SECOND INTERNATIONAL CONFERENCE ON

Aspect-Oriented Software Development

March 17–21, 2003

Northeastern UNIVERSITY

Higher Learning. Richer Experience.

Table of Contents

Registration:

Sunday:	4 PM–8 PM, second-floor foyer, Sheraton Boston Hotel
Monday:	7 AM–5 PM, 240 Raytheon Amphitheater, Egan Research Center, Northeastern University
Tuesday:	7 AM–3 PM, outside 240 Raytheon Amphitheater, Egan Research Center, Northeastern University
Tuesday:	4 PM–8 PM, second-floor foyer, Sheraton Boston Hotel
Wednesday:	7:30 AM–5 PM, second-floor foyer, Sheraton Boston Hotel
Thursday:	7:30 AM–5 PM, second-floor foyer, Sheraton Boston Hotel

The following abbreviations are used throughout this brochure:

AO:	aspect-oriented
AOP:	aspect-oriented programming
AOSD:	aspect-oriented software development
IDE:	integrated development environment
UML:	Unified Modeling Language

Conference Program

	NORTHEASTERN UNIVERSITY		SHERATON BOSTON HOTEL		
	Monday	**Tuesday**	**Wednesday**	**Thursday**	**Friday**
7:45–8:45 AM		**Talk** *I. Jacobson* (Egan, Raytheon Amphitheater)			
8:45–9:00 AM			**Opening**		
9:00–10:30 AM	**T1** (Sh 220) **T2** (Sh 210) **ACP4IS** (Sh 215) **Early Aspects** (Sh 315) **FOAL** (Sh 415)	**T5** (Sh 220) **T6** (Sh 210) **AOM** (Sh 215) **COMM** (Sh 315) **SPLAT** (Sh 415)	**Keynote** *G. Kiczales* (SBH, Republic Ballroom)	**Keynote** *S. Matsuoka* (SBH, Republic Ballroom)	**Papers G** (SBH, Republic Ballroom) **Demos IV** (SBH, Commonwealth Room)
11:00–12:30 PM	**T1** (Sh 220) **T2** (Sh 210) **ACP4IS** (Sh 215) **Early Aspects** (Sh 315) **FOAL** (Sh 415)	**T5** (Sh 220) **T6** (Sh 210) **AOM** (Sh 215) **COMM** (Sh 315) **SPLAT** (Sh 415)	**Papers A** (SBH, Republic Ballroom) **Demos I** (SBH, Back Bay D)	**Papers D** (SBH, Republic Ballroom) **Demos II** (SBH, Back Bay D)	**Papers H** (SBH, Republic Ballroom) **Demos I** (SBH, Commonwealth Room)
2:00–3:30 PM	**T3** (Sh 220) **T4** (Sh 210) **ACP4IS** (Sh 215) **Early Aspects** (Sh 315) **FOAL** (Sh 415)	**T7** (Sh 220) **T8** (Sh 210) **T9** (Sh 325) **AOM** (Sh 215) **COMM** (Sh 315) **SPLAT** (Sh 415)	**Papers B** (SBH, Republic Ballroom) **Demos II** (SBH, Back Bay D)	**Papers E** (SBH, Commonwealth Room) **Demos III** (SBH, Back Bay D)	
4:00–5:30 PM	**T3** (Sh 220) **T4** (Sh 210) **ACP4IS** (Sh 215) **Early Aspects** (Sh 315) **FOAL** (Sh 415)	**T7** (Sh 220) **T8** (Sh 210) **T9** (Sh 325) **AOM** (Sh 215) **COMM** (Sh 315) **SPLAT** (Sh 415)	**Papers C** (SBH, Republic Ballroom) **Demos III** (SBH, Back Bay D)	**Papers F** (SBH, Republic Ballroom) **Demos IV** (SBH, Back Bay D)	
5:30–7:00 PM			**Reception** (SBH, Independence Ballroom) *(18:00–19:00)*	**Banquet** (SBH, Back Bay C) *(18:30–20:30)*	
7:00–9:00 PM	**Birds of a Feather Sessions** (Egan)	**Birds of a Feather Sessions** (Egan)	**Student Research Extravaganza** (SBH, Republic Foyer)	**Banquet** *(continued)*	

Monday and Tuesday's events take place at Northeastern University; all others take place at the Sheraton Boston Hotel.

Sh = Shillman Hall, Northeastern University
SBH = Sheraton Boston Hotel
Egan = Egan Research Center, Northeastern University

Welcome to AOSD 2003

It is our pleasure to welcome you to AOSD 2003, the Second Conference on Aspect-Oriented Software Development. After last year's successful meeting in the Netherlands, we are confident that this new conference will be the major meeting forum for exchanging ideas and experiences, and demonstrating new techniques and results, in the rapidly growing field of aspect-oriented software development.

This second AOSD conference is organized by a team from Northeastern University under the auspices of the Aspect-Oriented Software Association, led by an international steering committee. The conference is sponsored by IBM Research and Intentional Software Corporation, and is held in cooperation with ACM SIGPLAN and ACM SIGSOFT.

The first two days of the conference are dedicated to workshops and tutorials, located at Shillman Hall on Northeastern University's campus. The main conference, including demonstrations, takes place at the Sheraton Boston Hotel, located at the Prudential Center. The conference program offers two keynote presentations, as well as research papers and a practitioner's report session. Of special interest to PhD students is the Student Research Extravaganza on Wednesday evening.

Thank you for joining us in Boston. We wish you a pleasant stay and a productive conference.

William Griswold, *General Chair*

Mehmet Aksit, *Program Chair*

Karl Lieberherr, *Organizing Chair*
College of Computer and Information Science
161 Cullinane Hall
Northeastern University
360 Huntington Avenue
Boston, MA 02115-5000
617.373.2077 (voice)
617.373.5121 (fax)
lieberherr@ccs.neu.edu (e-mail)

Supporters

AOSD 2003 is organized in cooperation with the Association for Computing Machinery SIGPLAN and SIGSOFT special interest groups. **The Association for Computing Machinery** (ACM) is an international scientific and educational organization dedicated to advancing the arts, sciences, and applications of information technology. With a world-wide membership of 80,000, ACM functions as a locus for computing professionals and students working in the various fields of information technology.

IBM Research is a financial supporter of the conference. Computer Science at IBM Research has more than one thousand researchers located at eight labs around the world. Leading-edge research across many disciplines, including programming languages and software engineering, is often done in concert with colleagues in academic and government research centers, as well as "in the marketplace" with customers who provide challenging research problems.

Intentional Software Corporation is a financial supporter of the conference.

Verizon Communications is a financial supporter of the conference.

The AOSD workshop and tutorial program is hosted by Northeastern University.

The **Aspect-Oriented Software Association** is a non-profit organization whose mission is to be the primary sponsor for the annual Conference on Aspect-Oriented Software Development.

Tutorials

T1: Aspect-Oriented Programming with AspectJ

Erik Hilsdale, Palo Alto Research Center
Wes Isberg
Level: Introductory

AspectJ is a seamless AO extension to Java. It can be used to cleanly modularize the crosscutting structure of concerns such as exception handling, multi-object protocols, synchronization, performance optimizations, and resource sharing.

When implemented in a non-AO fashion, the code for these concerns typically becomes spread out across entire programs. AspectJ controls such code-tangling and makes the underlying concerns more apparent, making programs easier to develop and maintain.

This tutorial will introduce AOP and show how to use AspectJ to implement crosscutting concerns in a concise, modular way. We will also demonstrate and use AspectJ's integration with IDEs such as JBuilder, NetBeans, Emacs, and Eclipse, in addition to the core AspectJ tools.

AspectJ is freely available at http://eclipse.org/aspectj/.

T2: JMangler: On-the-Fly Transformation of Java Class Files

Günter Kniesel, University of Bonn
Michael Austermann, SCOOP Software GmbH
Level: Intermediate

JMangler is a freely available framework for load-time transformation of compiled Java programs. The transformation of byte code is a core technology that can be used for a variety of purposes, including the implementation of new AOSD languages and tools. This tutorial will provide attendees with a thorough understanding of JMangler and Java byte code transformation, from an AOSD perspective.

Unlike simple byte code transformation libraries, JMangler provides a complete solution for hooking into the class loading process. It does so in a JVM and class loader independent way, which also works for classes that employ their own custom class loaders. Therefore, JMangler can transform any application classes and can be used in environments such as application servers, which make heavy use of custom class loaders.

Since JMangler uses no source code, it even can be applied to third-party libraries. Since it works at load time, it provides the guarantee that transformations will be applied to every class that will be executed at run-time, even if the class is created dynamically or loaded from some possibly remote host.

For programmers, we will provide guidance and hands-on experience in writing, composing, and applying their own transformer components. For AOSD language and tool developers, we will show how to use JMangler's capabilities for load-time aspect weaving and for injecting hooks that enable run-time weaving. For managers, our aim is to convey an understanding of the potential of load-time byte code transformation.

The tutorial will conclude with a demonstration of CC4J, a powerful code-coverage tool developed with JMangler. CC4J is an application that transforms programs in a way that is beyond the scope of all known high-level AOSD languages and systems.

T3: Hyper/J: Multi-Dimensional Separation of Concerns for Java

Peri Tarr, IBM, T. J. Watson Research Center
Harold Ossher, IBM, T. J. Watson Research Center
Stanley M. Sutton Jr., NFA
Level: Introductory

Multi-dimensional separation of concerns (MDSOC) is an AOSD approach that promotes flexible, powerful separation and integration of software based on all kinds of concerns. MDSOC allows developers to encapsulate overlapping, interacting, and crosscutting concerns, including features, aspects, variants, roles, business rules, components, frameworks, etc. MDSOC further supports developers in specifying relationships among concerns to allow concerns to be systematically separated, combined, and traced.

MDSOC treats all concerns as first-class and co-equal, including components and aspects, allowing them to be encapsulated and composed at will. This is in contrast to most AO approaches, which enable aspects to be

composed with components, but do not support composition of components (or aspects) with one another. Some other key benefits of MDSOC include the ability to identify and modularize concerns at any stage of the software lifecycle (prospectively and retrospectively); the ability to define, manipulate, and integrate different decompositions of the same software simultaneously; the ability to customize, adapt, reuse, and evolve software non-invasively; and the ability to reconcile different class hierarchies modeling overlapping domains from different perspectives. MDSOC can therefore reduce complexity, improve reusability, and simplify evolution and integration.

This tutorial describes how to accomplish MDSOC with Hyper/J, a prototype tool available for free download, in the context of standard Java development. It will demonstrate how Hyper/J addresses some real, pervasive problems in participants' own Java development. Example problems include adding a feature; adding instrumentation; creating and evolving product lines; separating concerns in retrospect (i.e., extracting concerns from existing software); supporting team development, including use of different domain models by different teams; separating, integrating, retrofitting, and reusing design patterns; and facilitating unplanned integration. Hyper/J works on Java class files, so it can be used on any off-the-shelf Java software, even when source code is not available. It requires no special compilers, development tools, or processes.

T4: Dynamic and Distributed Aspect-Oriented Programming with JAC

Renaud Pawlak, University of Lille
Lionel Seinturier, Laboratoire d'Informatique de Paris 6
Level: Intermediate

Modern business systems run in open and distributed environments, often involve the Web, and are often based on industry-standard middleware such as CORBA and Java RMI. Distributed applications are critical to businesses and must deal with concerns such as data consistency, scalability, dynamic resource discovery, fault tolerance, run-time maintenance, and remote version updating. For all these issues, the need for dynamic and fast software reconfiguration is increasing. Aspect-oriented techniques can provide powerful ways to deal with such challenges: indeed, separating concerns not only makes applications easier to develop and maintain, but also offers means to add or remove concerns to applications in a dynamic fashion at run time.

This tutorial presents Java Aspect Component (JAC), a fully operational programming environment for developing aspect-oriented, dynamically reconfigurable, distributed, and Web-based software. JAC offers the programmer a set of concepts for creating, manipulating, and composing aspects at run time within distributed and changing environments.

In this tutorial, we describe the entire process of developing and configuring AO software with JAC. After introducing the basic concepts and features of the environment, we explain why it is suitable for developing distributed applications. We show how to design an application in JAC's UML-flavored IDE, and we configure that application to work with the Web and use specific application-level aspects to implement a fully running online store. We then demonstrate the clustering features of JAC by deploying the application atop JAC remote containers using a communication layer implemented on RMI or CORBA. We show how to include dynamic adaptation features that allow the application to react to changing environments. Finally, at the end of the tutorial, we demonstrate the rapid development of applications and reuse of aspect capabilities.

JAC is available from http://jac.aopsys.com/.

T5: Advanced Aspect-Oriented Programming with AspectJ

Erik Hilsdale, Palo Alto Research Center
Wes Isberg
Level: Advanced

This tutorial will provide involved hands-on programming exercises that both use some of AspectJ's advanced features, and feature AspectJ used in advanced contexts. We will show how AspectJ can be used to solve problems in instrumentation (including logging), testing, quality management, and feature management. In addition, advanced parts of the AspectJ design and implementation will be introduced, along with discussions of possible future features. Exercises will use the core AspectJ tools and IDEs.

AspectJ is freely available at http://eclipse.org/aspectj/.

T6: Identifying and Modeling Aspects Using Domain Analysis Techniques

Mehmet Aksit, University of Twente
Lodewijk Bergmans, University of Twente
Level: Advanced

Although object-oriented design methods and programming languages offer several advantages, experience has shown that effective composition of software remains a difficult task. This is especially true if the software system is large and employs complex crosscutting behavior. Aspect-oriented techniques can help to manage such complexity, and "best practices" are beginning to emerge for applying these techniques to both new and existing software.

This tutorial consists of two parts: (1) guidelines for identifying aspects and (2) approaches to modeling and implementing aspects with current AOSD technologies.

The first part of the tutorial provides a set of guidelines to identify the obstacles that software engineers may encounter in designing large systems using object technology. To this aim, first, we will discuss general aspects that are common to many software domains. Second, we will consider the special characteristics and challenges of several specific kinds of software: application generators, concurrent systems, constraint systems, control systems, distributed systems, and real-time systems. This detailed analysis will help software engineers to identify the possible obstacles arising from crosscutting behavior in each of these areas.

In the second part of the tutorial, we will further describe the current AO design methods and languages in solving the identified obstacles. The tutorial will conclude with advantages and limitations of current aspect technologies, and give references to the relevant research activities.

T7: Feature-Oriented Programming for Product Lines

Don Batory, University of Texas at Austin
Level: Introductory

Feature-oriented programming (FOP) is both a design methodology and supporting tool for program synthesis. The goal is to specify a target program in terms of the features that it offers, and to synthesize an efficient program that meets these specifications. FOP has been used to develop product lines in widely varying domains, including compilers for extensible Java dialects, fire-support simulators for the U.S. Army, high-performance network protocols, and program verification tools.

GenVoca is a simple mathematical model of FOP that is based on step-wise refinement, a methodology for building programs by adding one feature at a time. The incremental units of implementation/design are refinements that encapsulate the implementation of an individual feature. GenVoca models of product-lines treat programs as values and refinements as functions (that map input programs to output programs with augmented features). Application designs are equations—compositions of functions and constants—that are amenable to optimization and analysis.

FOP and AOP are complementary, as both aspects and feature refinements encapsulate cross-cuts, i.e., fragments of multiple classes. The primary difference is emphasis: FOP follows more of a traditional object-oriented design approach to define cross-cuts, which focuses on how algorithms compose to build complete systems, rather than the AOP emphasis on join-points, point-cuts, and advice to modify existing programs.

This tutorial reviews basic results on FOP, including general models and tools for synthesizing a consistent set of code and non-code artifacts by composing refinements (cross-cuts), automatic algorithms for validating refinement compositions, synthesizing product-lines of product-families (e.g., tool suites), and automatic algorithms for optimizing application designs (equations).

T8: Aspect-Oriented Programming for Database Systems

Awais Rashid, Lancaster University
Level: Intermediate

Database systems are central to the day-to-day functioning of most businesses, but are notoriously costly to design and maintain. Like other software, database systems are subject to many crosscutting and overlapping concerns at both the design and implementation levels. AOP aims at easing software development by providing abstractions that serve to localize crosscutting concerns, e.g., code that cannot be encapsulated within one class but is tangled over many classes.

This tutorial will describe how AOP can be applied to address crosscutting concerns in database systems in order to make them more customizable, evolvable, and maintainable. The aims of the tutorial are fourfold. First, attendees will be introduced to the basic concepts of AOP. Second, the tutorial will highlight crosscutting concerns in database systems at both the DBMS and database levels. Third, the tutorial will describe the use of AOP concepts to achieve cost-effective customization and reduced maintenance overheads at the DBMS and database levels. Finally, new requirements imposed on database systems in terms of aspect storage and integration with AO programs will be discussed. Solutions to address these emerging requirements will be presented.

T9: Model-Driven Engineering with Contracts, Patterns, and Aspects

Jean-Marc Jezequel, IRISA
Level: Intermediate

The "non-functional" aspects of a software application—such as persistence, fault tolerance, and quality of service—should be separate and untangled from the "functional" aspects of that application. Furthermore, the specification of a non-functional aspect should be separate from any (platform-specific) implementation of that aspect. Languages and tools are needed to map from the design or model of an aspect to its ultimate implementation, for example, atop middleware such as .NET or others.

UML gives the software designer a rich set of views on a model, and also provides many ways for the designer to add non-functional annotations to a model. In this tutorial, we will show how to organize models around the central notions of (1) quality of service contracts for specifying non-functional aspects and (2) aspects for describing how those contracts can be implemented. Based on our experience in previous projects, we will show how to model contracts in UML with a small set of stereotypes, and how to represent aspects and applications of design patterns at the meta-model level using parameterized collaborations equipped with transformation rules expressed in an extension of OCL2.

The second part of this tutorial will present our transformation-based approach, implemented in our UMLAUT framework and tool, to build design-level aspect weavers. An aspect weaver, based on a meta-level interpreter, reads a platform-independent model (written in UML), processes the various aspect applications as specified by the designers, and then outputs a new platform-specific model (also in UML) that can serve as the basis for application code generation. Tutorial attendees will learn how to use UMLAUT for managing aspects in their own model-centric software development projects.

Workshops

ACP4IS: Aspects, Components, and Patterns for Infrastructure Software

Yvonne Coady, University of British Columbia
Eric Eide, University of Utah
David H. Lorenz, Northeastern University

Aspect-oriented programming, component models, and design patterns are modern and actively evolving techniques for improving the modularization of complex software. In particular, these techniques hold great promise for the development of "systems infrastructure" software, e.g., application servers, middleware, virtual machines, compilers, operating systems, and other software that provides general services for higher-level applications. The developers of infrastructure software are faced with increasing demands from application programmers needing higher-level support for application development. Meeting these demands requires careful use of software modularization techniques, since infrastructural concerns are notoriously hard to modularize.

Building on the meeting on the ACP4IS meeting at AOSD 2002, this workshop aims to provide a highly interactive forum for researchers and developers to discuss the application of and relationships between aspects, components, and patterns within modern infrastructure software. The goal is to put aspects, components, and patterns into a common reference frame and to build connections between the software engineering and systems communities.

Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design

João Araújo, Universidade Nova de Lisboa
Awais Rashid, Lancaster University
Bedir Tekinerdogan, Bilkent University
Ana Moreira, Universidade Nova de Lisboa
Paul Clements, Software Engineering Institute

This workshop aims to support the cross-fertilization of ideas in requirements engineering, software architecture design and AOSD. From a requirements engineering and architecture design perspective, aspects will improve and broaden the understanding of the identification and management of requirements and architecture-level concerns. From an aspect-orientation perspective, the workshop will provide attendees with a forum for discussing issues that can lead to a better understanding of how aspects can be used to support systematic and rigorous development of software from the very early stages.

The workshop will focus on challenges to defining methodical software development processes for aspects from early on in the software life cycle, and explore the potential of proposed methods and techniques to scale up to industrial applications.

FOAL: Foundations of Aspect-Oriented Languages

Gary T. Leavens, Iowa State University
Curtis Clifton, Iowa State University

FOAL is a forum for research in the foundations of AOP languages. Areas of interest include but are not limited to: semantics of AO languages, specification and verification of such languages, type systems, static analysis, theory of testing, theory of aspect composition, theory of aspect translation (compilation) and rewriting, and applications of such theories in practice (such as language design studies). The workshop aims to foster work in foundations, including formal studies, promote the exchange of ideas, and encourage workers in the semantics and formal methods communities to do research in the area of AOP languages.

COMM: Commercialization of AOSD Technology

Ron Bodkin, New Aspects of Security
Adrian M. Colyer, IBM UK
Juri Memmert, JPMDesign
Arno Schmidmeier, Sirius Software GmbH

This workshop will address the development of commercially successful AOSD technology. Topics of interest include value propositions, requirements for adoption (technical, organizational, standards), business cases, business models, strategies, industry lessons, selling, likely customers, and communication mechanisms. The goal is to bring together practitioners, users, consultants, and vendors to discuss the opportunities and challenges in delivering commercial solutions using AOSD. These discussions are intended to improve market opportunities and increase the scale and number of deployments of AOSD. This workshop will also start a conversation about mechanisms for cross-industry discussion and common initiatives to support market awareness and support for AOSD.

The workshop format will consist of structured discussions about topics drawn from position papers. A warm-up discussion will draw together various threads, discuss open issues, and reach conclusions.

SPLAT: Software-Engineering Properties of Languages for Aspect Technologies

Lodewijk Bergmans, University of Twente
Johan Brichau, Vrije Universiteit Brussel
Peri Tarr, IBM, Thomas J. Watson Research Center
Erik Ernst, University of Aarhus

The ultimate goal of AO languages and systems is to improve the quality of software by enhancing software-engineering properties such as modularity, comprehensibility, evolvability, composability, and analyzability. Consequently, each feature included in an AO language is intended to promote good software-engineering properties.

Yet the design of AO languages and systems is much more complex than it appears from examining a language construct or system feature in isolation. Each feature entails a trade-off among different software engineering properties—e.g., power vs. comprehensibility, flexibility vs. analyzability, complexity vs. evolvability. Moreover, individual features may interact in beneficial or undesirable ways, resulting in either improvement or loss of the software-engineering abilities targeted by each feature.

This workshop will advance the field of AOSD language design by emphasizing the need to understand the practical consequences of design decisions on the software-engineering properties of AO software. In particular, it will help language designers understand and evaluate the tradeoffs entailed by aspect language features, and address the need for consistent language design with respect to composability of language constructs and features.

AOM: Aspect-Oriented Modeling with UML

Omar Aldawud, Lucent Technologies
Mohamed Kandé, Swiss Federal Institute of Technology
Grady Booch, Rational Software Corp.
Bill Harrison, IBM, Thomas J. Watson Research Center
Dominik Stein, University of Essen

AO modeling is a critical part of AOSD that focuses on techniques for identifying, analyzing, managing, and representing crosscutting concerns in software design and architecture, while filling the gap between aspect-oriented requirements engineering and aspect-oriented programming.

This workshop is dedicated to the definition of AO modeling techniques, methods, and tools based on UML. Suggested issues are: How can we apply UML artifacts to AOSD? Are the existing notations and modeling techniques of UML sufficient to model aspects, or do we need to extend UML to support AOSD? Is UML the appropriate modeling language on which to base modeling for AOSD? Is UML capable of expressing "core" components and "aspectual" components as well as associations linking them together? If we have to extend UML, are the extension mechanisms provided by UML adequate? What could then be a UML profile for AOSD? Or would it be possible to rely only on a restricted subset of the UML for AOSD? What would this subset be?

Keynotes

Making the Code Look like the Design

Gregor Kiczales, University of British Columbia
and Intentional Software Corp.

Many software development advances can be characterized as making the code look more like the design—from the looping constructs of structured programming, to object-oriented programming and model-based development. The first part of this keynote will outline the contribution AOP is making to this evolution. What is unique about AOP? What does it share with previous advances? What can we learn from previous advances about work we still have to do?

The second part of the talk will explore a further step in the evolution. Intentional technology is a synthesis of object-oriented, aspect-oriented, and intentional programming techniques. This integrated technology enables code for a wider variety of systems to look more like the design. Examples will illustrate what intentional technology can be, and explore the unique role that aspect-orientation has to play.

Aspects of Grid Computing

Satoshi Matsuoka, Global Scientific Information and
Computing Center, Tokyo Institute of Technology

"The Grid" is slated to become one of the major infrastructures for network computing. Indeed, there are very large and active national and international projects to not only build grids but to carry out significant research and development on grid middleware and applications. This keynote will introduce the current status quo of grid research, and will attempt to identify its aspects as well as prospects for AOP to play a role in its construction, since the current trend is to define a set of "grid services" as Web-based component models.

Talk

Use Cases and Aspects: Working Together

Ivar Jacobson,
Rational Software Corp.

Use cases have been adopted for requirements universally. Use cases start there and are translated into collaborations in analysis and design, and to test cases in test; this is the central idea behind use-case driven development. With use cases we can cut the system into use-case slices with elements from each life cycle model. Almost. It is just "almost true" because today the coding of a component or a class requires us to merge the code derived from several use cases so that the individual slices will be dissolved and not recognizable any more. The root problem is limitations in currently used languages.

AOP in general is "the missing link." It will allow us to slice the system cleanly, use case by use case over many models, to achieve separation of concerns all the way down to code. In fact, we will get use-case modules crosscutting many models and their artifacts. It will subsequently allow us to recompose or weave back these slices into a consistent whole: the deployed system. The result is a variant of AOSD, AOSD with use cases. And it is here to be harvested, now.

Papers

Papers A: Analysis and Design

Architectural Views of Aspects

Mika Katara, Tampere University of Technology
Shmuel Katz, The Technion

Modularization and Composition of Aspectual Requirements

Awais Rashid, Lancaster University
Ana Moreira, Universidade Nova de Lisboa
João Araújo, Universidade Nova de Lisboa

JAsCo: An Aspect-Oriented Approach Tailored for Component-Based Software Development

Davy Suvée, Vrije Universiteit Brussel
Wim Vanderperren, Vrije Universiteit Brussel

Papers B: Program Analysis

Static Analysis of Aspects

Damien Sereni, Oxford University Computing Laboratory
Oege de Moor, Oxford University Computing Laboratory

A Case for Statically Executable Advice: Checking the Law of Demeter with AspectJ

Karl Lieberherr, Northeastern University
David H. Lorenz, Northeastern University
Pengcheng Wu, Northeastern University

Back to the Future: A Retroactive Study of Aspect Evolution in Operating System Code

Yvonne Coady, University of British Columbia
Gregor Kiczales, University of British Columbia and Intentional Software Corp.

Papers C: Programming Languages (I)

Arranging Language Features for Pattern-Based Crosscuts

Kris Gybels, Vrije Universiteit Brussel
Johan Brichau, Vrije Universiteit Brussel

Aspect-Oriented Programming with Jiazzi

Sean McDirmid, University of Utah
Wilson C. Hsieh, University of Utah

Parametric Introductions

Stefan Hanenberg, University of Essen
Rainer Unland, University of Essen

Papers D: Dynamic Weaving

Conquering Aspects with Caesar

Mira Mezini, Darmstadt Technical University
Klaus Ostermann, Siemens AG

Just-in-Time Aspects

Andrei Popovici, Swiss Federal Institute of Technology Zurich
Gustavo Alonso, Swiss Federal Institute of Technology Zurich
Thomas Gross, Swiss Federal Institute of Technology Zurich

Web Cache Prefetching as an Aspect: Towards a Dynamic-Weaving-Based Solution

Marc Segura-Devillechaise, Ecole des Mines de Nantes/INRIA
Jean-Marc Menaud, Ecole des Mines de Nantes/INRIA
Gilles Muller, Ecole des Mines de Nantes/INRIA
Julia L. Lawall, DIKU University of Copenhagen

Papers E: Systems

Persistence as an Aspect

Awais Rashid, Lancaster University
Ruzanna Chitchyan, Lancaster University

Quantifying Aspects in Middleware Platforms

Charles Zhang, University of Toronto
H.A. Jacobsen, University of Toronto

Model-View-Controller and Object Teams: A Perfect Match of Paradigms

Matthias Veit, Fraunhofer FIRST
Stephan Herrmann, Technical University Berlin

Papers F: Programming Languages (II)

Aspects and Polymorphism in AspectJ

Erik Ernst, University of Aarhus
David H. Lorenz, Northeastern University

Pointcuts and Advice in Higher-Order Languages

David B. Tucker, Brown University
Shriram Krishnamurthi, Brown University

Strategic Programming Meets Adaptive Programming

Ralf Laemmel, Free University of Amsterdam
Eelco Visser, Utrecht University
Joost Visser, Software Improvement Group

Papers G: Practitioner Reports

Aspect-Oriented Profiler

Jonathan Davies, Cambridge University
Nick Huismans, Imperial College London
Rory Slaney, Edinburgh University
Sian Whiting, Imperial College London
Matthew Webster, IBM UK
Robert Berry, IBM UK

Performance analysis is motivated as an ideal domain for benefiting from the application of AO technology. The experience of a ten-week project to apply AO to the performance analysis domain is described. We show how all phases of a performance analysts' activities—initial profiling, problem identification, problem analysis, and solution exploration—were candidates for AO technology assistance, some being addressed with more success than others. A profiling workbench is described that leverages the capabilities of AspectJ, and delivers unique capabilities into the hands of developers exploring caching opportunities.

Using AspectJ to Eliminate Tangling Code in EAI Activities

Arno Schmidmeier, Sirius Software GmbH

Enterprise application integration (EAI) imposes various non-functional requirements on integration teams and application manufacturers, which are hard to separate with object-oriented languages and tools. This paper describes how the overall integration effort has been dramatically reduced by using AspectJ to integrate different Sirius EOS Service Monitors in a New Generation Operations Systems and Software-compliant EAI architecture realized with Vitria BusinessWare.

Applying AOP for Middleware Platform Independence

Ron Bodkin, New Aspects of Security
Adrian M. Colyer, IBM UK
Jim Hugunin, Palo Alto Research Center

This report discusses experiences applying AspectJ in a consulting project at IBM. The purpose of this project was to evaluate the suitability of AspectJ for modularizing crosscutting concerns in a middleware product line. This report describes and assesses the design approaches, tools integration, and cultural effect.

Papers H: Tools

Navigating and Querying Code without Getting Lost

Doug Janzen, University of British Columbia
Kris De Volder, University of British Columbia

Visual Separation of Concerns through Multidimensional Program Storage

Mark C. Chu-Carroll, IBM, T. J. Watson Research Center
James Wright, IBM, T. J. Watson Research Center
Annie T. T. Ying, University of British Columbia

Demonstrations

Demonstrations I: Aspects and Performance

Aspect-Oriented Profiler

Matthew Webster, IBM UK
Robert Berry, IBM UK

Performance measurement, analysis, and improvement are central activities in the software development life cycle. Ideally, performance considerations play an early role (e.g., at design time), as recommended in performance-oriented design methodologies. But most often, they factor into the later stages of the development process, and require the involvement of performance analysts wielding specialized profiling technology.

Further, most existing profilers focus on CPU time and program flow. Instead, we sought a solution to allow applications to be profiled on the basis of data flow, a key requirement for a common, but not well-addressed problem of caching-opportunity detection. We required a technique and environment to selectively gather information on certain methods, argument values, and return values, to conduct correlation analysis between these, and to couple that information with timing information. While some of this information could certainly be gathered manually (e.g., Java debuggers based on JVM debug interface are able to track arguments and return values), this is typically a single-step debugging operation. Manually inserted instrumentation (e.g., using System.out.println statements, or through the use of a logging or other API) is also possible. Alternately, specialized instrumentation can be developed based on byte-code modification techniques—interestingly this is where we began, but the flexibility of AOSD techniques quickly suggested a different approach.

One of the key benefits of AOSD and AspectJ in particular is the ability to identify specific methods within a Java program and add new logic in a non-invasive manner. Reflective mechanisms also allow the extraction of argument and return values at runtime. AspectJ is further exploited when prototyping caching opportunities. Any method identified as performing below its expected level of performance can be intercepted through the use of around advice. A simple caching scheme can then be deployed such as checking to see if the method's argument values have been used before and, if so, returning the same return value; if not, calling proceed and storing the obtained return value in the cache.

The profiler is an Eclipse-based plugin developed as part of an Extreme Blue project. The AspectJ plugin is used to build an aspect into the program being profiled. The aspect is generated by the profiler so a deep understanding of AOSD is not required by the user. The demonstration will show the identification, analysis, and solution to a performance problem in a Java program. At each stage, the exploitation of AOSD techniques will be highlighted.

AspectC++: Bringing Aspects into Deeply Embedded Devices

Olaf Spinczyk, University of Erlangen-Nürnberg
Andreas Gal, University of Erlangen-Nürnberg

AspectC++ is an AO language extension for C++. Its design was strongly influenced by the core concepts of AspectJ and, thus, can be seen as an AspectJ equivalent in the C++ world. With this demonstration, the AspectC++ developer team wants to increase the awareness of the project and its current state in the AOSD community. Furthermore, it should be demonstrated that AOP with C++ can be much easier to understand and much more powerful than approaches that are based on C++ template meta-programming. There are several domains in computer science and the IT industry where C/C++ still dominates Java. One of these domains is the area of small (so-called "deeply") embedded systems. This area is characterized by extreme constraints in memory and processing power. Most of these embedded systems are equipped with 8-bit microcontrollers and only a few kilobytes of RAM. However, in the year 2000, the segment of 4- and 8-bit microcontrollers had a market share of 80 percent of all produced units. With AspectC++, it is now possible to apply AOSD concepts even in such restricted, but very important, domains. The focus of the demonstration will therefore be the minimal resource consumption of the AspectC++ generated code and its runtime system.

This will be underlined by the presentation of a small embedded device equipped with meteorological sensors and an 8-bit microcontroller running AspectC++ code. The whole picture of the language and its implementation will be rounded up with a demonstration of the unique language features of AspectC++. One of these features is the concept of aspect-behavior contract, which helps to reduce the number of required tests after changes in aspect or component code. Depending on the state of the implementation in March, it is also planned to extend the code of a well-known open source project with an aspect during the demonstration. This should point out the standard and dialect conformance of the AspectC++ parser and proves that the implementation has reached a state where it can deal with real-world projects.

Demonstrations II: IDE Extensions

The AspectJ Development Tools Project: Developing with AspectJ in Eclipse

Adrian M. Colyer, IBM UK
Andy Clement, IBM UK
Matthew Webster, IBM UK

This demonstration will show how Eclipse can be used to develop AO software using AspectJ. We will walk through a set of a typical project development scenarios, showing how AspectJ can be applied and along the way illustrating the features of the latest AspectJ Development Tool plugin for Eclipse. The plugin and the Eclipse IDE are both made freely available so everything shown can be tried out on your own development projects too.

FEAT: A Tool for Locating, Describing, and Analyzing Concerns in Source Code

Martin Robillard, University of British Columbia
Gail Murphy, University of British Columbia

Developers working on existing programs repeatedly have to address concerns, or aspects, that are not well modularized in the source code comprising a system. In such cases, a developer has to first locate the implementation of the concern in the source code comprising the system, and then document the concern sufficiently to be able to understand it and perform the actual change task.

In this demonstration, we will present FEAT, a tool for locating, describing, and analyzing the code implementing a concern in a Java system. The demonstration will consist of using the tool to locate and analyze a set of concerns scattered in an existing code base. Specifically, we will show how, by visually navigating structural program dependencies through the tool's graphical interface, we can rapidly locate the code implementing a concern, and store the result as an abstract representation consisting of building blocks that are easy to manipulate and query. We will also show how the representation of the concerns supported by FEAT can be used to investigate the relationships between the captured concerns and the base code, and between the different concerns. Finally, we will show how this representation can be used to robustly keep track of the actual source code implementing the concern.

We argue that the FEAT tool supports AOSD by allowing users to easily produce and analyze descriptions of the actual code implementing concerns in existing systems. The novelty of our approach is to capture concerns using an abstract representation that can be mapped back to source code, instead of working directly at the level of program text. This way, developers can use the abstract representation as a support for managing the code in a concern, and can potentially use the representation as a basis from which to refactor the concern into an AO programming language.

FEAT Version 2 is implemented as a plugin for the Eclipse platform. It uses the compiled representation (bytecode) of programs to extract the structural relationships between different program elements, such as classes, methods, or fields. It uses IBM's Jikes Bytecode Toolkit to represent and manipulate Java classes at run-time.

Demonstrations III: Tool Infrastructure

Towards a Concern-Manipulation Environment: An Open, Extensible Environment for Aspect Tools

Peri Tarr, IBM, T. J. Watson Research Center
William Harrison, IBM, T. J. Watson Research Center
Harold Ossher, IBM, T. J. Watson Research Center
Vincent Kruskal, IBM, T.J. Watson Research Center
Andrew Clement, IBM UK
Adrian M. Colyer, IBM UK
John Hatcher, IBM UK

This demonstration will show early work towards a concern-manipulation environment (CME). The CME is envisioned as a set of open, extensible, reusable components upon which are built a suite of tools that support AOSD across the software life cycle. The CME represents the next stage of research and development on multi-dimensional separation of concerns and Hyper/J. An important goal, however, is to support multiple AOSD approaches. At present, a variety of aspect models exist, each with different benefits. The CME will permit the use of multiple models, to allow developers to leverage their respective benefits, and will aid in the development of, and experimentation with, new models.

As part of the CME work, we will provide an initial set of tools that support multiple models of AOSD and multiple artifacts (e.g., UML class diagrams, Java source, and Java class files). These tools will be integrated into the Eclipse environment.

The use of the tools will be demonstrated by running through an aspect development and evolution scenario, and showing how different aspect models can be used to develop software, and how these models can be used in an integrated manner; and how the standard Eclipse tools can be used, in conjunction with the CME tools, to perform AOSD.

We will also give AOSD tool developers a sense of how to build on the CME components when creating their own tools. For example, the concern-assembly toolkit provides common, low-level weaving support on artifacts of different kinds, and is suitable for use as a back-end in a variety of AOSD tools supporting composition or weaving.

JMangler: Load-Time Weaving for Java Class Files

Günter Kniesel, University of Bonn
Michael Austermann, SCOOP Software GmbH

AOSD improves separation of concerns by making it possible to express crosscutting concerns of a system modularly. However, modular expression of a concern requires techniques to "weave" the related code back into the code of all the affected classes.

JMangler is a freely available framework for load-time transformation of compiled Java programs that provides a comprehensive infrastructure for load-time weaving. This means that an AOSD system can translate its aspects to JMangler transformer components and let them be applied at load time. Alternatively, a knowledgeable programmer can use Jmangler directly to express aspect-like crosscutting transformations of arbitrary application classes. Yet another option is to inject code at load time that enables run-time weaving.

Load-time weaving has many advantages, the smallest being that it requires no source code and can hence be applied to third-party libraries. More importantly, it provides the guarantee that transformations will be applied to every class that will be executed at run-time, even if the class is created dynamically or loaded from some possibly remote host. Last, but not least, load-time weaving inherently processes only classes relevant to the running application and applies to them only the adaptations required in that context. It can therefore prevent static proliferation of adapted program versions that might never be used.

Unlike simple bytecode transformation libraries, JMangler provides a complete solution for hooking into the class-loading process. It does so in a JVM- and class-loader-independent way, which works also for classes that employ their own custom class loader. Therefore, it can transform any application classes and can be used in environments like application servers, which make heavy use of custom class loaders.

In addition to its general applicability, Jmangler provides another unique feature. It is the only approach for load-time adaptation that provides a partial solution to the problem of aspect interference. For a certain class of transformations, it can guarantee that their joint use will not lead to undesired effects (interferences) even if

the transformations have been developed independently, unaware of each other.

The demo will consist of two parts that will be presented piecemeal, in alternation: the introduction of the base concepts and the demonstration of their practical use. The shown examples will include, among others, a code-coverage tool developed with JMangler. This is an application that requires transformations at the level of individual lines of code or individual statements, hence at a finer granularity than expressible in all known high-level AOSD languages and systems.

Demonstrations IV: From Design to Code

Aspect-Oriented Software Development with Codagen Architect

Mario Cardinal, Codagen Technologies Corp.

This demonstration will present Codagen Architect, a "model-driven architecture" tool that enables AOSD by allowing the transformation of UML models directly into working software code. Codagen Architect enables software development teams to promote AOSD by abstracting aspects such as architectural issues, encapsulating aspects in transformation templates, and weaving aspects across business objects during the generation process.

Codagen Architect has existed since 1999, and version 3.0 generates Java, C#, C++, and Visual Basic code. Codagen Architect is one of the first model-driven-architecture-compliant tools that enables AOSD.

Codagen Architect implementation techniques are based on UML metaprogramming. The audience will learn how easy it is to abstract aspects such as debugging issues inside Codagen transformation templates and how, using a UML model as input, they can easily weave the debugging code inside business objects.

From Aspect-Oriented Design with Concepts to Aspect-Oriented Programming with Composition Filters

Lodewijk Bergmans, University of Twente
Dennis Wagelaar, Vrije Universiteit Brussel

Composition filters are an extension to the object-oriented model that address a number of modeling obstacles that conventional object-oriented techniques cannot address or can only solve with poorly maintainable designs. The composition-filters approach can be classified as an AO approach that integrates aspects and classes, retains strong encapsulation, and supports composability. ComposeJ is a tool that takes Java classes and (separate) composition filters specifications, and transforms the Java classes so that they implement the behavior as specified in the composition filters specification.

Like object-oriented programming, the benefits of AOP can only be exploited through appropriate design methods and tool support. CoCompose is a design tool that addresses this. The main characteristic of CoCompose is that it is a visual, concern-oriented tool (i.e., supports the modeling of software as independent, possibly crosscutting, concerns) that allows for recursive definition and reuse of concerns. Concerns in the design phase may be similar to entities or classes, or to single operations, or even to complete design pattern templates. A single concern may in fact have several implementation forms; during code generation, the best possible (interoperating) combination of forms is selected. Thus, CoCompose works as a design tool that can generate the best possible program from a concern-based design (if the concerns include one or more implementations).

During the demonstration, we will show how the design of a particular example problem can be approached in a concern-oriented manner and modeled with CoCompose. We will demonstrate how CoCompose can generate implementations in several programming languages (especially Java and composition filters). One of the important points of this demonstration is to argue that, even with the ability to generate code, the composability of the target language is important in order to retain the structure of the design, and hence the ability to revise and extend the design in an incremental manner.

In particular, we will look at generated composition filters code, illustrate that it is structurally close(r) to the design, and explain the basic composition filter mechanism. We will demonstrate our tool Compose/J that translates combined composition filters/Java code into pure Java code. One of the interesting features of Compose/J that we will highlight (and demonstrate) is the ability of generating optimized (i.e., inlined) code from a declarative specification.

Student Research Extravaganza

Goals

- Provide a venue for students to vet research ideas and directions with experts in the field.
- Expose students to possible research directions.
- Allow students to intermingle and get to know each other.

Format

- Formal Portion: After a brief introduction from the organizers, two experts in the field each will speak for five minutes about where they see the field heading and interesting problems likely to be encountered along the way. We will also introduce other experts attending.
- Semi-Formal Portion: Various parts of the room will be designated as focusing on different topics such as programming languages or software engineering. Student posters will be displayed in the appropriate part of the room. Students will be invited to present their posters. One or two experts in the field will move around the relevant area and comment on work.
- Informal Portion: Mingling around posters will continue, giving students a chance to talk to each other and the roaming experts.

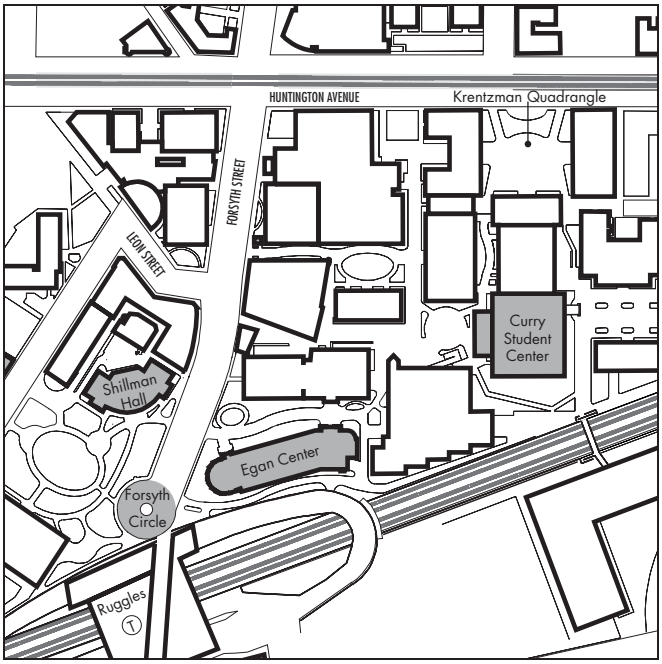
Students should put up posters prior to the start of the event. Food will be available.

Attendees

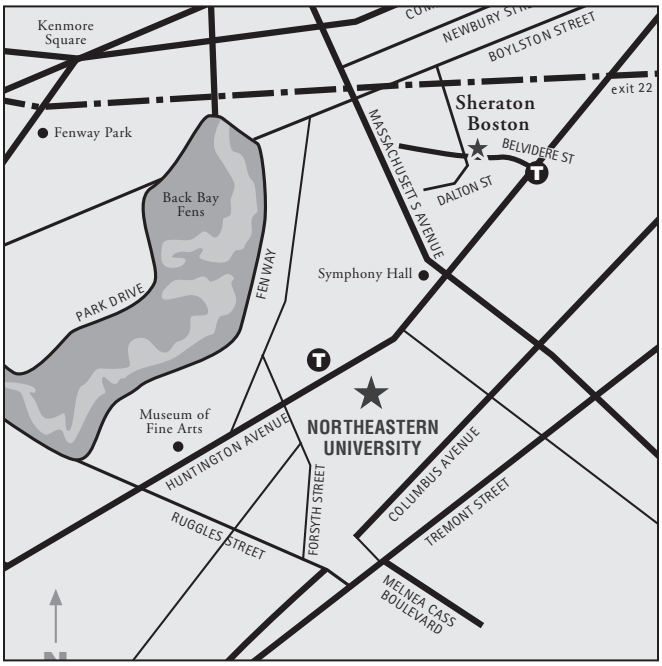
The event is only open to students and participating experts to ensure that the students have ample opportunities to discuss their work with these experts.

Finding Your Way

Northeastern University Campus Map



Boston Area Map



To get to the conference from the Sheraton Boston Hotel:

On foot:
Take Dalton Street south to Belvidere Street. Go left on Belvidere and take it 250 yards to Huntington Avenue (The Avenue of the Arts). Go right on Huntington and take it approximately two-thirds of a mile to Forsyth Street. Take a left on Forsyth. After one block you will see Shillman Hall on the right and Egan Research Center on the left.

By subway (known as the "T"):
Take Dalton Street south to Belvidere Street. Go left on Belvidere and take it 250 yards to Huntington Avenue (The Avenue of the Arts). Just to the left you will find an entrance to the Prudential station on the "E" branch of the Green Line subway. Take the Green Line two stops in the direction of Heath Street (outbound). Get off at the Northeastern stop. Continue a half block down Huntington Avenue (in the direction the train was going) to Forsyth Street. Take a left on Forsyth. After one block you will see Shillman Hall on the right and Egan Research Center on the left.

IBM Research