The Concern Manipulation Environment [OOPSLA/GPCE]

Peri Tarr, William Chung, William Harrison,
Vincent Kruskal, Harold Ossher,
Stanley M. Sutton Jr.
IBM Thomas J. Watson Research Center
P.O. Box 708
Yorktown Heights, NY 10598 USA
+1 914 784 7279
tarr@watson.ibm.com

Andrew Clement, Matthew Chapman,
Helen Hawkins, Sian January
IBM Hursley Park
Hursley Park
Hursley SO212JN England
+44 01962 816658
clemas@uk.ibm.com

ABSTRACT

The Concern Manipulation Environment (CME) aims to provide a set of open, extensible components and a set of tools that promote aspect-oriented software development (AOSD) throughout the software lifecycle. It has two main goals:

- To provide an open, integrated development environment (IDE) to enable software engineers to use AOSD techniques throughout the software lifecycle, and to allow them to use different AOSD approaches in an integrated manner.
- To promote the rapid development of new tools supporting AOSD at any stage of the software lifecycle, and to serve as an integrating platform for such tools, facilitating development and experimentation with new AOSD approaches.

This demonstration will highlight a number of tools and components that are useful to software developers and to AOSD tool providers and researchers. Tools for software developers include ones that allow developers to identify, model and visualize concerns, aspects and relationships in their software, covering software artifacts of any type, including both code and non-code artifacts, and including latent concerns or aspects that were not separated in the artifacts; that enable flexible queries over software; and that compose/integrate aspects and other concerns. For AOSD tool providers and researchers, the demonstration will describe some of the CME's support for integration of tools and approaches within the environment, highlighting the integration of Java, AspectJ and Ant artifacts within the CME, and how to use the CME's extensible components to create new AOSD tools or prototypes rapidly.

Categories and Subject Descriptors

D1.m [**Programming Techniques**]—*aspect-oriented software development* D2.6 [**Software Engineering**]: Programming Environments—*integrated environments, programmer workbench.* D.3.3 [**Programming Languages**]: Language Constructs and Features–*frameworks, patterns.*

General Terms

Design, Languages.

OOPSLA'04, Oct. 24–28, 2004, Vancouver, British Columbia, Canada.
ACM 1-58113-833-4/04/0010.

Keywords

Aspect-oriented software development (AOSD), separation of concern, software design, software composition, integration, extraction, concern modeling, software query, full-lifecycle software engineering, integrated development environment, Eclipse open source

1. INTRODUCTION

The area of aspect-oriented software development (AOSD) has seen much progress in the past few years towards improving the quality of object-oriented, generative, and component-based software engineering, including some use in large-scale applications. Research and development efforts have demonstrated that support for large-scale AOSD must address the full software lifecycle, with all its tasks, activities, artifacts, their interrelationships, and consistency across them. Thus, large-scale AOSD requires tool, paradigm, and methodology support for:

Multiple aspect models: Different aspect models come with different benefits and different costs. It is clear that any given development effort will often need different approaches to accomplish different tasks and goals, and at different stages of the software lifecycle.

Multiple artifacts and formalisms: Different software engineering activities produce and manipulate different artifacts, such as use cases, designs, architectures, models, code, build and deployment artifacts (e.g., Ant scripts), test cases, and bug reports, and each type of artifact may be represented using different formalisms, such as source and binary files, databases, and collaborative development software.

Multiple tasks and activities: A variety of tasks and activities occur in different software development processes and methodologies..

Despite the progress in the AOSD field, we have noted the difficulty research and development efforts have in producing new tools, paradigms, and methodologies, and the difficulty which AOSD end users have in adopting these technologies. This is, we believe, largely because the development of tools to realize different AOSD approaches represents a huge investment of time and effort, as each one must currently be built from scratch or from low-level abstractions. Consequently, the tools themselves represent isolated point solutions, and rarely have any ability to interoperate or be integrated. This has impeded the development and validation of full-lifecycle AOSD support, and the extension

of existing technologies to other paradigms, artifacts, formalisms, tasks, and activities. It has also significantly hindered the use of existing tools and paradigms by end users, who find themselves unable to use available tools and paradigms together.

The Concern Manipulation Environment (CME) [4] aims to provide a set of open, extensible components and a set of tools that promote aspect-oriented software development (AOSD) throughout the software lifecycle, addressing all of the points above. Its goals are:

- To provide an open, integrated development environment (IDE) for those producing software using aspect-oriented software development techniques throughout the software lifecycle, and to allow developers to use different AOSD approaches in an integrated manner.
- To promote the rapid development of new tools supporting AOSD at any stage of the software lifecycle, and to serve as an integrating platform for such tools, enabling development and experimentation with new AOSD approaches.

Towards the first goal, the CME initially surfaces tool support for two major AOSD approaches: the next generation of Hyper/J and multidimensional separation of concerns [2], and AspectJ [1], while providing underlying support for a broad spectrum of AOSD approaches. These two approaches and sets of tools are integrated seamlessly within the CME, along with (and largely through the means of) generic support for querying software and for modeling, extracting and composing concerns.

To address the second goal, the CME provides a rich collection of reusable components, with a wide variety of open points, which provide AOSD tool developers with higher-level abstractions with which to build and/or integrate AOSD tools and other artifact languages, such as UML, C++, XML, etc. The CME is now an Eclipse [3] open source project, and the first beta release occurred in April 2004. The tools are built as Eclipse plugins, but the underlying components do not depend on Eclipse, and they can be used to build freestanding tools or tools for other environments.

For those seeking IDE support for AOSD, this demonstration highlights some CME tools. Several are useful during regular software development, even without AOSD. Tools to be shown include ones that:

- Identify, model and visualize concerns and aspects in software and relationships among concerns and software *units* (i.e., pieces of software artifacts of any type, including both code and non-code artifacts). Concerns can be identified up-front, during the software's initial creation, or on demand, at any point during the software's lifecycle when the concern manifests itself. For example, an end-user feature such as "XML streaming" may be identified early and encapsulated as a separate concern. Other concerns may manifest later in the software lifecycle. For example, when faced with the task of migrating from one security protocol to another, a developer might want a concern that encapsulates, non-invasively, all parts of the software that pertain to the original security protocol, even if they were not encapsulated together in the initial software design.

  The identification and modeling activity promotes software understanding and multiple perspectives, refactoring, and re-engineering, and it is also generally the first activity developers undertake when identifying and separating particular aspects or concerns. Note that this capability adds value for both AOSD developers and Java developers who are not using AOSD, and thus it provides a no-cost entry path to AOSD.
- Enable flexible queries over software, which help developers to find concerns and aspects in existing software, and to navigate and understand the software and its interrelationships. The entry barrier for using this capability is also low, since this tool provides the ability to query standard, existing Java, AspectJ, and Ant software.
- Flexibly compose/integrate aspects and other concerns. This supports, for example, optional features or instrumentation, mix-and-match of features, and product lines.
- Disentangle latent concerns or aspects from existing software, making them first-class entities.

Development incorporating multiple types of artifacts, such as Java, AspectJ, Ant, and UML, towards full lifecycle AOSD support, will be described, as will the integrated use of Ant build control, AspectJ weaving and Hyper/J composition on the same software and projects.

For AOSD tool providers and researchers, this demonstration will describe:

- Some of the CME's support for integration of tools and support for new languages and paradigms within the environment. Different levels of integration are supported. For example, at the highest level, the concern and relationship modeling and visualization capabilities are generic across different AOSD approaches, and can be used to provide end users with uniform access to concerns, aspects, and relationships, no matter what formalism or language is used to represent the underlying artifacts. Deeper integration occurs via open points within various components. Support for other artifacts, such as AspectJ and Ant, are provided and integrated using different CME open points.
- How to use the CME's extensible components to create and/or integrate AOSD tools, technologies, or prototypes more rapidly. For example, high-quality tools for any AOSD approach must clearly identify join points and show what is attached to them. The CME provides generic modeling and visualization support for this.

For more information, see the CME web site at http://www.eclipse.org/cme.

2. REFERENCES

[1] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm and William C. Griswold. "Getting Started with AspectJ." *CACM* 44(10): 59–65, October 2001.

[2] Harold Ossher and Peri Tarr. "Using Multi-Dimensional Separation of Concerns to (Re)Shape Evolving Software." *CACM* 44(10): 43–50, October 2001.

[3] http://www.eclipse.org.

[4] http://www.eclipse.org/cme.