*, vol. 4, pp. 60–72, July 1987.

[4] G. Fischer, S. Henninger, and D. Redmiles, "Cognitive tools for locating and comprehending software objects for reuse," in *ICSE*, 1991, pp. 318–328.

[5] M. Grechanik, C. Fu, Q. Xie, C. McMillan, D. Poshyvanyk, and C. Cumby, "A search engine for finding highly relevant applications," in *ICSE*, 2010, pp. 475–484.

[6] R. Hoffmann, J. Fogarty, and D. S. Weld, "Assieme: finding and leveraging implicit references in a web search interface for programmers," in *UIST*, 2007, pp. 13–22.

[7] D. Hou and H. J. Hoover, "Using SCL to Specify and Check Design Intent in Source Code," *IEEE Trans. Softw. Eng.*, vol. 32, no. 6, pp. 404–423, 2006.

[8] D. Hou and L. Li, "Obstacles in Using Frameworks and APIs: An Exploratory Study of Programmers' Newsgroup Discussions," in *ICPC*, 2011, 10 pp. to appear.

[9] D. Hou, C. Rupakheti, and H. Hoover, "Documenting and evaluating scattered concerns for framework usability: A case study," in *APSEC*, 2008, pp. 213 –220.

[10] D. Hou, K. Wong, and H. J. Hoover, "What can programmer questions tell us about frameworks?" in *IWPC*, 2005, pp. 87–96.

[11] A. J. Ko and B. A. Myers, "Debugging reinvented: asking and answering why and why not questions about program behavior," in *ICSE*, 2008, pp. 301–310.

[12] D. F. Redmiles, "Reducing the variability of programmers' performance through explained examples," in *CHI*, 1993, pp. 67–73.

[13] M. B. Rosson, J. M. Carroll, and C. Sweeney, "A view matcher for reusing smalltalk classes," in *CHI*, 1991, pp. 277–283.

[14] C. R. Rupakheti and D. Hou, "An Empirical Study of the Design and Implementation of Object Equality in Java," in *CASCON*, 2008, pp. 111–125.

[15] ——, "An Abstraction-Oriented, Path-Based Approach for Analyzing Object Equality in Java," in *WCRE*, 2010, pp. 205–214.

[16] B. G. Ryder, "Dimensions of precision in reference analysis of object-oriented programming languages," in *CC'03*, 2003, pp. 126–137.

[17] C. Scaffidi, "Why are apis difficult to learn and use?" *Crossroads*, vol. 12, pp. 4–4, August 2006.

[18] M. Sharir and A. Pnueli, "Two approaches to interprocedural data flow analysis," in *Program Flow Analysis: Theory and Applications*, S. Muchnick and N. Jones, Eds. Englewood Cliffs, NJ: Prentice-Hall, 1981, pp. 189–233.

[19] J. Stylos and B. A. Myers, "Mica: A web-search tool for finding api components and examples," in *VLHCC*, 2006, pp. 195–202.

[20] R. Vallée-Rai, P. Co, E. Gagnon, L. Hendren, P. Lam, and V. Sundaresan, "Soot - a Java Bytecode Optimization Framework," in *CASCON*, 1999, pp. 125–135.

[21] T. Xie and J. Pei, "Mapo: mining api usages from open source repositories," in *MSR*, 2006, pp. 54–57.