Spotter: Towards a Unified Search Interface in IDEs

(Preprint *)

Aliaksei Syrel[1], Andrei Chiş[1], Tudor Gîrba[3], Juraj Kubelka[2], Oscar Nierstrasz[1], and Stefan Reichhart[1]

Software Composition Group, University of Bern, `scg.unibe.ch`[1], PLEIAD Laboratory, University of Chile, `pleiad.cl`[2], `tudorgirba.com`[3]

Abstract

Program comprehension requires developers to reason about many kinds of highly interconnected software entities. Dealing with this reality prompts developers to continuously intertwine searching and navigation. Nevertheless, most integrated development environments (IDEs) address searching by means of many disconnected search tools, making it difficult for developers to reuse search results produced by one search tool as input for another search tool. This forces developers to spend considerable time manually linking disconnected search results. To address this issue we propose SPOTTER, a model for expressing and combining search tools in a unified way. The current implementation shows that SPOTTER can unify a wide range of search tools. More information about SPOTTER can be found at `scg.unibe.ch/research/moldablespotter`

Categories and Subject Descriptors D.2.6 [*Software engineering*]: Programming Environments – Integrated environments, Interactive environments

Keywords search, navigation, IDEs, integration

1. Research Problem and Motivation

Searching is a pervasive activity during program comprehension tasks [2, 6]. Consider *Question 41* from the set of developer questions identified by Sillito *et al.*: *How can we know that this object has been created and initialized correctly?* [5]. As it is not an easy question, developers often pursue it by asking several supporting questions, like: *(i)* what file contains the class of that object, *(ii)* what methods initialize that object, *(iii)* what classes use those methods, *(iv)* are there any known bugs that prevent that object from being initialized correctly, *(v)* are there any examples of how to initialize that object, *etc.* Addressing this, as well as similar questions, requires developers to intertwine searching with navigation and perform multiple exploratory searches that build upon each other on different data types [2, 4].

In spite of this, current mainstream IDEs address searching by means of multiple disconnected search tools, each applicable on specific types of data (*e.g.*, methods, classes, files, source code, objects). For example, answering the aforementioned question using the default Eclipse Java IDE would require a developer to use several search tools (*e.g.*, Find Type, Quick Search, Open Call Hierarchy) and, at each step, manually keep track of the current search results.

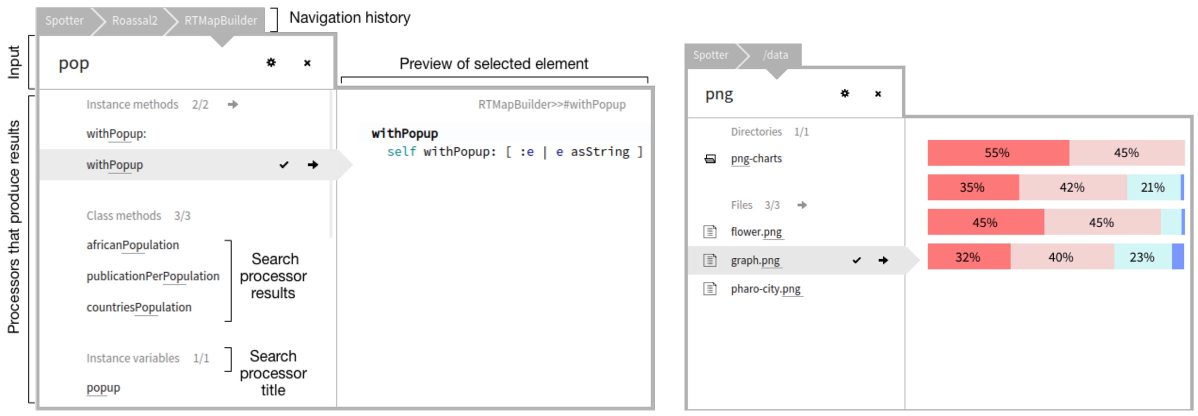
Approaches to searching followed by operating systems address part of the problem: they allow users to search over a wide range of different types of data (*e.g.*, files, applications, web pages, settings) all at once[1]. However, they are limited to one independent search. JQuery, a tool for exploring crosscutting concerns, enables developers to specify searches that build upon each other [1]. Nevertheless, JQuery requires a developer to explicitly indicate the dependency or the type of search she wants to explore. In the initial stages of an exploration task deciding what data to include in a search, or what dependencies are relevant, can be an issue.

To improve search support within IDEs the research question that we are investigating is:

> ***RQ:*** *How should an IDE support searches that build upon each other while allowing developers to rapidly identify relevant searches?*

* In Proceedings of the Companion Publication of the 2015 ACM SIGPLAN Conference on Systems, Programming, and Applications: Software for Humanity (SPLASH Companion 2015), October 25–30, 2015, Pittsburgh, PA, USA. `DOI: 10.1145/2814189.2817269`

[1] *Windows Search* – `http://bit.ly/1KJy87z`
Mac OS Spotlight – `http://bit.ly/1EgTIDN`



```
withPopup
  self withPopup: [ :e | e asString ]
```

(a) (b)

Figure 1: The user interface of SPOTTER: results from each search processor are displayed together; a breadcrumb indicates the navigation history between steps; new steps can be created on any result; a preview gives specific details about the selected result. (a) Search step opened on a class; the developer previously searched for the Roassal2 package, dived in that package and then dived in the RTMapBuilder class. (b) Search step opened on a file.

2. Spotter

To address our research question we start from the realization that having disconnected search tools within an IDE, each designed to work in isolation, is part of the problem. We propose a model, referred to as SPOTTER, for *expressing and combining search tools in a unified way*.

Expressing search tools An individual search tool, *i.e.*, *search processor*, consists of a *data source* (the collection of elements that are searched through, *e.g.*, the classes from a project), a *search filter* (the strategy for extracting a subset of elements from a data source, *e.g.*, searching for classes by name using a regular expression), and a *query processor* (a transformation applied on the raw query, *e.g.*, compiling a regular expression). Each search processor is associated with a target software entity (*e.g.*, method, bug report, object).

Combining search tools One search consists of a series of *search steps*. Each search step takes as input a software entity and executes all processors applicable on that type of entity. For example, opening a search step on a class loads all processors that search through data related to that class: methods, subclasses/superclasses, users, bug reports, documentation, instances — Figure 1a. When a user enters a query all search processors from the current search step are executed in parallel. Developers can spawn new search steps by selecting any search results from the current step.

To address the aforementioned question a developer can search for the object's class, create a search step on the class, and enter a query to trigger searches through all data relevant to that class. Once she found, for example, an interesting method she can spawn a new search step on that method and scope her next search to that method. This enables developers to uniformly compose searches on different data types, combine searching and navigation in one workflow, and automatically discover searches associated with an entity.

3. Current Results

To evaluate if SPOTTER can unify and improve searching within an IDE we first implemented it in Pharo (`pharo.org`), a modern Smalltalk dialect, as part of the GToolkit project (`gt.moosetechnology.org`). Due to positive feedback from software developers using our prototype, we have continuously improved it and integrated a stable version into the Pharo 4 IDE. Currently, SPOTTER has more than 100 search processors for 30 different data types, created both by us and by the developers of several libraries from the Pharo ecosystem. On average, extending SPOTTER with a new type of search requires 8 lines of code. This indicates that SPOTTER can unify a wide range of searches with a low effort.

We are further collecting anonymous usage statistics with the goal of understanding how developers use SPOTTER, what kind of queries they perform, and what kind of data they search for. Current data indicates that while most developers use the tool on a daily basis, they are unaware of several features provided by SPOTTER and do not use the navigation feature to its fullest potential [3]. Considering this observation, as well as extending the range of searches supported by SPOTTER, is our future work.

Acknowledgments

Andrei Chiş is supported by the Swiss National Science Foundation through the project "Agile Software Assessment" (SNSF project Nr. 200020-144126/1, Jan 1, 2013 - Dec. 30, 2015). Juraj Kubelka is supported by a Ph.D. scholarship from CONICYT, Chile. CONICYTPCHA/Doctorado Nacional/2013-63130188

References

[1] D. Janzen and K. de Volder. Navigating and querying code without getting lost. In *AOSD*, pages 178–187, 2003.

[2] A. Ko, B. Myers, M. Coblenz, and H. Aung. An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. *IEEE Trans. Softw. Eng.*, 32, Dec. 2006.

[3] J. Kubelka, A. Bergel, A. Chiş, T. Gîrba, S. Reichhart, R. Robbes, and A. Syrel. On understanding how developers use the Spotter search tool. In *VISSOFT–NIER*. to appear, IEEE, 2015.

[4] M. P. Robillard, W. Coelho, and G. C. Murphy. How effective developers investigate source code: An exploratory study. *IEEE Trans. Softw. Eng.*, 30, Dec. 2004.

[5] J. Sillito, G. C. Murphy, and K. De Volder. Asking and answering questions during a programming change task. *IEEE Trans. Softw. Eng.*, 34:434–451, July 2008.

[6] J. Starke, C. Luce, and J. Sillito. Searching and skimming: An exploratory study. In *ICSM*, pages 157–166, Sept. 2009.