Concern Management and Evolution at The Architectural Level

[Doctoral Symposium Submission]

Eugen Nistor
Advisor: André van der Hoek
Donald Bren School of Computer Sciences
University of California, Irvine
Irvine, CA, 92697
enistor@ics.uci.edu

ABSTRACT

We provide an approach for identification and visualization of concerns at the code and architectural level, and maintenance of traceability of these concerns throughout software evolution.

Categories and Subject Descriptors

H.4 [**Software engineering**]: Concern-based development, software architecture, traceability, consistency.

1. INTRODUCTION

Separation of concerns is a leading principle in software engineering, used to address the inherent complexity of software. There is no precise definition of what a concern is but, in general, a concern can denote anything of importance. During development, the languages that we choose to express our software into impose their own modularization, and all different concerns that are addressed will inherently crosscut this modularization. Concerns will naturally become scattered throughout implementation and tangled together with other concerns [20].

A number of approaches have addressed separation and identification of concerns at the source code level. Aspect-oriented programming advocates the modularization of concerns as separate aspect modules that are weaved into the existing implementation at certain join points [12]. Other approaches just manage separate concern structures on top of the existing implementation [9, 19, 10]. In this case, a concern is realized in source code as a set of conceptually related, but seemingly unrelated code elements.

Being able to identify concerns and explore the system based on concerns is especially important during evolution and long term maintenance. Concerns can enhance understandability, since they document the purpose of the artifacts that belong to them. Concerns can define the scope of a change, since they point out related artifacts that might be affected. And, since concerns can also represent functional and non-functional features, they can improve traceability of artifacts throughout the development lifecycle.

Our research is aimed at bringing these benefits in the context of software architecture. Software architecture presents high-level models of a system, usually consisting of components, connectors and their configuration, abstracting the smaller details. In this way, the architecture provides an overview of the system together with its most important properties. As concerns not only crosscut existing representations in code but can transcend each individual development phase, software architecture descriptions need to accommodate concerns, and support for tracing these concerns between architecture and source code is needed.

In this paper we present our approach for identification and management of concerns at the architectural level. We provide tool support for identification of concerns in code, displaying them at the architectural level, and managing their evolution over time. We plan to use this tool to gain more insight into what types of concerns are relevant at the architectural level and what types of information related to these concerns would have the most impact during development.

2. BACKGROUND AND RELATED WORK

Our work relates to two main research fields: concern-based development and software architecture.

2.1 Concern-based Development

A number of approaches were developed for identifying and managing concerns at the source code level. In all these approaches, the set of elements that comprise a concern can be specified either as a query, in which case the concern specification is intensional, or as a fixed set, which denotes an extensional concern. At any point in time, an intensional concern can be evaluated in order to produce its extensional counterpart, but while software evolves, the extension of an intensional concern might change too.

CME [9] consists of a number of Eclipse [8] plug-ins that support concern description and exploration. A central, hierarchical concern model is provided in Concern Explorer,

and custom loaders can load concerns from source code and other development artifacts. The concern model can be explored using a query language, but at the same time each query represents an intensional concern. Concern Highlight [15] was built as an extension to CME that can highlight concern occurrences in source code files.

FEAT [19, 18] describes concerns as Concern Graphs. Concern graphs model concerns as relations between program elements, including classes, fields and methods. FEAT is built as an Eclipse plug-in and offers support for both defining the concern queries and evaluating these queries in order to determine the extension of a concern. Furthermore, the extension set can be recorded at any time the source code changes, and a concern graph inconsistency is detected if the new extension set differs from the previous one.

Intensional Views [10] describe concerns as a set of source code entities, either classes or methods. The concern description is intensional, using a custom predicate language that can evaluate relations over source code elements, implemented for Smalltalk programs. A concern can have multiple alternative definitions, each such definition producing an extensional set of occurrences. A concern definition is consistent if the extension sets from all its alternate definitions are the same.

Aspect-oriented software development [12] and aspect mining are only tangentially related, since we are interested in identification of concerns rather than in modularizing them or in automatically inferring them from code at this point.

2.2 Software Architecture

The architecture of a software system is a high level model described using an architecture description language (ADL). In supporting linking the architecture to implementation, some ADLs have either associated implementation frameworks, such as xADL2.0 [7], or in the case of ArchJava [1] the code and architectural description are tied together in a single program written using Java language extensions for architectural element notations. These approaches constrain the implementation to using the framework or the architectural extension notations, and therefore they cannot be used to show the architecture of a system that does not adhere to the constraints. Tracing concerns is a different approach to linking architecture to implementation without constraining either one of them.

Concerns that span the entire development lifecycle are also known as *early aspects* [3]. The Theme [2] approach discusses identification and traceability of concerns from requirements to UML design. While this work is interesting in the way it addresses aspect-orientation in other types of artifacts than code, it does not provide insights into how concern management can be done at the architectural level.

There is relatively recent interest in investigating representation of concerns in architecture, but without any concrete resolutions. DAOP-ADL [17] extends an architectural description language with first-level aspect constructs, and provides support for composition between aspects and components. Katera and Kats [11] propose an aspect architecture view where aspects are reusable elements that are composed together but discusses architecture in context of UML, and not in a specific ADL. And a another paper [4] argues that architectural abstractions are essentially concerns and there is no need for new first-level aspect elements.

3. APPROACH

At the basis of our approach is the support for side-by-side development of architecture and code. A common development setup today includes two monitors, and we want to use this screen real estate to leverage the use of traceability between architecture and code: one monitor can accommodate code development, while the other will show the high-level architecture. If changes in the code influence the architecture, then they should be reflected at the architectural level. A quick view on the architectural level should show up problems with source code concerns.

One of the hypotheses that we plan on investigating is that concerns can be used to describe the rationale in a software architecture. A seminal paper in software architecture [16] defines the architecture of a system as a tuple consisting of *Elements*, *Form* and *Rationale*. A survey of different ADLs [13] shows that although they support descriptions of different structural and behavioral properties of a system, a common denominator is the description of components, connectors and their configuration. While these cover the elements and the form of an architecture, a representation of the rationale is generally missing from these ADLs.

We also want to transform software architecture so that it becomes the place for showing high-level concern information. Source code can become very complex, especially in large systems, and concern identification in code will generate concern structures of comparable complexity. Since architectures are primarily aimed at giving an overview of the system, they can be used to show information about how a concern is implemented in code at a higher level of abstraction. An overview of the architecture, in relation to a specific concern, should be enough for somebody to figure out 1) if the concern is implemented at all 2) which components, connectors, interfaces implement the concern 3) to what degree is the concern implemented 4) if there are any concern-related errors or inconsistencies.

Although software architecture is a relatively mature field, developing explicit architectural descriptions is still not a generally accepted practice in common software development. This is partially due to the fact that the link between ADLs and implementation has been generally overlooked or considered trivial, and partially because maintaining consistency between architecture and implementation over time is not an easy task. During evolution, the architectural description gradually becomes inconsistent with the implementation, phenomenon known as *architectural erosion*, which causes the architecture to lose its value.

One of the reasons behind architectural erosion is that there are no well-defined mappings between the formalisms used at the architectural level, such as components and connectors, interfaces and links, and the formalisms used at the implementation level, such as classes, interfaces, methods. A hypothesis that we plan to investigate is that this problem can be alleviated by using concerns as links between architecture and code, transforming the architecture from being an one-time blueprint for the implementation to being an accurate representation of the system at all times.

4. CURRENT STATUS

In order to explore these question in more detail, we have build tool support for an integrated environment where concerns can be traced from architecture to implementation and

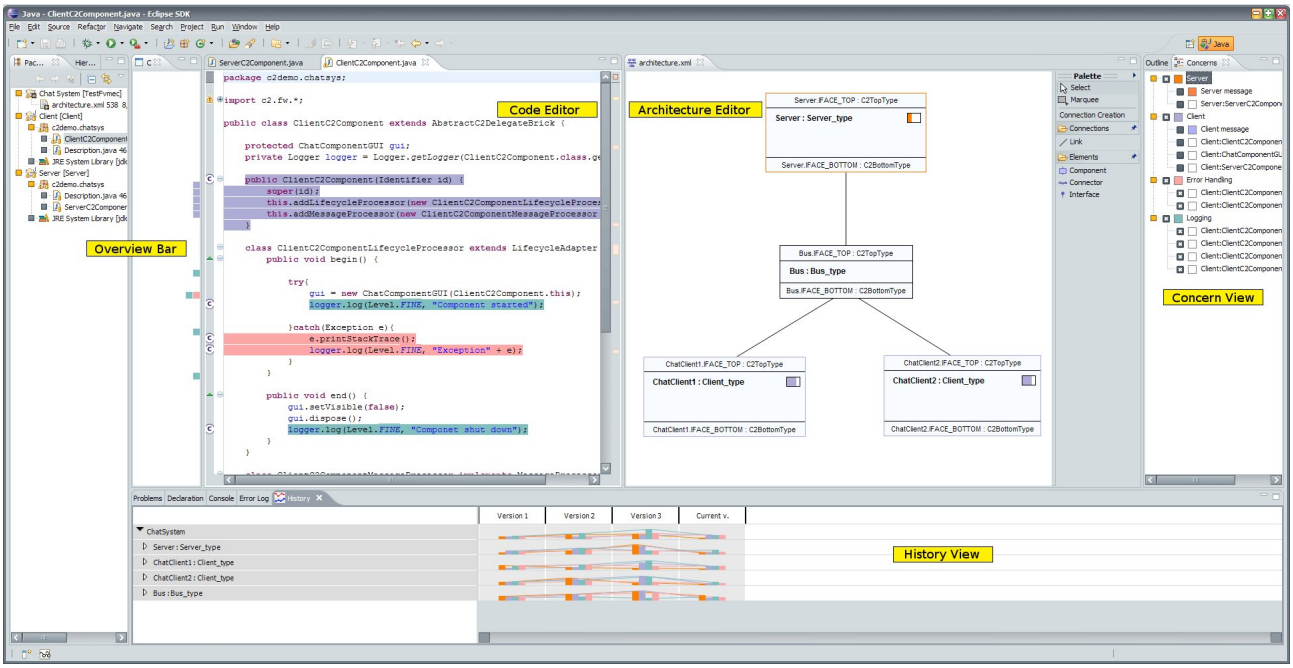

Figure 1: Side-by-side code and architecture editors with concern highlight views.

back, with their evolution being managed as architecture and code evolve.

Our tool consists of a number of Eclipse plug-ins. Eclipse is an easily extensible environment that already provides extensive support for source code development in Java. In order to enable our side-by-side editors vision, we have developed a graphical editor for software architecture. The editor is used to create boxes-and-arrows architectural diagrams based on more complex descriptions that use the xADL2.0 [6] language.

An integral part of our solution is the link between architecture and code provided by ArchEvol [14]. ArchEvol links components and connectors from an architectural description to Eclipse Java projects that contain their source code implementation. At the same time, if architectural projects and implementation projects are stored in a Subversion [5] repository, ArchEvol can help maintain the mapping links accurate between their different revisions. While this is a coarse-level link between architecture to code, concerns provide a finer-grained mapping of architecture to code elements.

Figure 1 shows our environment, with the source code editor on the left and the architecture editor on the right. On top of this setup, we provide support for visualization of concerns in a number of other views. The main view is the Concern View, on the far right side, which manages the hierarchy of concerns and helps select concerns for visualization.

At the source code level, concerns are highlighted in the current editor or shown on a per-line basis in the overview bar on the left of the editor. While related research uses intensional concern definitions, we use extensional concern definitions, identified by user selection of source code text. Although it is a simple concern identification method, it is also the one that gives the most options, allowing the user to select any relevant parts in the source code text, without being limited to only types, methods or fields. For example, this is useful when a concern is related to only a parameter in a method or to partial blocks of code inside a method.

Annotating concerns in code can become a time consuming, tedious task. In order to ease this effort, we also support collection of concern-based information while the code is being developed, based on the observation that development tasks are related to one or more concerns that are addressed at a time. Using the automated concern recording feature, the developer can choose a number of concerns as active, and all subsequent code changes will be automatically mapped as belonging to those concerns.

A number of metrics can be used to show useful information at the architectural level based on the source code mappings of concerns. Default metrics included with the tool show the presence of a concern in the architecture or aggregate values related to the concern, such as the percentage of implementation files that have the concern marked over the total number of files.

With the help of ArchEvol, information in both the architecture and code can be versioned together on top of an Subversion repository. The values of the concern metrics are saved along with each version of the system, in order to show the evolution of concerns over time. The History View, placed at the bottom of the figure, shows the architectural elements tree on the left, and for each version from the repository, shows the concerns and the associated values over different architectural versions.

Of course, not all problems are fully addressed at this point in time. One question in particular that we would like to address in the future is how can this environment be used in a collaborative and distributed environment.

Up to this point, we have built the necessary infrastructure that allows further experimentation with concern identification and management. As future work, we plan to use this environment to get better insights into how to map and

identify concerns in architecture and code, and we expect this to materialize in a set of extensions to architectural description languages needed to accommodate concerns.

5. CONTRIBUTIONS AND EVALUATION

In this paper we have presented our vision for explicit concern identification and management in both architecture and code. This vision is supported by a tool that allows definition of concerns in architecture and code, offers support for visualization of concern location and concern-related properties, and stores the evolution of these concerns for further analysis.

We expect that this tool will help us identify how concerns can take a more prominent role in software development and especially in software architecture. We want to use these lessons in creating an architectural description language and tool support for defining architectural descriptions that can remain consistent with the source code implementation throughout evolution.

In order to evaluate our assumptions, we plan to use our tool to identify and manage concerns on an existing application. A medium size open source application will provide a good test bed since it contains a large enough source code base, will have a non-trivial architecture, and will have a number of design documents, bug reports and feature plans needed to determine what are the important concerns and how these concerns manifest themselves in architecture and code. This evaluation will also give us information about the limits of scalability of our visualizations and the evolvability of our concern mappings.

6. REFERENCES

[1] J. Aldrich, C. Chambers, and D. Notkin. Archjava: Connecting software architecture to implementation. In *Proceedings of the 24th International Conference on Software Engineering*, 2002.

[2] E. Baniassad and S. Clarke. Theme: An approach for aspect-oriented analysis and design, 2004.

[3] E. Baniassad, P. C. Clements, J. Araujo, A. Moreira, A. Rashid, and B. Tekinerdogan. Discovering early aspects. *IEEE Softw.*, 23(1):61–70, 2006.

[4] T. Batista, C. Chavez, A. Garcia, A. Rashid, C. Sant'Anna, U. Kulesza, and F. C. Filho. Reflections on architectural connection: seven issues on aspects and adls. In *EA '06: Proceedings of the 2006 international workshop on Early aspects at ICSE*, pages 3–10, 2006.

[5] Collabnet. *Subversion*. http://subversion.tigris.org, 2005.

[6] E. M. Dashofy, A. van der Hoek, and R. N. Taylor. A highly-extensible, xml-based architecture description language. In *Working IEEE/IFIP Conference on Software Architecture (WICSA 2001)*, 2001.

[7] E. M. Dashofy, A. van der Hoek, and R. N. Taylor. An infrastructure for the rapid development of xml-based architecture description languages. In *24th International Conference on Software Engineering (ICSE 2002)*, 2002.

[8] Eclipse Foundation. *Eclipse*. http://www.eclipse.org, 2005.

[9] W. Harrison, H. Ossher, S. Sutton, and P. Tarr. Concern modeling in the concern manipulation environment. In *MACS '05: Proceedings of the 2005 workshop on Modeling and analysis of concerns in software*, pages 1–5, 2005.

[10] F. P. K. Mens, A. Kellens and R. Wuyts. Co-evolving code and design with intensional views - a case study. In *Elsevier Journal on Computer Languages, Systems and Structures*, pages 140–156, October 2006.

[11] M. Katara and S. Katz. Architectural views of aspects. In *AOSD '03: Proceedings of the 2nd international conference on Aspect-oriented software development*, pages 1–10, 2003.

[12] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *Proceedings European Conference on Object-Oriented Programming*, volume 1241, pages 220–242, 1997.

[13] N. Medvidovic and R. N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, 26(1):70–93, 2000.

[14] E. C. Nistor, J. R. Erenkrantz, S. A. Hendrickson, and A. van der Hoek. Archevol: versioning architectural-implementation relationships. In *SCM '05: Proceedings of the 12th international workshop on Software configuration management*, pages 99–111, 2005.

[15] E. C. Nistor and A. van der Hoek. Concern highlight: A tool for concern exploration and visualization. In *AOSD 2006 Workshop on Linking Aspect Technology and Evolution (LATE)*, March 2006.

[16] D. E. Perry and A. L. Wolf. Foundations for the study of software architecture. *SIGSOFT Softw. Eng. Notes*, 17(4):40–52, 1992.

[17] M. Pinto, L. Fuentes, and J. M. Troya. Daop-adl: an architecture description language for dynamic component and aspect-based development. In *GPCE '03: Proceedings of the second international conference on Generative programming and component engineering*, pages 118–137, 2003.

[18] M. P. Robillard and G. C. Murphy. Concern graphs: finding and describing concerns using structural program dependencies. In *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*, pages 406–416, 2002.

[19] M. P. Robillard and G. C. Murphy. Just-in-time concern modeling. *SIGSOFT Softw. Eng. Notes*, 30(4):1–3, 2005.

[20] P. Tarr, H. Ossher, W. Harrison, and J. Stanley M. Sutton. N degrees of separation: multi-dimensional separation of concerns. In *ICSE '99: Proceedings of the 21st international conference on Software engineering*, pages 107–119, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.