GraPacc: A Graph-Based Pattern-Oriented, Context-Sensitive Code Completion Tool

Anh Tuan Nguyen, Hoan Anh Nguyen, Tung Thanh Nguyen, and Tien N. Nguyen
Electrical and Computer Engineering Department
Iowa State University
{*anhnt,hoan,tung,tien*}*@iastate.edu*

***Abstract*—Code completion tool plays an important role in daily development activities. It helps developers by auto-completing tedious and detailed code during an editing session. However, existing code completion tools are limited to recommending only context-free code templates and a single method call of the variable under editing. We introduce GraPacc, an advanced, context-sensitive code completion tool that is based on frequent API usage patterns. It extracts the context-sensitive features from the code under editing, for example, the API elements on focus and the current editing point, and their relations to other code elements. It then ranks the relevant API usage patterns and auto-completes the current code with the proper elements according to the chosen pattern.**

***Keywords*-Pattern-oriented Code Completion; Usage Pattern**

I. Introduction

Code completion tools are useful for developers during their daily programming activities. Such a tool helps them to auto-complete detailed code during an editing session and enables them to focus more on their high-level algorithmic solutions. Therefore, it has become a crucial part in many modern integrated development environments (IDEs).

Aiming to improve developers' productivity in coding, a code completion tool should be able to predict developers' intention and to automatically fill in as much code as possible, while ensuring that the new code is consistent with the current code under editing. However, existing code completion tools [1], [2], [3] are limited to filling in at the level of *single method call* or *single object initialization*. For instance, as a user types a variable `v` and requires code completion, such a tool will produce a list of available methods for `v` by analyzing the type `T` of `v` and its associated methods. As (s)he progressively types more characters of the wanted method, the list of recommended methods will be shortened. As (s)he selects a method, the tool will auto-complete the method call for the currently edited variable `v`. Such support is valuable, however, the user must still go through a long list of too many options of recommended method calls.

To predict user's intention for better auto-completion, several approaches [2], [3], [4] have been proposed to rank the list of recommended method calls. However, the volume of auto-completed code is limited to a single method call for a variable. To address that, other tools [1], [5] provide code templates for common programming constructs, e.g. `for`/`while` loops with `Iterator` objects. Unfortunately, as a template is filled in, no context of currently edited code is considered.

Aiming to provide higher volume of code yet correctly considering the context of the code under editing, we develop **GraPacc**, a novel code completion tool that is based on the frequent usages of APIs. When working with a library, developers often repeat the correct ways of using APIs, which are called *API usage patterns*. Each pattern is a valid combination of the usages of variables, method calls, control structures and the orders among them. The usage patterns can be re-used in different tasks, therefore, it will be helpful if a code completion tool can take advantage of the patterns to auto-complete more code for developers.

Based on that principle, GraPacc first uses our prior tool, GrouMiner [6], to mine and to build a database of frequent API usage patterns from the source code of any given projects. It also allows users to define and to upload their own usage patterns. During an editing session, GraPacc extracts the context-sensitive features from the current code, i.e. the API elements on focus and the current editing cursor, and their relations to other code elements in order to rank the relevant API usage patterns and then to auto-complete the current code with the proper elements for the chosen pattern.

This paper presents the core architecture and novel features of GraPacc. Technical details can be found in [7]. The tool and video demo are available at our project Web site[1].

II. GraPacc: Pattern-Oriented Code Completion

GraPacc is built as an Eclipse's plugin. It has a view attached to Eclipse's workbench, from which users can invoke its functions. It also connects to the workbench's editor to get the current code, tokens' positions, and the cursor's location. It has a simple interface to the database for adding and querying API usage patterns. In general, GraPacc has two core sets of functionality including code completion support and API usage pattern importing and displaying support.

A. Code Completion Support

As a plugin, GraPacc operates within Eclipse's editor and can be invoked via Ctrl+Space shortcut. Figure 1

[1]http://home.engineering.iastate.edu/~anhnt/Research/GraPacc/

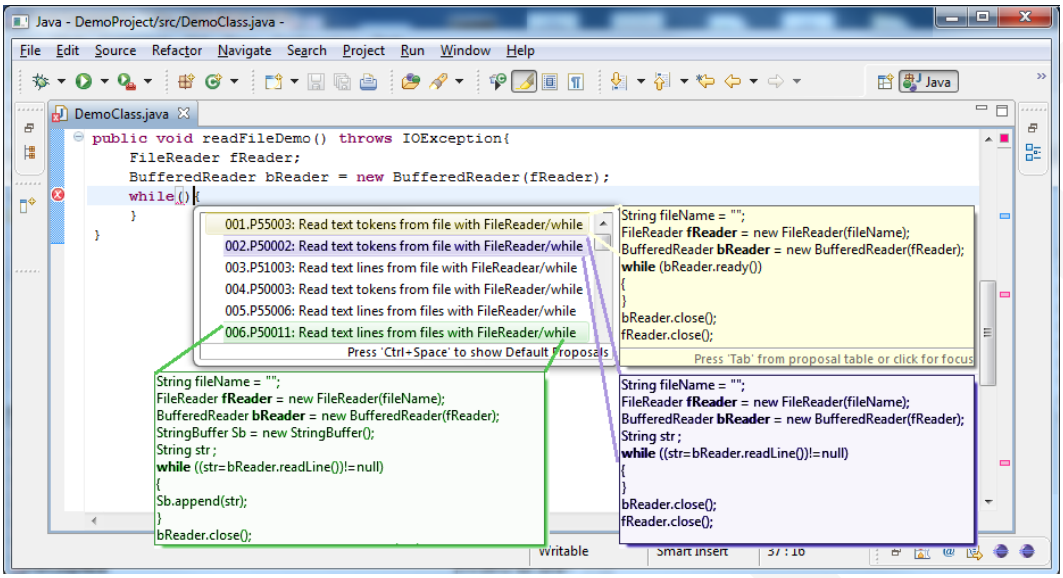

Figure 1. GraPacc: Graph-based Pattern-Oriented, Context-Sensitive Code Completion

shows a screenshot from GraPacc. A developer wants to read the content of a text file line-by-line with two Java classes, FileReader and BufferedReader, and a while loop. After typing while, (s)he invokes GraPacc for code completion. GraPacc then analyzes the currently edited code and extracts context-sensitive features including 1) the currently used data types (e.g. FileReader and BufferedReader), 2) the control and branching structures (e.g. while, if), and 3) the distances of those elements with respect to the current editing position. GraPacc uses those features to search in its database of API usage patterns to find and rank the usage patterns relevant to the current code. It allows the developer to browse the list of returned patterns and observe the code completion preview for each pattern. For example, as shown in the ordered list of patterns in Figure 1, the pattern 55003 is ranked first as it contains more similar features as those in the current code. With less similarity levels, other patterns are ranked lower in the recommended list (For the illustration purpose, we show the contents of three patterns. GraPacc shows only the content of the selected pattern).

As the developer selects a pattern P, GraPacc compares the current code with the pattern P and determines the tokens in P that need to be filled in. It then makes the proper transformations to the current code, rather than placing the entire selected pattern at the end of the current text in the editor as in the existing tools. For example, in Figure 1, as pattern 55003 is selected, GraPacc automatically completes the initialization of two variables fileName and fReader, and the condition of the while loop, while still maintaining the already-edited text in the initialization of variable bReader.

There are several key differences from GraPacc in comparison with existing code completion tools. First, it takes

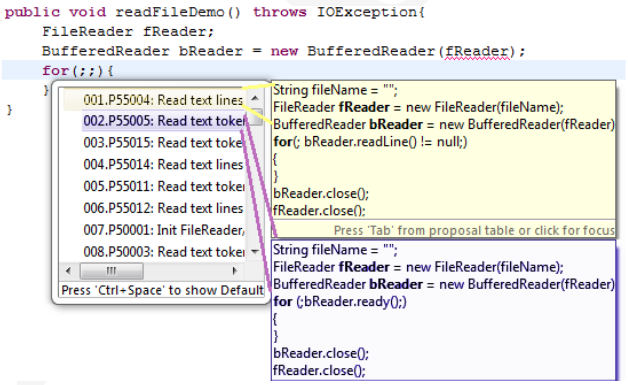

Figure 2. Ranking of Patterns based on Context-sensitive Features

into account several features from the *context* of the current code and is able to auto-complete *more code* with the entire selected usage pattern. In the existing tools [1], [2], [3], only the currently editing variable is considered and a *single method call* is auto-completed. Second, GraPacc considers *multiple variables* in *different types* in the current code (e.g. FileReader and BufferedReader), and a user can invoke it at *any point* within their code (e.g. within the while statement). In contrast, other tools require users to request code completion only on the selected variable. Third, GraPacc is context-sensitive in which as the user changes his/her editing cursor to a different program element or types more words in the current code, the tool will adjust the ranking of the returned usage patterns. In Figure 1, a user types the keyword while, thus, the patterns corresponding to the procedure of reading a text file via a while loop are ranked higher. Assume that (s)he instead wanted to read the file via a for loop and (s)he typed

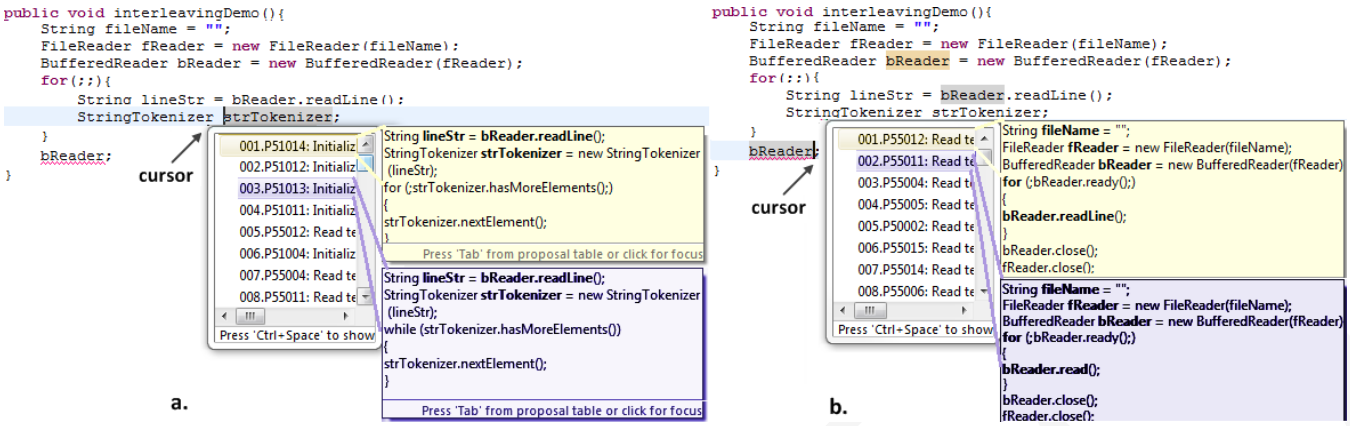

Figure 3. Switching between Interleaving Patterns

the keyword for. The patterns with FileReader, BufferedReader, and a for loop would be ranked higher (see Figure 2).

Another advanced feature is that GraPacc can recognize the user's switching between interleaving usage patterns and rank those patterns accordingly to his/her intention based on the editing cursor. For example, assume that in our current running example (Figure 3), the developer proceeds the task of reading a text file line-by-line via FileReader, BufferedReader, and a for loop. (S)he then continues with the task of tokenizing each line read from the text file via StringTokenizer inside the for loop. In this case, as (s)he invokes GraPacc and the cursor is at strTokenizer, the patterns relevant to StringTokenizer are ranked higher (see Figure 3a). However, if (s)he decides to switch his/her task to continue with closing the opened file as the cursor is at bReader (Figure 3b), GraPacc will rank the patterns of file reading higher and complete the code with bReader.close(); and fReader.close();. In other words, GraPacc is able to switch between two interleaving patterns: one for text file reading and one for string tokenizing depending on the editing cursor.

B. API Usage Patterns Importing and Visualizing Support

GraPacc also allows developers to import new patterns into its database. This function helps developers expand their own pattern databases. The code for a pattern is given by developers and is processed by GraPacc, which creates a graph-based representation, called Groum [6], and stores it into a database. Details on Groum can be found in [6].

Figure 4 shows the Groum of the current code in Figure 1 and that of the pattern 55003. The two Groums share four nodes and the dependencies between them. They reflect the high relevance between the current code and that pattern. The nodes will be matched and the unmatched nodes in the pattern will be considered for inserting into the current code.

III. GRAPACC ARCHITECTURE

Figure 5 shows GraPacc's architectural overview. It includes a pattern database manager, a processing module for

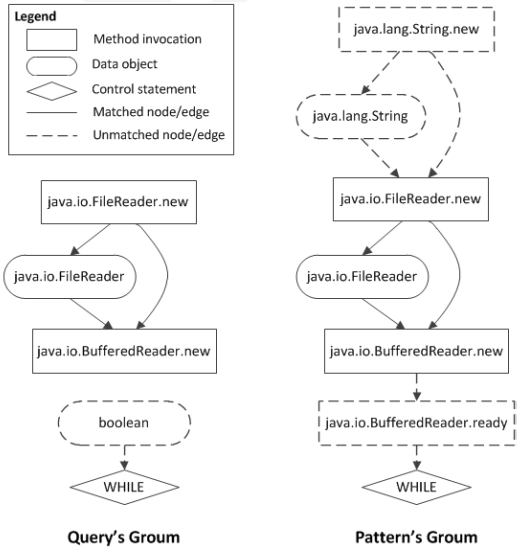

Figure 4. Graph-based Representation for the Query and Pattern

incomplete code under editing (called a *query*), a pattern searching and ranking module, and a code completion module.

The pattern database manager is responsible for importing/storing usage patterns. Each pattern, which is manually input or automatically mined via GrouMiner [6], is stored as a Groum with an associated code template. For efficient retrieval, GraPacc uses a weighted invert indexing technique to store each feature and the indexes of its containing patterns.

The query processing module parses the code under editing into tokens as well as into an Abstract Syntax Tree by using the Partial Program Analysis (PPA) tool [8].

The pattern ranking module searches the pattern database and recommends a ranked list of patterns relevant to the query. GraPacc analyzes the query's AST to build the Groum and extracts important features. It uses the query's features and un-parsable tokens to find the relevant patterns via the

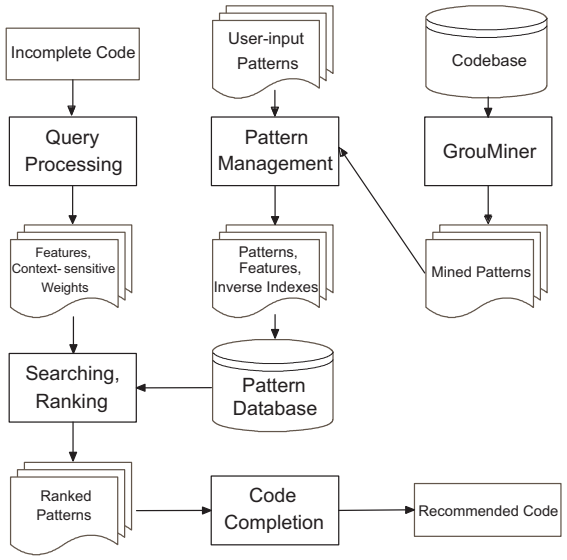

Figure 5. GraPacc's Architecture Overview

inverted indexes in the database. It then ranks those patterns based on the extracted context-sensitive features.

The code completion module analyzes the query and the selected pattern to correctly transform the variables, method invocations, and statements from that pattern to the source code of the query. The transformation to the pattern includes the alignment and the name conversions of variables, and the insertions of those missing tokens to the current code.

IV. Related Work

Bruch *et al.* [2] propose three algorithms to improve the ranking of the suggested method calls for a single variable v under editing based on the code examples in a code database. The first algorithm, FreqCCS, suggests the most frequently used method in the database. The second one, ArCCS, mines associate rules $m \rightarrow n$ in which if method m is used, method n is often called and it is suggested. In contrast, GraPacc suggests usage patterns (i.e. most frequently used graph-based API usages). The features in FreqCCS and ArCCS correspond to individual nodes (for method calls) and individual edges (for pairs of calls) in a Groum.

The third algorithm, BMN (best-matching neighbors), adapts k-nearest-neighbor algorithm to recommend for variable v. BMN encodes the current context and the examples in the database as binary feature-occurrence vectors [2]. The features for a context are the *un-ordered set* of method calls of v in the current code (i.e. a query) and the names of the methods that use v. The set of vectors of examples with the same smallest Hamming distance to the query vector is called the BMN set. Then, BMN ranks the method calls based on their frequencies in the examples in BMN set. In comparison, GraPacc captures richer contextual information of the code under editing, with all *ordered* method calls, multiple variables, and control structures in API usages, while BMN represents a context by an *un-ordered set* of method calls of a *single variable*. Importantly, GraPacc can handle *multiple variables* in *different types*. Finally, with API patterns, GraPacc recommends *more code elements*.

Hill and Rideout [4]'s code completion approach relies on code clones. GraPacc leverages code similarity at *API-usage* level. Robbes and Lanza [3] propose strategies to improve code completion using recent histories of modified/inserted code during an editing session. Their approach is limited to single method call. Eclipse [1] and other IDEs [5] complete for a method call of the variable under editing. Eclipse also supports template-based completion for common construct-s/APIs (for/while, Iterator) without considering the context.

MAPO [9] is a code example recommendation tool. It mines and indexes API usage patterns, and recommends associated code examples. It does not support auto-completion.

V. Conclusions

GraPacc is a code completion tool that takes context-sensitive features from the current code under editing and searches for relevant graph-based API usage patterns from a database. GraPacc can automatically and correctly complete more code in accordance with the chosen pattern.

Acknowledgment

This project is funded by US National Science Foundation (NSF) CCF-1018600 grant. It was also funded in part by a Vietnam Education Foundation grant for the first author.

References

[1] "Eclipse," www.eclipse.org.

[2] M. Bruch, M. Monperrus, and M. Mezini, "Learning from examples to improve code completion systems," in *ESEC/FSE'09*. ACM, 2009, pp. 213–222.

[3] R. Robbes and M. Lanza, "How program history can improve code completion," in *ASE'08*. IEEE CS, 2008, pp. 317–326.

[4] R. Hill and J. Rideout, "Automatic method completion," in *ASE'04*. IEEE CS, 2004, pp. 228–235.

[5] "Intellisense," http://blogs.msdn.com/b/vcblog/archive/tags/intellisense/.

[6] T. T. Nguyen, H. A. Nguyen, N. H. Pham, J. M. Al-Kofahi, and T. N. Nguyen, "Graph-based Mining of Multiple Object Usage Patterns," in *ESEC/FSE '09*. ACM Press, 2009, pp. 383-392.

[7] A. T. Nguyen, T. T. Nguyen, H. A. Nguyen, A. Tamrawi, H. V. Nguyen, J. Al-Kofahi, and T. N. Nguyen, "Graph-based pattern-oriented, context-sensitive source code completion," in ICSE'12, IEEE CS, 2012.

[8] B. Dagenais and L. Hendren, "Enabling static analysis for partial Java programs," in *OOPSLA'08*. ACM Press, 2008.

[9] H. Zhong, T. Xie, L. Zhang, J. Pei, and H. Mei, "MAPO: Mining and recommending API usage patterns," in *ECOOP 2009*. Springer-Verlag, 2009, pp. 318–343.