

Mismar: a new approach to developer documentation

Barthélemy Dagenais
McGill University
bart@cs.mcgill.ca

Harold Ossher
IBM T.J. Watson Research Center
ossher@us.ibm.com

Abstract

Successful open source projects foster collaboration and innovation while benefiting from a faster pace of development, but are often plagued by poor developer's documentation. In this paper, we present the rationale and the architecture of Mismar, a toolset tightly integrated in the Eclipse environment and implementing a concern-oriented approach to documentation. As opposed to traditional documentation artifacts, guides produced by Mismar are almost wordless and automatically maintain implementation examples. Moreover, since Mismar was built from the ground up to be extensible, it is easy to support artifacts written in multiple languages or modeling approaches.

1. Introduction

Successful open source projects foster collaboration and innovation while benefiting from a faster pace of development resulting from the great variety of contributions. Unfortunately, most projects, even mature ones such as Eclipse [3], cannot fully leverage the power of the open source community for various reasons, one of them being the lack of proper developer's documentation. Contributors often have to reverse engineer a complete application to understand it before beginning to actively collaborate. While some developers are more talented in this area than others, all suffer from this lack of documentation.

2. A concern oriented approach

In [2], we proposed a toolset, tightly integrated within the Eclipse development environment [3], to leverage software developers' knowledge by simply asking them to point out important elements in the software system relevant to a task, such as classes, methods, files or even web pages. From these elements, a guide with an appropriate step for each type of element is created. For example, if the user chooses the

XYZ class, an "Extend XYZ class" step is created. Hence, a developer can create a complete guide by simply dragging and dropping (or copying and pasting) elements from a system to the guide editor. Once a step is created, comments and references to other artifacts can be added to provide contextual information. We referred to this approach as *concern oriented* since it focuses on software artifacts and their relationships instead of the steps or process. Indeed, concerns [10] are usually said to capture every element in a software system that is relevant to a particular point of interest.

The resulting guide is also tightly integrated into the development environment, providing interactive support to the user following it. For example, when the user performs the "Extend XYZ class" step by double-clicking on it, the "Create new class" wizard in Eclipse is launched. The wizard is customized by the information provided by the guide author: the dialog title and description fields are taken from the step title and description fields and the wizard automatically proposes to extend the XYZ class. This is what we called *active concerns*, in contrast to traditionally passive concerns, since new behaviors are inferred from the elements of a concern.

3. User experience

Programmer's guides, tutorials or cheat sheets [1] usually contain steps to be performed, described with complete sentences. If the guide is complex, the process of creating or following it can be tedious, simply because there is a lot of text to write or read. Such guides may include pictures, such as screenshot or diagrams, but they are mainly text based. It is then hard for the reader to quickly capture important concepts and artifacts and to understand their relationships.

We argue that guides created by Mismar are worth a thousand words and embody a lot of knowledge carried by the user interface elements instead of by words. When creating a guide, the user typically does not have to enter a single word. Steps' titles and references' labels are inferred from the elements selected by the user:

we go so far as to parse the clipboard to retrieve the title of a pasted web link. As another example, if the user selects the ABC interface, the guide will create an “Implement ABC interface” step. Furthermore, icons, taken from the Eclipse environment, are used to represent the different types of steps and artifacts. The user, familiar with these icons, is then able to quickly assess the types of steps and elements involved. The order of the steps or references also carries important information, e.g., dependence, priority and relevance. Guides can be used for exploring, without or before execution.

Relationships between artifacts and steps are indicated by their proximity. When the user selects a step, the view is refreshed to show the step references and output examples. We prevent information overload [6] by showing only relevant information for a task and prevent user interface cluttering by using proximity instead of words or arrows to represent relationships.

Code examples have often been recognized as one of the most useful sources of information when trying to understand a program [5][7]. Our guide leverages the power of examples by recording every output resulting from a step execution. These outputs are then proposed as implementation examples when a user performs the same step in another context, i.e., when following the same guide to create another result.

4. Architectural overview

The model behind this toolset was inspired by prior work aimed at defining an extensible and customizable model for concerns [4][9]. Like more recent tools such as ConcernMapper [8] and Mylar [6], our toolset hides the concrete model from the end-user to some extent, but makes it available for third-party tools. The end user is thus not aware that by creating and using a guide, s/he is building a complete concern model that can then be analyzed and transformed by other tools.

Extensibility was the top priority during the design of the toolset. Indeed, we initially wanted to support any type of reference (i.e., dragging and dropping any type of element, even from third party tools or plugins), but this capability soon extended to the possibility of customizing the interactive support and being able to create new types of concerns, intensions and steps.

Extensibility is provided through the extension point mechanism in Eclipse [3]. Our toolset mainly provides two extension points named *type* and *extender*. To support a new type of element, the developer must create a *type* extension specifying a unique identifier, an icon, and the kind of element (e.g., a new type of concern or a new type of reference). Then, the developer must specify *extenders*, which are Java classes provid-

ing element-type-specific implementations of interfaces that are called by the toolset. Both core and UI-related extenders are typically needed.

5. Conclusion

With Mismar, software developers get the chance to easily and rapidly document their systems, which could encourage collaboration and contribution. In this paper, we presented the rationale and the extensibility mechanisms behind the toolset. Using a concern-oriented approach led to interesting features, including wordless guides and associated implementation examples. We are hoping to release Mismar with an academia-friendly license in the near future.

6. References

- [1] “Building cheat sheets in Eclipse V3.2.” <http://www-128.ibm.com/developerworks/library/os-ecl-cheatsheets/>
- [2] Dagenais, B., Ossher H. “Guidance Through Active Concerns,” To appear in *Proceedings of the Eclipse Technology Exchange* at OOPSLA 2006.
- [3] Eclipse. <http://www.eclipse.org/>
- [4] Harrison W., Ossher H., Sutton S., Tarr P., “Concern modeling in the concern manipulation environment.” In *Proceedings of the Workshop on Modeling and Analysis of Concerns in Software*, 2005.
- [5] Holmes, R and Murphy G. C., “Using structural context to recommend source code examples.” In *Proceedings of the International Conference on Software Engineering*, pages 117–125. ACM Press, 2005.
- [6] Kersten, M. and Murphy G. C. “Mylar: a degree-of-interest model for IDEs.” In *Proceedings of the 4th Conference on Aspect-Oriented Software Development*, pages 159-168, 2005.
- [7] Nykaza, J., Messinger R., Boehme, F. Norman C. L., Mace M., Gordon M. “What programmers really want: results of a needs assessment for SDK documentation.” In *Proceedings of the 20th annual international conference on Computer Documentation*. Pages 133-141, 2002.
- [8] Robillard, M. and Weigand-Warr, F. “ConcernMapper: Simple View-Based Separation of Scattered Concerns.” In *Proceedings of the Eclipse Technology Exchange* at OOPSLA, 2005.
- [9] Robillard, M. P. *Representing Concerns in Source Code*. Ph.D. Thesis. Department of Computer Science, University of British Columbia. November 2003
- [10] Tarr, P., Ossher, H., Harrison, W. and Sutton, Jr., S. M., “N degrees of separation: Multi-dimensional separation of concerns.” In *Proceedings of the 21st International Conference on Software Engineering (ICSE '99)*, 107–119, IEEE, May 1999.