

# Just-in-Time Concern Modeling

Martin P. Robillard  
School of Computer Science  
McGill University  
Montréal, QC, Canada  
martin@cs.mcgill.ca

Gail C. Murphy  
Department of Computer Science  
University of British Columbia  
Vancouver, BC, Canada  
murphy@cs.ubc.ca

## ABSTRACT

In this position paper, we propose the notion of just-in-time concern modeling. As some concerns emerge late in the software life cycle and can be ephemeral, we argue that mechanisms should be available to capture descriptions of concerns as they emerge or become relevant. Based on our experience with the FEAT concern modeling and analysis tool, we highlight the essential characteristics, benefits, and pitfalls of just-in-time concern modeling at the source code level.

## Categories and Subject Descriptors

D.2.3 [Software Engineering]: Coding Tools and Techniques; D.2.6 [Software Engineering]: Programming Environments—*Interactive environments*; D.2.7 [Software Engineering]: Distribution, Maintenance, and Enhancement

## General Terms

Documentation, Human Factors, Experimentation

## Keywords

Separation of concerns, concern modeling, inconsistency management, program investigation

## 1. INTRODUCTION

The general notion of a *concern* has proven a useful tool to organize software development. Simply stated, a concern is anything a stakeholder may want to consider as a conceptual unit, including features, non-functional requirements, and design idioms.

As studies and experience have shown, the realization of a concern in a software system is often scattered and tangled throughout the different artifacts comprising the system, and in particular through source code [1, 3, 10]. Among the many reasons explaining the scattering and tangling of concerns, we find that new concerns often *emerge* as a system evolves. Concerns can emerge because of changes to an application's domain (e.g., new requirements [11]), or because maintenance tasks require that new concepts be understood in isolation. In any case, the unplanned nature

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

*Workshop on Modeling and Analysis of Concerns in Software (MACS 2005)*  
16 May 2005, St. Louis, MO, USA  
Copyright 2005 ACM 1-59593-119-8/05/05 ...\$5.00.

of an emerging concern implies that its corresponding realization will generally cut across multiple modules, making any change associated with the concern more difficult and more risky.

Mitigating the impact of emerging concerns is difficult for two main reasons: (a) the number of potentially emerging concerns is infinite, and (b) the continued relevance of emerging concerns is difficult to predict. Combined, these reasons make it not cost-effective to explicitly re-modularize systems to account for *all* emerging concerns (e.g., through refactoring [2, 5] or Aspect-Oriented Programming [3]).

To address the problem of emerging concerns, we have been researching an approach that allows developers to model the implementation of concerns *as they become relevant*. This approach relies on a model called concern graphs [8, 9] which represents the subset of the source code of a system relevant to a concern. The concern graph model is supported by a tool, called FEAT,<sup>1</sup> that allows developers to build models of scattered concerns while they are investigating the source code, and to make changes to a system through a simplified view of the system provided by the concern models.

This paper briefly describes how to model concerns using FEAT and concern graphs, highlights the characteristics of the approach that support just-in-time concern modeling, and discusses the benefits and tradeoffs of just-in-time modeling at the source code level.

## 2. MODELING CONCERNS WITH FEAT

The Feature Exploration and Analysis Tool (FEAT) is an Eclipse plug-in allowing developers to create models describing the implementation of concerns in Java source code, and to modify the code of a system through a simplified view provided by the models. Eclipse is an integrated development environment for Java with an architecture that supports the addition of modules (called plug-ins) that add to the development environment's functionality [4]. Since the complete description of FEAT is available from other sources [6, 9], we only present a brief scenario of a developer modifying source code with FEAT.

First, the developer investigates the source code of a system using a set of queries provided by the FEAT tool (Figure 1). FEAT queries are similar to the functionality offered by the Eclipse Search Engine but offer a greater number of query types and can be used to model concerns. Query results are presented in a view listing the domain and range of the query (Figure 2). Query results can be further analyzed.

When a concern of interest emerges in the code explored by the developer, the developer creates a concern model. At this point a concern model is simply a hierarchical organization of concerns

<sup>1</sup><http://www.cs.ubc.ca/labs/spl/projects/feat>

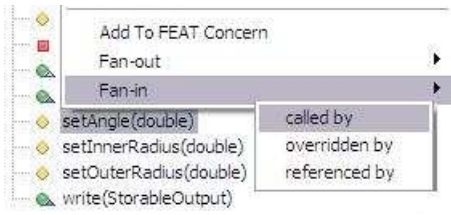


Figure 1: A FEAT query accessed from the Package Explorer

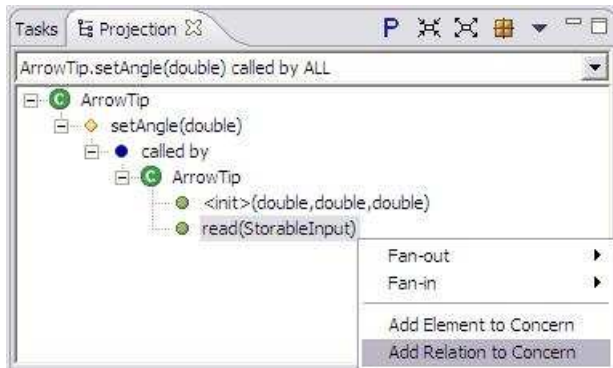


Figure 2: Query results in FEAT

and their sub-concerns. The left part of Figure 3 shows an example. In this model, a top-level concern represents the “Save” feature often found in GUI-based applications. This top-level concern is divided into three sub-concerns that represent the code that implements different subsets of the main concern.

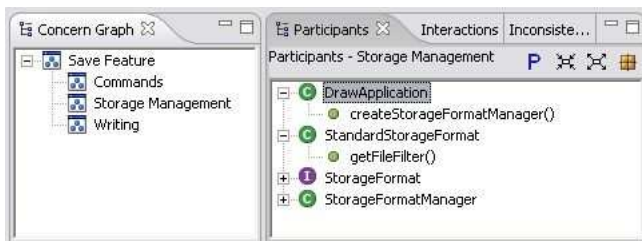


Figure 3: A Partial Concern View in FEAT

During code exploration, the developer builds a concern model by adding individual elements (methods, fields, or classes) that were discovered during the exploration to the concern model. The entire results of queries can also be added to the concern model as one “building block”. At any point, the current concern model can be viewed by selecting a box in the Concern Graph View (left panel of Figure 3). Selecting a concern model displays all of the elements in the concern (see Figure 3, right panel), and their relations (displayed in a separate view not shown in this paper). From these views, it is possible to drill down to the source code and perform modifications.

### 3. ESSENTIAL CHARACTERISTICS

FEAT has a number of characteristics that support just-in-time concern modeling at the source code level.

- **Non-intrusive.** Concern models produced by FEAT are purely descriptive. They consist of a graph of relations (e.g., “calls”, “accessed by”) between elements in a program [8]. As a result, using FEAT does not require any preprocessing of the code besides the static analysis required to support the queries. As a result, a developer can begin modeling concerns with FEAT at any point in a task. The developer can also stop modeling concerns at any point and use partial results.
- **Integration with program investigation.** Besides the trivial step of naming concern models, the only task necessary to model concerns in FEAT is to flag query results as part of a concern. This task can be integrated in the normal program investigation activities of developers, and does not require any context switching or even the use of graphical views that are different than the ones used for program investigation.
- **Inconsistency detection.** Concern models that describe parts of other software engineering artifacts can become inconsistent as the artifacts are modified. As a simple example, a list of methods of interest in a software system will become inconsistent if a method is deleted or if any of its uniquely identifying characteristic is changed (e.g., identifier, signature). To ensure that concern models remain valid, FEAT incorporates an inconsistency detection and management mechanism that ensures that any concern description available to a developer is either consistent or explicitly flagged as inconsistent [9]. As a result, developers can alternate the tasks of modeling concerns and modifying the corresponding source code without fear of invalidating the models produced.

### 4. DISCUSSION

The concern graph approach as implemented in FEAT was evaluated formally through five empirical studies [6, 8, 9] and informally through discussions with numerous users. These experiences enable us to present a synthesis of the main benefits and pitfalls of the idea of just-in-time concern modeling at the source code level.

#### 4.1 Benefits

*Just-in-time concern modeling naturally supports program investigation activities.* As part of an empirical study of program modification in Eclipse (not involving FEAT), we observed many characteristics of developer behavior that can benefit from the support of concern models [7]. For example, we found that developers who successfully implemented a requested change tended to document their planned code modifications. Just-time concern models can be used to help developers build this plan. We also observed that all developers periodically re-investigated the same methods during a program modification task. Just-in-time concern models can help developer keep track of their current knowledge of a concern, potentially limiting the amount of re-investigation necessary. As part of another empirical study, this time with subjects using FEAT [6], we observed developers referring to methods stored in a concern graphs on multiple occasions during a modification task. For example a developer would first access an element in the concern model to implement a feature, and then use the concern model to quickly find the same element at a later stage to fix an incorrect

implementation of the feature. Such evidence indicates that concern models created on-the-fly during software modification tasks have immediate benefits for developers involved in the task.

*The relative cost of just-in-time concern modeling is minimal when performed as part of program investigation activities.*

Controlled studies and informal user feedback have shown that investigating source code by following structural dependencies is a natural process for many developers. In fact many users of FEAT report using the tool mostly for its powerful cross-referencing features. Because the basic idea of just-in-time concern modeling with FEAT is to piggy-back the creation of concern models onto this natural program investigation task, the true cost of building concern models is minimal. This low cost is an important property of just-in-time modeling as it removes the necessity for the (sometimes implicit) analysis required to ensure that the benefits of using concern models will surpass the cost of their creation. Because the cost of just-in-time concern modeling is so low, it is practical to model concerns in this fashion for all concerns but the ones having the simplest implementation. In practice, we have found that it becomes useful to build concern models with FEAT as soon as the implementation of a concern involves code in more than three or four different methods.

## 4.2 Pitfalls

*The context associated with just-in-time concern model is implicit.* The current support for modeling concerns in FEAT is limited to descriptions of the source code. The only additional information in a concern model is a free-form concern name provided by a developer (e.g., “caching mechanism”, “save feature”). By capturing limited information about the context in which a concern model is relevant, just-in-time concern models are at risk of being misinterpreted. For this reason, care must be taken in choosing descriptive names for concern models.

*The nature of just-in-time concern models is very task- and developer-specific.* Concern models created on-the-fly tend to be very specific because they only include information about code structures traversed during program investigation activities for a specific task. For this reason, they may be incomplete. For example, a concern model for a feature might only describe half of the code relevant to the feature if the developer who created the model only needed to explore half of the feature’s implementation to carry out the task. As a result, developers must be careful about any assumptions they make about the completeness of just-in-time concern models when attempting to reuse these models.

## 5. CONCLUSION

Not all concerns can be foreseen and a number will inevitably emerge during the evolution of a software system. Emerging concerns whose implementation is scattered throughout a system are particularly problematic as it may not be cost-effective to re-structure the code of the system to modularize them. We propose to model emerging concerns using just-in-time techniques. Our FEAT concern modeling and analysis tool supports just-in-time concern modeling by integrating the concern modeling and program investigation activities. Experience and empirical studies involving FEAT have shown that just-in-time concern modeling can provide direct benefits to developers in the context of a single task and, given careful management, can result in concern models that are useful for multiple tasks.

## 6. REFERENCES

- [1] Stephen G. Eick, Todd L. Graves, Alan F. Karr, J.S. Marron, and Audis Mockus. Does code decay? assessing the evidence from change management data. *IEEE Transactions on Software Engineering*, 27(1):1–12, 2001.
- [2] Martin Fowler. *Refactoring—Improving the Design of Existing Code*. Object Technologies Series. Addison-Wesley, Boston, MA, USA, 2000. With contributions by Kent Beck, John Brant, William Opdyke, and Don Roberts.
- [3] Gregor Kiczales, John Lamping, Anurag Menhdhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In *Proceedings of the 11th European Conference on Object-oriented Programming*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242. Springer-Verlag, June 1997.
- [4] Object Technology International, Inc. Eclipse platform technical overview. White Paper, July 2001.
- [5] William F. Opdyke. *Refactoring Object-Oriented Frameworks*. PhD thesis, University of Illinois, Urbana-Champaign, IL, USA, 1992.
- [6] Martin P. Robillard. *Representing Concerns in Source Code*. PhD thesis, Department of Computer Science, University of British Columbia, Canada, November 2003.
- [7] Martin P. Robillard, Wesley Coelho, and Gail C. Murphy. How effective developers investigate source code: An exploratory study. *IEEE Transactions on Software Engineering*, 30(12):889–903, December 2004.
- [8] Martin P. Robillard and Gail C. Murphy. Concern Graphs: Finding and describing concerns using structural program dependencies. In *Proceedings of the 24th International Conference on Software Engineering*, pages 406–416. ACM Press, May 2002.
- [9] Martin P. Robillard and Gail C. Murphy. Evolving descriptions of scattered concerns. Technical Report SOCS-TR-2005.1, School of Computer Science, McGill University, Montréal, QC, Canada, January 2005.
- [10] Peri Tarr, Harold Ossher, William Harrison, and Stanley M. Sutton. N degrees of separation: Multi-dimensional separation of concerns. In *Proceedings of the 21st International Conference on Software Engineering*, pages 107–119. IEEE Computer Society Press, Los Alamitos, CA, USA, May 1999.
- [11] Jilles van Gurb and Jan Bosch. Design erosion: problems and causes. *Journal of Systems and Software*, 61:105–119, 2001.