Using Transitive Changesets to Support Feature Location

Sukanya Ratanotayanon
Dept. of Computer Science
Thammasat University
Bangkok, Thailand
sratanot@cs.tu.ac.th

Hye Jung Choi
Dept. of Informatics
University of California, Irvine
Irvine, CA, USA
hchoi7@uci.edu

Susan Elliott Sim
Dept. of Informatics
University of California, Irvine
Irvine, CA, USA
sesim@uci.edu

ABSTRACT
In this paper, we present a new construct, called Transitive Changeset, that can be used for feature location. Transitive Changesets are created by extending changesets from revision control systems with additional information. A changeset temporally associate changes and conceptual descriptions provided in a commit transaction. By following transitive relationships from these changesets, we can create a Transitive Changeset that relates concepts in the problem domain to a list of program elements that enclose changes made in the transaction and other relevant program elements. We have implemented a prototype Eclipse plug-in, Kayley, to create Transitive Changesets.

Categories and Subject Descriptors
D.2.3 [**Software Engineering**]: Coding Tools and Techniques – *programmer* editors.

General Terms
Algorithms and Documentation.

Keywords
Feature location, source code search, changesets, commit transactions.

1. INTRODUCTION
Feature location deals with finding where a specific feature is implemented within software artifacts. It is an important problem, because in order to complete maintenance tasks a programmer needs to know where these features are located in the source code. Locating a feature can be challenging because relevant lines can be scattered through the code base and tangled with other features [2, 5]. A common approach is to search the source code [9] using keywords from the problem domain; programmers often know how to describe the features at a conceptual level, but not the implementation details. However, this approach has limitations, because there is a gap between the search keywords, which uses vocabulary from the problem domain, and the strings in source code, which uses vocabulary from the solution domain [1, 6, 8]. The conceptual keywords are usually not present in the code, and on the rare occasions when they are, the conceptual keywords may not appear in all the places where the feature is implemented. In this paper, we propose an approach to bridging the gap between programmers' domain knowledge and information in the source code text using a construct called Transitive Changeset.

ASE'10, September 20–24, 2010, Antwerp, Belgium.

Transitive Changesets are created from information that is recorded by revision control systems and other common software tools, such as issue trackers, and extend the available information using transitive relationships. Transitive Changesets contain conceptual-level information that is difficult to find in the source code and relates this information to a list of program elements in the code. Consequently, performing keyword search on the changesets could yield better results than straight code search. The relevant code sections can be located via the program elements included in the returned changesets.

Using information retrieval techniques, the Transitive Changesets are indexed to create a searchable repository with which programmers can locate features through searching changesets. We indexed the Transitive Changeset using their descriptions and names of included program elements. The conceptual descriptions included in the changesets allow programmers to search using keywords from the problem domain. Descriptions from bug reports or feature requests can be added to the changeset provide more descriptive data.

The searches return a list of changesets that match the search keyword. The user selects relevant changesets, which are analyzed further using program analysis techniques. The program elements included in the group of selected changesets are 1) validated in the current version of code; 2) expanded using a static dependency graph (SDG); and 3) ranked using relevance metrics. Metrics used to rank the results, include the TF-IDF score of the changeset containing the program elements and the depth of program elements in the static dependency graph. This analysis aims to increase the completeness of final results and to make a large set of results more manageable. We have built a prototype of this approach as an Eclipse plug-in, named Kayley.

The paper proceeds as follow: Section 2 reviews previous work in feature location, especially ones using information from revision control systems. Section 3 presents the definition of Transitive Changesets. In Section 4, we give an overview of our approach to feature location through Transitive Changesets. Future work and conclusions are given in Section 5.

2. BACKGROUND

Feature location is both an activity performed by programmers and a research problem. It is common for programmers to know what they need to do, e.g. add a new button to a dialog box, but not know where to make the change in the source code. The process that the programmer undertakes to figure what lines of code need to be changed is called feature location. In turn, algorithms for feature location has been a topic of research for many years [1]. Approaches include querying for a specific feature and mining for a many features at once. Our approach builds on the former.

When locating a feature in the implementation, programmers often can describe features only at the conceptual level, but do not yet know precisely where they are in the code. They commonly perform searches for a feature using keywords at the conceptual level, which do not always appear in the code. Commit transactions can be used to bridge this gap. Research has found version histories from a revision control system, such as CVS or Subversion, to be valuable sources of information, which can be used to locate code sections relevant to features. Comments provided in each commit transactions are helpful in identifying the purpose of the changes made. CVSSearch [3] uses the comments of commit transactions as metadata to label lines of code that are modified in the commit transactions. In addition to commit transactions, data from other sources can be used as metadata to improve the repository. Hipikat [4] showed that artifacts from issues trackers, such as bug reports and features requests, contain useful explanations of feature implementations. The commit transaction can also be linked with artifacts from these trackers.

A number of Information Retrieval (IR) techniques, which deal with the problem of retrieving information from large collections of unstructured data [5], have been applied to locate features using diverse software artifacts. Latent Semantic Indexing (LSI) [11, 12] and vector space model [1] compute the similarity between sections of documents. These have been used, for instance, to find traceability relations between requirements and source code. These IR techniques can be applied to information from the revision control system to improve search and ranking.

Static analysis has also been used to improve feature location. For example, SNIAFL [10] combines IR techniques with static analysis, and addresses the issue of low precision and recall by expanding and analyzing search results using call graphs. This technique addresses the problem of sections of code that are relevant, but not modified in a commit transaction; these code sections are not returned if only changesets are used in the repository. Static analysis can reveal relationships between program elements and locate these missing code sections.

3. TRANSITIVE CHANGESETS

A changeset is a concept commonly found in revision control systems. Tools such as Subversion have the concept of atomic commit, that is, a set of changes that are successfully committed together as a whole. Each time a developer performs a commit transaction, a changeset is created. We view a changeset as a logical container that includes information about the atomic changes and other metadata. The information commonly stored in a changeset consists of a list of changed artifacts, revision number, author of the changes, the time the changes are made and a comment.

3.1 Changesets

We can retrieve such information from commit logs as seen in Figure 1. The example below shows two commit transactions. The first one has a commit comment, but the second one does not. They list the files that were affected and the specific lines can be identified from the corresponding diff files.

```
$ svn log
------------------------------------
r3 | max | Wed, 1 Jul 2009 15:30
Change Paths:
M /trunk/code/Queue.java
M /trunk/code/Playlist.java
A /trunk/code/doc/README

Add a song to playlist

------------------------------------
r2 | Hye | Mon, 28 Jun 2009 14:00
Change Paths:
M /trunk/code/Treeview.java
M /trunk/code/Playlist.java
M /trunk/code/doc/TODO
```

Figure 1: Excerpt from a Transaction Log

There are three important characteristics of changesets that make them compatible with feature location. These are 1) multiple levels of abstraction; 2) a multi-modal vector; and 3) a temporal relationship. Each of these characteristics helps us bring to bear a wider variety of data to the problem of feature location.

Changesets span multiple levels of abstraction because they contain information at the domain level and the program level. Comments usually are short descriptions about the purpose of the change or task that was completed; these details are at the conceptual or problem domain level. The list of changes to files and lines contains information about the feature at the lexical or program level.

Changesets are multi-modal vectors because they collect up different kinds or modes of data. They bring together textual descriptions, path names that are pointers into the file system, file and line numbers that are pointers into the source code, and other metadata, such as the id of the committer.

Changesets makes a new kind of information available, specifically the time and date of the change. This tells us that files and lines that were changed at the same time are somehow related each other. This relationship can be a strong or weak one, but this information is not available in any other source.

3.2 Transitive Relations

Starting from a changeset, we can create a Transitive Changeset by expanding the stored information to pull in information from other levels of abstraction and associating the new information via transitive relations. The information that is incorporated inherits characteristics from the changeset.

To associate program elements to the changeset, we analyze the list of changed files. As a result of this analysis, we can identify the lines that were changed in each file. Given the line numbers

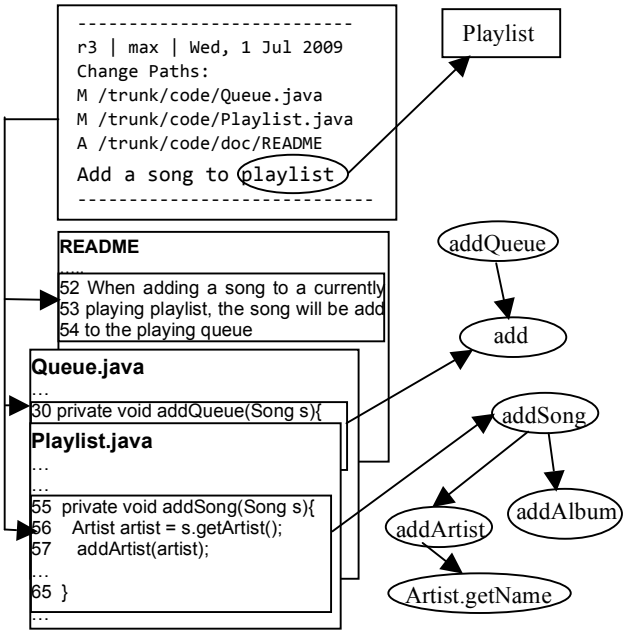

Figure 2: Transitive Changeset

that were changed, we can infer a list of program elements that included the changed lines by applying static analysis.

For example, in Figure 2, we found that some of the lines that were changed are line 55 to line 65 in the file Playlist.java. These lines correspond to the method 'addSong'.

These program elements affected by changes are temporally associated with each other, as they are changed together in a commit transaction. Therefore, we can infer that these program elements are related even though there is no explicit relationship in the code. In addition, they can be associated with the lines in non-code text files that are changed in the transaction and the concept provided in the comment due to the transitive rule.

There can be program elements related to the concept other than the ones inferred from affected lines, because some program elements may be related to the task, but were not modified. We can use other types of relationships between program elements to infer additional relevant program elements such as structural and dependency relationship. Continuing the example, we can use dependency to discover methods that are used by the 'addSong', namely 'addArtist', 'addAlbum' and 'Artist.getName()'. Using transitive relationship, we can associate these additional methods to the Transitive Changeset, which in turn associate them to the concept as well.

Transitive Changesets can be used to link information at different levels of abstraction, such as between problem and solution domains, or indirect relationships, e.g. between program elements and document sections. In addition, other artifacts can also be linked to the Transitive Changeset as well. For example, sometimes items from issue trackers for bug reports and feature requests are linked to specific commit transactions. We can include concepts discussed in the descriptions of these items in the Transitive Changesets. Also, in projects using agile method, test cases are implemented for specific user stories [7]. Therefore, we can establish a connection between the code being tested by the test cases and the concepts included in the user stories. Since many types of relationships can be used, the choice of transitive relation determines the information or levels that can be spanned.

4. USING TRANSITIVE CHANGESETS

Transitive Changesets contain conceptual-level information that is difficult to find in the source code and associate this information to related program elements. Therefore, they can

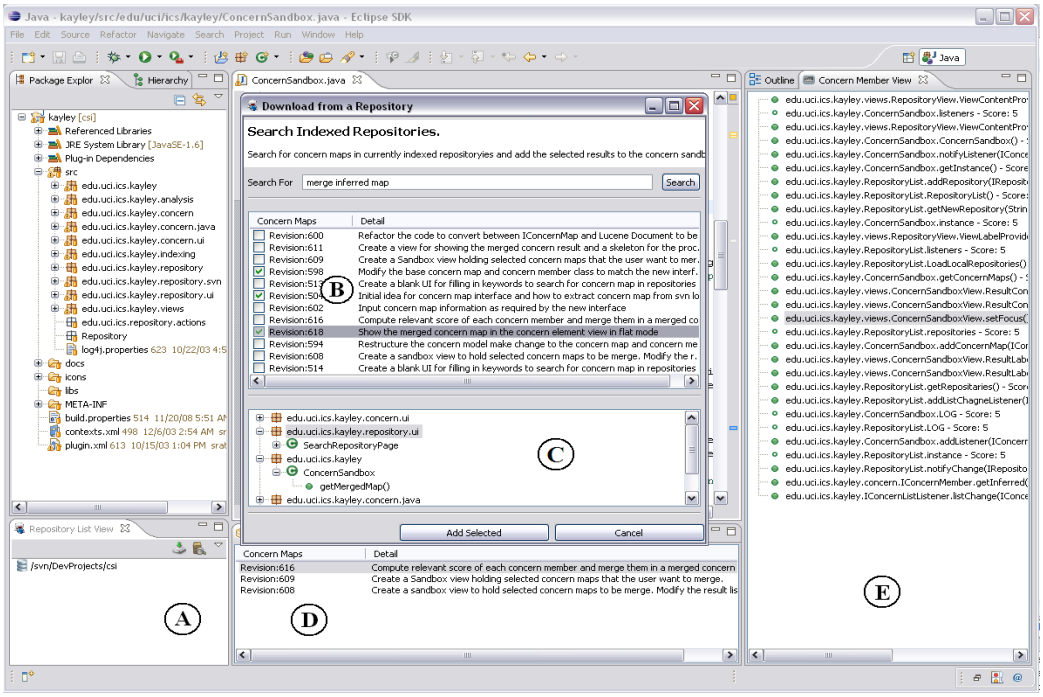

Figure 3: Search and Result Views of the Prototype

support feature location by bridging the gap between search keywords used by programmers and source code. Performing keyword search on the changesets can yield better results than searching directly on source code. The relevant code sections can be located via the program elements included in the returned changesets.

We implemented an Eclipse plug-in, Kayley, as a prototype for our approach; a screenshot is shown in Figure 3. Kayley is capable of creating a searchable repository of Transitive Changesets by importing a commit history from Subversion. To discover Java program elements enclosing lines that were added or modified, Kayley uses ASTParser provided in Eclipse to create an Abstract Syntax Tree (AST) of the file. Information from an issue tracker, if available, can be imported from an XML file and added to the Transitive Changesets. The different available repositories are in the view labeled 'A.'

Using the prototype, after issuing a query, s/he will be presented with a list of changesets matching the search keywords as seen in the area labeled *'B'*. The user needs to examine the list and select an initial set of relevant changesets. The program elements in the selected changeset are shown in the panel labeled '*C.*' An user can select the changesets that are considered related and add them to the ones that were previously selected (shown in a view labeled '*D*'.) Program elements included in all selected changesets will be processed and returned in the area labeled *'E'*.

Before returning program elements as final results, every program element included in the changesets are validated against the current version of the code and expanded using a static dependency graph to increase the accuracy and improve recall. Lastly, to reduce false positives, returned program elements will be ranked using relevance metrics. In this prototype, the metric we use are: the TF-IDF score of its enclosing changeset, the number of occurrences in the selected changesets, depth in SDG, size of the program element, and number of uses in the application.

5. CONCLUSION AND FUTURE WORK

This paper proposes the use of a construct called Transitive Changesets for improving feature location through searching with conceptual keywords. Transitive Changesets are created using information from commit transactions from a revision control system. These simple changesets associate commit comments with changed lines of code, thereby joining high level concepts with low level implementation details using a temporal relation. The information in the commit transactions are extended through transitive relations. Program elements related to the ones directly affected by the changes such as their callees can also be added to the Transitive Changeset to increase the completeness. We implemented our approach in Kayley, an Eclipse plug-in, which creates an index of the Transitive Changesets that programmers can use for feature location through search. Program elements included in the selected changesets are then extended, ranked, and returned as search results. Because Transitive Changesets join information at different levels of abstraction and includes information at the conceptual level that match the keywords provided by programmers, searching on the Transitive Changesets would provide better results than directly searching on the source code.

We propose to continue this work by refining the algorithm for ranking the changesets and program elements that are returned. In addition, we plan to conduct an empirical evaluation of Kayley, both in terms of its ability to locate features.

6. ACKNOWLEDGMENTS

Special thanks to Rosalva E. Gallardo-Valencia and Roy Tiburcio, for their valuable advice in implementation and preparing this manuscript.

7. REFERENCES

[1] T. J. Biggerstaff, B. G. Mitbander, and D. Webster, "The concept assignment problem in program understanding," in *Proceedings of the 15th International Conference on Software Engineering* Baltimore, Maryland, 1993, pp. 482-498.

[2] G. Canfora and L. Cerulo, "How crosscutting concerns evolve in JHotDraw," in *Proceedings of the 13th International Workshop on Software Technology and Engineering Practice*, 2005, pp. 65-73.

[3] A. Chen, E. Chou, J. Wong, A. Y. Yao, Z. Qing, Z. Shao, and A. Michail, "CVSSearch: searching through source code using CVS comments," in *Proceedings of IEEE International Conference on Software Maintenance*, 2001, pp. 364-373.

[4] D. Cubranic and G. C. Murphy, "Hipikat: recommending pertinent software development artifacts," in *Proceedings of the 25th International Conference on Software Engineering* Portland, Oregon, 2003, pp. 408-418.

[5] M. Eaddy, A. Aho, and G. C. Murphy, "Identifying, assigning, and quantifying crosscutting concerns," in *Proceedings of the First International Workshop on Assessment of Contemporary Modularization Techniques*, 2007, p. 2.

[6] G. Fischer, S. Henninger, and D. Redmiles, "Cognitive tools for locating and comprehending software objects for reuse," in *Proceedings of the 13th International Conference on Software Engineering* Austin, Texas, 1991, pp. 318-328.

[7] C. Mike, *User Stories Applied: For Agile Software Development*: Addison Wesley Longman Publishing Co., Inc., 2004.

[8] V. Rajlich and N. Wilde, "The role of concepts in program comprehension," in *Proceedings of the 10th International Workshop on Program Comprehension*, 2002, pp. 271-278.

[9] J. Singer, T. Lethbridge, N. Vinson, and N. Anquetil, "An examination of software engineering work practices," in *Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research* Toronto, Ontario, Canada, 1997, p. 21.

[10] W. Zhao, L. Zhang, Y. Liu, J. Sun, and F. Yang, "SNIAFL: Towards a static noninteractive approach to feature location," *ACM Transactions on Software Engineering Methodology,* vol. 15, pp. 195-226, 2006.