July 6th - 10th 2009, Genova, Italy

23rd European Conference on Object-Oriented Programming

ECOOP2009

http://2009.ecoop.org

Conference Guide

ECOOP 2009

Welcome to ECOOP 2009 in Genova!

ECOOP 2009's Organizing Committee is pleased to welcome you to Genova, for the 23rd European Conference on Object-Oriented Programming. Since it was first held, in Paris in 1987, ECOOP has traveled across Europe and is now returning to Italy for the second time after previously being held in Bologna (1994). This year's edition continues the tradition of being a premier international forum to discuss and advance a broad range of topics woven together by the common thread of object technology. It offers a well-integrated collage of events, including outstanding invited speakers, carefully refereed technical papers, topic-focused workshops and a summer school offering selected tutorials. The conference venue is Palazzo Ducale, the Doge historical residence, in the heart of Genova. In such a remarkable context, we trust you will find the program interesting and enjoyable.

The main reason for ECOOP's ongoing success is its excellent technical program, put together each year from a large number of high-quality submissions. This year's Program Chair is Sophia Drossopoulou, with whom it has been a great pleasure to cooperate. We are delighted to announce three outstanding keynote talks. The conference keynotes will be given by Simon Peyton Jones and Cliff Click. The last keynote talk will be given by David Ungar, the 2009 winner of the AITO Dahl-Nygaard Award. We wish to thank all those who are organizing workshops, giving tutorials, presenting demonstrations, offering posters and running the impromptu birds-of-a-feather sessions for their significant efforts, all of which contribute to the richness of the ECOOP tapestry. ECOOP 2009 is being organized by the Department of Computer and Information Sciences of the University of Genova in cooperation with the Department of Informatics and Communication of the University of Milano. It is being held under the aegis of AITO, the Association Internationale pour les Technologies Objets, in cooperation with ACM, SIGPLAN, and SIGSOFT. The conference would not have been possible without the financial support of numerous sponsors. Special thanks go to ERCIM, IBM Research, Google, Microsoft Research, and Yahoo! Research.

Welcome to Genova and enjoy ECOOP 2009!

Giovanna Guerrini and Elena Zucca
on behalf of the Organizing Committee

Cover by Daniela Peghini, Graphic Design by Walter Cazzola

Contents

ECOOP'09 Conference at Glance

Monday, July 6th

	Workshops	Summer School
8:00-9:00	Registration	
9:00-10:30	W1, W2, W3, W4, W5	
10:30-11:00	Coffee Break	
11:00-12:30	W1, W2, W3, W4, W5	
12:30-14:00	Lunch	
14:00-15:30	W1, W2, W3, W4, W5	
15:30-16:00	Coffee Break	
16:00-17:30	W1, W2, W3, W4, W5	

Tuesday, July 7th

	Workshops	Summer School
8:00-9:00	Registration	
9:00-10:30	W6, W7, W8, W9, W10, W11, W12	
10:30-11:00	Coffee Break	
11:00-12:30	W6, W7, W8, W9, W10, W11, W12	
12:30-14:00	Lunch	
14:00-15:30	W6, W7, W9, W10, W11, W12	PyPy
15:30-16:00	Coffee Break	
16:00-17:30	W6, W7, W9, W10, W11, W12	

Wednesday, July 8th

	Technical Papers	Summer School
8:00-9:00	Registration	
9:00-9:30	Welcome	
9:30-10:30	Keynote: Simon Peyton-Jones	
10:30-11:00	Coffee Break	
11:00-12:30	Types, Frameworks and Modelling	Project Fortress
12:30-14:00	Lunch	
14:00-15:30	Aliasing and Transactions	VeriFast
15:30-16:00	Coffee Break	
16:00-17:30	Access Control and Verification	Flexible Task Graphs
17:30-18:30	Poster & Demo Session	

Thursday, July 9th

	Technical Papers	Summer School
9:00-9:30	Awarding of Dahl-Nygaard Prize	
9:30-10:30	Talk of Dahl-Nygaard laureate David Ungar	
10:30-11:00	Coffee Break	
11:00-12:30	Modularity	Crystal-izing
12:30-14:00	Lunch	
14:00-15:30	Mining and Extracting	Spec#
15:30-16:00	Coffee Break	
16:00-17:30	Refactoring	Object Teams
17:30-18:30	Poster & Demo Session	

Friday, July 10th

	Technical Papers	Summer School
9:00-10:00	Keynote: Cliff Click	
10:30-11:00	Coffee Break	
11:00-12:30	Concurrency, Exceptions and Initialization	
12:30-14:00	Lunch	
14:00-15:30	Concurrency and Distribution	
15:30-16:00	Closing	

AITO Dahl-Nygaard Prize

Thursday, July 9th, 9:00-9:30

ECOOP 2009 is delighted to host the fifth AITO Dahl-Nygaard Prize Awards.

The AITO Ole-Johan Dahl and Kristen Nygaard prize is awarded annually to two individuals that have made significant technical contributions to the field of Object-Orientation. The work should be in the spirit of the pioneer conceptual and/or implementation work of Dahl and Nygaard which shaped our present view of programming and modeling, now known as Object-Orientation. The prize is presented each year at the ECOOP conference. The prize consists of two awards given to a senior and to a junior professional; while the prizes are usually awarded annually, there may be exceptional occasions when one or both are left unawarded. The senior professional should have made a significant long-term contribution to the field in research or engineering. The junior professional should have made a promising contribution to the field through a paper, a thesis or a prototype implementation. A Prize Committee of three persons nominated annually by the AITO General Assembly recommends the award winners to the AITO executive committee. Every year AITO will solicit proposals until September 30th. Such notice shall be placed on the AITO/ECOOP websites and also emailed to the ECOOP mailing list and other organizations such as OOPSLA and AOSD as appropriate. The Prize Committee will propose two names to the AITO executive committee, not later than December 31st of that year. Proposals may be submitted by anyone in the community using by sending an email to `dahl-nygaard@aito.org`. The decision of the committee will be made public and official before the end of January and published on the AITO web site. The prizes will be officially presented at the ECOOP conference of the following year. The winners may be asked to give a talk on this occasion. Both recipients will be invited by AITO to ECOOP (travel expenses and conference registration will be paid by AITO). In addition to the presentation of the prize, the junior winner will receive a cash award roughly corresponding to the prize of a portable computer (approximately 2000 Euro in the year 2004).

AITO is very proud to announce the winner of the 2009 Dahl-Nygaard Prize: David Ungar (IBM Research) for his contribution to the creation of the Self programming language. David Ungar will be the keynote speakers on Thursday morning.

Thursday, July 9th, 9:30-10:30

Self and Self: Whys and Wherefores

David Ungar (IBM Research, USA)

Abstract. Generational garbage collection, prototype-based languages, dynamic optimization, cartoon animation for legibility, all tremendous fun, none done alone. What were they? How did they happen? Why did they matter? Looking back, what

is worth learning about these experiences beyond the technical innovations? Combining hindsight with others' wisdom, it is possible to abstract some thoughts that may be useful in other situations: when (not) to listen to wise council; whom to follow into the cafeteria at lunch time; the benefit of striking a balance between one's own vision and those of ones collaborators; which chance events might alter one's course; and how one's best work can sometimes arise from things that, on the surface, have nothing to do with work at all. At a deeper level still, the notion that values, principles, and practices arise in that particular order serves to unify the work and the experiences, and perhaps points the way forward as we all strive to invent the future.

Speaker's Bio. David Ungar has long been fascinated by programming paradigms that can change the way people think, novel implementation techniques that make new languages feasible, and user interfaces that vanish. With Dr. Randall B. Smith at PARC, he designed a simple yet powerful prototype-based object-oriented programming language called "Self." As an Assistant Professor at Stanford, David and his students developed new compilation techniques and heap structures for pure object-oriented programming languages. Rejoining Dr. Smith at Sun Microsystems Laboratories, David co-led a project to create a complete programming environment for Self. The implementation techniques developed for Self have been harnessed for Sun's HotSpot Java's Virtual Machine. David's Klein project explored metacircularity in pursuit of simpler, more malleable high-performance virtual machines and better development environments for them.

David's doctoral research was performed at the University of California at Berkeley with David Patterson, and concerned the development of a RISC for Smalltalk. The dissertation was published by the MIT press as an ACM Distinguished Dissertation. It introduced a fast automatic storage reclamation algorithm, Generation Scavenging, which has since influenced many production systems, and isolated those architectural features that significantly improved performance.

David Ungar is an ACM Distinguished Engineer, and three of his papers have been recognized as having been among the most influential in their respective fields: one on the Self language, one on the application of cartoon animation techniques to user interfaces, and one on generational garbage collection.

Since 2007, David has been privileged to be part of IBM Research, where he has added a facility for collaboration to a performance-analysis system (Tuning Fork), and where, in collaboration with Sam Adams, he investigates new programming paradigms for manycore architectures.

Previous AITO Dahl-Nygaard Prizes

	Senior Prize	**Junior Prize**
2005 (Glasgow)	Bertrand Meyer	Gail Murphy
2006 (Nantes)	E. Gamma, R. Helm, R. Johnson, and J. Vlissides	
2007 (Berlin)	Luca Cardelli	Jonathan Aldrich
2008 (Paphos)	Akinori Yonezawa	Wolfgang De Meuter

Workshops at Bristol Palace

Monday, July 6th, 9:00-12:30, 14:00-17:30

W1 ELW — European Lisp Workshop (Sala Colombo)
Didier Verna (EPITA, Paris, France), Charlotte Herzeel (Vrije Universiteit, Brussel, Belgium), Robert Strandh (LaBRI, University of Bordeaux I, France), Christophe Rhodes (University of London, UK) and Hans Hübner (Software Developer, Berlin, Germany).

W2 ICOOOLPS — Implementation, Compilation, Optimization of Object-Oriented Languages, Programs and Systems (Sala Michelangelo)
Ian Rogers (University of Manchester, UK) and Olivier Zendra (INRIA, France).

W3 STOP — Script to Program Evolution (Sala Marconi)
Nate Nystrom (IBM T.J. Watson Research Center, USA), Jan Vitek (Purdue University, USA) and Tobias Wrigstad (Purdue University, USA).

W4 FTfJP — Formal Techniques for Java-like Programs (Sala Leonardo)
Anindya Banerjee (IMDEA Software, Spain), Sophia Drossopoulou (Imperial College, UK), Susan Eisenbach (Imperial College, UK), Gary T. Leavens (University of Central Florida, USA), Peter MÏler (ETH Zřich, Switzerland), Arnd Poetzsch-Heffter (University of Kaiserslautern, Germany) and Erik Poll (Radboud University Nijmegen, Netherlands).

W5 Doctoral Symposium (Sala Bellini)
Stephen Nelson (Victoria University of Wellington, New Zealand).

Tuesday, July 7th, 9:00-12:30, 14:00-17:30

W6 COP — Context-Oriented Programming (Sala Michelangelo)
Pascal Costanza (Vrije Universiteit Brussel, Belgium), Richard P. Gabriel (IBM Research, USA), Robert Hirschfeld (Hasso-Plattner-Institut, Germany) and Jorge Vallejos (Vrije Universiteit Brussel, Belgium).

W7 DO21 — Distributed Objects for the Twenty-First Century (Sala Leonardo)
Tom Van Cutsem (Vrije Universiteit Brussel, Belgium), Jorge Fox (Trinity College Dublin, Ireland), Ole Lehrmann Madsen (Aarhus University and Alexandra Institute, Denmark), Eric Jul (University of Copenhagen, Denmark) and Gilad Bracha (Ministry of Truth, USA).

W8 XOODB — The marriage of XML and Object-Oriented Database Technologies and their Future (Sala Bellini) **Morning Only**
Dario Colazzo (University of Paris Sud, France), Marco Mesiti (DICo, Università di Milano, Italy), Carlo Sartiani (University of Basilicata, Italy) and Emmanuel Waller (University of Paris Sud, France).

W9 POOSC — Parallel/High-Performance Object-Oriented Scientific Computing (Sala Colombo)
Wolfgang Bangerth (Texas A&M University, USA), Kei Davis (Los Alamos National Laboratory, USA), Peter Gottschling (Technische Universität Dresden, Germany), René Heinzl (Technische Universität Wien, Austria), Bernd Mohr (Juelich Supercomputing Centre, Germany), Jörg Nolte (Brandenburg University of Technology, Germany), Laurent Plagne (EDF, France), and Jörg Striegnitz (University Of Applied Sciences, Germany).

W10 RAOOL — Relationships and Associations in Object-Oriented Languages (Sala Paganini B)
Stephanie Balzer (ETH Zürich, Switzerland), Gavin Bierman (Microsoft Research, UK), Stephen Nelson (Victoria University of Wellington, New Zealand) and Frank Tip (IBM T.J. Watson Research Center, USA).

W11 IWACO — International Workshop on Aliasing, Confinement and Ownership in Object-Oriented Programming (Sala Mazzini)
Dave Clarke (K.U. Leuven, Belgium), Sophia Drossopoulou (Imperial College, UK), James Noble (Victoria University of Wellington, New Zealand) and Tobias Wrigstad (Purdue University, USA).

W12 RAM-SE — Reflection, AOP and Meta-Data for Software Evolution (Sala Marconi)
Walter Cazzola (DICo, Università di Milano, Italy), Shigeru Chiba (Tokyo Institute of Technology, Japan), Manuel Oriol (University of York, UK) and Gunter Saake (Otto-von-Guericke-Universität Magdeburg, Germany).

ECOOP 2009 Summer School

ECOOP 2009 offers 7 Summer School sessions. The first session is on Tuesday, in parallel with workshops, whereas all other sessions are on Wednesday and Thursday, in parallel with the technical paper sessions of the main conference. Attendance at the Summer School sessions is included with registration to the main conference; attendance at the first session is also inlcuded with registration to workshops only. The Summer School sessions will be offered on a first-come, first-served basis: if you wish to attend a particular session, make sure you are registered, and get to the Summer School room early!

Tuesday, July 7th (14:00-15:30, Bristol, Sala Bellini)

Writing Interpreters for Dynamic Languages Using PyPy (and Getting Them Fast)
Antonio Cuni (Università di Genova, Italy),
Carl Friedrich Bolz and Armin Rigo (Heinrich-Heine-Universität Düsseldorf, Germany)

Abstract. We propose to give a tutorial on how to use the PyPy project to write interpreters for dynamic languages. PyPy supports compiling such interpreters to various target environments (like .NET or C/Posix) and automatically generating a Just-In-Time compiler for them. PyPy is thus an emerging tool to help implement fast, portable multi-platform interpreters with less effort.

Antonio Cuni is a Phd student at the University of Genova, Italy. His main area of interests is about the implementation of dynamic languages, particularly on top of virtual machines like .NET or the JVM. He has been one of the core developers of PyPy since 2006, working in particular on the backends for CLI and JVM and on the JIT compiler generator.

Carl Friedrich Bolz is a PhD student at the University of Düsseldorf, Germany. He is mostly interested in the efficient implementation of dynamic programming languages. In 2005 he started to contribute to the PyPy project and subsequently became one of the core developers. He has been involved in nearly all areas of PyPy development, from garbage collectors to stackless implementation. Additionally he is very interested in partial evaluation, particularly at runtime.

Armin Rigo was born in 1976 in Lausanne, Switzerland. He is a researcher at the Heinrich-Heine Universität Düsseldorf, previously at the University of Southampton. He obtained his Ph.D. in Logic and Set Theory at the Free University of Brussels. He is the main author of several commercial, open source and research programs. He developed with the Psyco project novel techniques for efficient interpretation of dynamic programming languages. From there on, he was involved in the core of PyPy with the same goal of making efficient interpreters, but writing them in a high-level and straightforward style.

Wednesday, July 8th (11:00-12:30, Palazzo Ducale, Sala «Camino»)

Project Fortress: A Multicore Language for Scientists and Engineers
Sukyoung Ryu and Jan-Willem Maessen (Sun Microsystems, USA)

Abstract. The computing world is currently undergoing dramatic changes. More powerful computers have brought high-performance computing (HPC), an endeavor historically relegated to national labs and government institutions, to the mainstream. At the same time, microchip manufacturers are designing new chips that contain increasing numbers of cores on a single chip to improve performance. Yet unfortunately, modern programming languages are ill-equipped for these changes. Most languages do not directly support any notion of parallelism and even languages with support for parallelism, such as the JavaTM Programming Language, support only course-grained notions of parallelism, best suited for tasks such as networking and GUI programming. Fortress is a new programming language designed for HPC with high programmability. It provides mathematical syntax to enable scientists and engineers to write programs in a notation they are accustomed to. It also provides built-in support for parallel programming. Fortress is designed for growth by community participation and development and its syntax and semantics have been formally designed and specified. The tutorial introduces Project Fortress, an open-source project with a reference implementation of the Fortress programming language. Anyone who works on scientific programming, parallel programming, quality-critical software development, or open-source projects will find this tutorial of interest.

Sukyoung Ryu is a Member of Technical Staff in Sun Microsystems Laboratories, where she works on formally designing and developing the Fortress programming language. Before that, she was a Research Associate in Computer Science at Harvard, where she worked on the Debugging Everywhere project. She received her Ph.D. (2001), M.S. (1996), and B.S. (1995) in Computer Science from Korea Advanced Institute of Science and Technology. Her most recent research focuses on developing language features that are both useful in practice and proven to be sound. She is leading the effort to construct the core calculi of the Fortress language, to improve the Fortress prose specification, and to build a full-fledged static front-end for Fortress that runs entirely on the JVM.

Jan-Willem Maessen has been part of Project Fortress since its inception in 2002, and is presently the primary maintainer of the libraries for the Fortress programming language. He has been heavily involved in the design of the Fortress language and in its implementation, and drafted the original specifications of the parallel portions of the language. His many interests include language design, memory consistency models, concurrent algorithms, compilation, semantics, and architecture. Prior to joining Sun, Jan developed Eager Haskell, a Haskell implementation that executes programs using resource-bounded eager evaluation rather than lazy evaluation. He also contributed heavily to the compiler and libraries for pH, an implicitly-parallel programming language.

Wednesday, July 8th (14:00-15:30, Palazzo Ducale, Sala «Camino»)

VeriFast: A Simple, Sound Verifier for Rich Separation Logic Contracts

Bart Jacobs and Jan Smans (Katholieke Universiteit Leuven, Belgium)

Abstract. VeriFast is a program verification tool for single-threaded and multi-threaded C and Java programs that we are developing. Successful verification guarantees absence of hard to catch programming errors such as data races and memory leaks, as well as compliance with rich preconditions and postconditions specified by the programmer in source code annotations in a form of separation logic. To enable rich specifications, the programmer may define inductive datatypes, primitive recursive pure functions over these datatypes, and abstract separation logic predicates. To enable verification of these rich specifications, the programmer may write lemma routines, i.e., routines that serve only as proofs that their precondition implies their postcondition. The verifier checks that lemma routines terminate and do not have side-effects. The tool takes an annotated program as input and reports success or failure without further interaction. The tool verifies each function/method separately, by symbolic execution, where values are represented as terms of first-order logic with a set of equality and inequality constraints, and memory is represented as a separate conjunction of concrete and abstract elements. Since neither VeriFast itself nor the underlying SMT solver need to do any significant search, verification time is predictable and low. We have written and verified various example programs, including data-race-freedom of a small multi-threaded chat server, and full functional correctness of a linked list implementation with iterators, a binary tree implementation, and an implementation of the Composite pattern. We have developed an IDE that allows the user to step through a failed symbolic execution path, inspecting the symbolic values of locals and the symbolic heap at each point.

Bart Jacobs is a post-doctoral researcher and **Jan Smans** is a final-year PhD student at the DistriNet research group of the Department of Computer Science at the Katholieke Universiteit Leuven, Belgium. Both presenters have extensive experience in program verification and contributed to various verification research projects, including Spec#, Chalice, and VCC at Microsoft Research. Bart Jacobs presented a tutorial on Spec# at FM 2005.

Wednesday, July 8th (16:00-17:30, Palazzo Ducale, Sala «Camino»)

Flexible Task Graphs: A Framework for Experimental Real Time Programming in Java

Joshua Auerbach (IBM Research, USA)

Abstract. This ECOOP 2009 tutorial will show how to use the newly available open-source Flexible Task Graphs (Flexotask) framework to develop real-time Java applications and as a test-bed for new ideas in real-time scheduling, and model-driven real-time programming. Participants interested in real-time will gain a tool

that will be immediately useful in their own research. Participants who are merely curious about the emerging real-time support in Java will get a survey of the field (the Real Time Specification For Java, real-time garbage collection) followed by an in-depth look at "restricted thread programming models" for Java (Eventrons, Reflexes, Exotasks, StreamFlex, and especially Flexotask, which was designed to unify and subsume the others). The tutorial will be presented by Joshua Auerbach who will take overall responsibility. If available, one or more of Jan Vitek, Jesper Honig Spring, and/or David Bacon may assist.

Joshua Auerbach has been a Research Staff Member at the IBM Watson Research Center since 1983, contributing in the areas of protocol conversion, distributed computing, virtualization, and programming languages. Since 2005 he has worked on Real Time Java, contributing to IBM's WebSphere Real Time product and the experimental Eventron, Exotask, and Flexotask models. He holds a Ph.D from Yale University.

Thursday, July 9th (11:00-12:30, Palazzo Ducale, Sala «Camino»)

Crystal-izing Sophisticated Code Analyses

Ciera Jaspan, Kevin Bierhoff, and Jonathan Aldrich (Carnegie Mellon University)

Abstract. In recent years we have seen many researchers and practitioners build tools which statically analyze object-oriented programs. These code analyses have become incredibly sophisticated: they are often based on complex algorithms and type systems, require a dataflow analysis, use annotations, summaries, or other mechanisms to achieve precision in reasoning about method calls, and take advantage of conditional tests in the program or are even path-sensitive. These techniques typically interact with the language being analyzed in nontrivial ways. This makes it almost impossible to turn analyses into tools for widely used programming languages without support from an Integrated Development Environment (IDE) such as Eclipse or a code analysis frameworks like FindBugs. But the effort for the analysis writer is still high: IDEs facilitate interaction with the developer of the program being analyzed, but they provide no support for performing sophisticated analyses. On the other hand, code analysis frameworks typically do provide support for performing dataflow analyses, but interacting with the programmer becomes difficult. Additionally, the intricacies of dealing with a real language are generally left to the analysis writer, and popular techniques such as taking advantage of conditionals are not facilitated by many analysis frameworks. This tutorial presents our experience in writing static code analyses for Java using the Crystal analysis framework, which we developed and refined over the last four years. Crystal is itself a plugin to the Eclipse IDE and builds on Eclipse's well-tested AST. Thus its code base is relatively small and allows analysis writers to leverage the Eclipse platform where needed. Originally developed for teaching dataflow analysis techniques in a classroom setting, Crystal is built for simplicity. The interfaces are based upon the theoretical principles of static

analysis and make it simple to transition from theory to prototype. In our experience, the framework allows rapid development of even sophisticated static analyses.

Ciera Jaspan is a graduate student in the Software Engineering Ph.D. program at Carnegie Mellon University. Her research interests include software frameworks, program analysis, cost-effective development tools, and software engineering education. She received the 2008 John Vlissides award for her continuing thesis work on a static analysis to detect misuse of software frameworks. Jaspan earned a bachelor's degree in software engineering from Cal Poly, San Luis Obispo.

Kevin Bierhoff recently graduated from Carnegie Mellon University with a Ph.D. in software engineering. His research contributions center around practical formalization and enforcement of API protocols in the context of object-oriented programming languages. Kevin co-developed the Crystal static analysis framework and used it for both teaching and his own research.

Jonathan Aldrich is an Associate Professor in the School of Computer Science at Carnegie Mellon University. Aldrich's research contributions include techniques for verifying object and component interaction protocols, modular reasoning techniques for aspects and stateful programs, and new object-oriented language models. For his work on verifying software architectures, Aldrich received a 2006 NSF CAREER award and the 2007 Dahl-Nygaard Junior Prize, given annually for a significant technical contribution to object-oriented programming. His graduate course in program analysis was the original motivation for creating the Crystal analysis framework. Aldrich earned a bachelor's degree in computer science from Caltech and a doctorate in computer science from the University of Washington.

Thursday, July 9th (14:00-15:30, Palazzo Ducale, Sala «Camino»)

Program verification using the Spec# Programming System

K. Rustan M. Leino (Microsoft Research, USA) and Rosemary Monahan (National University of Ireland, Ireland)

Abstract. The Spec# Programming System consists of the Spec# programming language which is an extension of C#, the Spec# compiler which is integrated into the Microsoft Visual Studio development environment and the Spec# static program verifier which translates Spec# programs into logical verification conditions. The result is an automatic verification environment which establishes the correctness of a program before allowing it to be executed. In addition to the run-time checking traditionally found in the design by contract approach, Spec# offers the possibility to verify contracts statically. The goal of this analysis is to help programmers to detect errors more quickly than they might be found by traditional methods, hence providing an opportunity to correct the errors when they are still relatively cheap to fix. A unique feature of the Spec# Programming System is its guarantee of maintaining invariants in object-oriented programs in the presence of callbacks, threads and

interobject relationships. In addition, the Spec# Programming System has been extended so that programs that involve the mathematical domains that are so familiar to textbook examples (such as sum, count, product, min, and max) may be verified. In this tutorial, we give an overview of the Spec# Programming System. We focus on writing programs and their specifications, so that users develop experience in verifying C# programs. Maintaining object invariants in the presence of callbacks, threads, and inter-object relationships will be presented with examples, exploring the more difficult aspects of program verification in an object-oriented environment.

K. Rustan M. Leino is a Principal Researcher at Microsoft Research, where his research centres on programming tools. As part of the Research in Software Engineering (RiSE) group, he leads the Spec# project, focusing on the design and implementation of the Spec# programming language and its static program verifier.

Rosemary Monahan is Lecturer at the National University of Ireland, Maynooth (NUIM), where she lectures on topics such as program verification, program language semantics and computer programming. Rosemary's research, as part of the Principles of Programming (PoP) group, is concerned with the static and dynamic analysis of object-oriented programs.

In recent collaborations, Rosemary and Rustan have extended the Spec# programming system to support typical classroom exercises. At ECOOP 2009, Rosemary will present these and other features, demonstrating how to use the Spec# Programming System for the specification and verification of object-oriented programs.

Thursday, July 9th (16:00-17:30, Palazzo Ducale, Sala «Camino»)

Object Teams: Programming with Contextual Roles
Stephan Herrmann (Technische Universität Berlin, Germany)

Abstract. From the early days of object orientation it is well-known that inheritance between classes and delegation between objects provide similar capabilities. While each approach has its specific strengths and limitations, only few programming languages support a combination of both. The concept of roles provides an excellent intuition for objects that are instances of a class and also delegate to a base or parent object. Early attempts of integrating the role concept into an object oriented language add accidental complexity to programs because one important property of roles had been neglected: roles depend on context. More recent languages thus reify the context in which roles live as "teams" (ObjectTeams/Java (OT/J)) or "institutions" (powerJava) etc. Of these languages OT/J is an excellent representative due to the maturity of the language and its tools. Steimann has identified 15 criteria from what various authors had considered relevant for the role concept. OT/J supports all 15 criteria. OT/J has been thoroughly integrated with Java, accounting for all relevant features of Java up-to version 6, as documented in the comprehensive language definition. Furthermore, team inheritance treats roles as virtual classes and fully implements the concept of family polymorphism, with additional options towards more

strictness (in terms of object confinement) and towards more flexibility (in terms of migration between teams). For typesafe substitution of roles and bases, OT/J introduces translation polymorphism, realized by the implicit translations "lifting" and "lowering". With smart lifting a powerful mechanism is provided for mapping an inheritance hierarchy of base classes to a similar, but potentially different, hierarchy of role classes. Finally, several mechanisms for activating and deactivating a team support the notion that roles can be interpreted as a specific form of disciplined aspects, where aspect bindings utilize and extend the power of Java generics to overcome typing issues that have been identified in AspectJ. Summarizing, OT/J is a modern extension of the object oriented paradigm, borrowing from several approved concepts, combining all into a mature language. This language is supported by an industrial strength development environment based on and deeply integrated with the Eclipse JDT.

Stephan Herrmann received his Ph.D. at Technische Universität Berlin in 2002 for his work on applying new techniques for separation of concerns to the development of a multi-view software engineering environment. Since then his focus is on developing the language ObjectTeams/Java (OT/J) and the full-featured Eclipse-based Object Teams Development Tooling. Much of his practical experience with OT/J stems from the fact that the OTDT itself is written in OT/J. He has a long track-record of teaching OT/J in classroom, conference and workshop presentations as well as various demos and tutorials on OT/J and its tools. Conference presentations include OOPSLA, AOSD and also EclipseCon and JAX. Stephan Herrmann is a member of AITO. His current focus is on spreading the word about Object Teams and on resolving potential barriers for adoption, should they occur.

Technical Papers at Palazzo Ducale

Wednesday, July 8th (Sala del «Minor Consiglio»)

9:00-9:30 Welcome

9:30-10:30 Invited Talk *Richard Jones (University of Kent, UK)*

Classes, Jim, but not as we know them; Type Classes in Haskell: what, why, and whither
Simon Peyton Jones (Microsoft Research, UK)

Haskell is now quite widely used, but its most important contributions are the ideas that it embodies. In this talk I will focus on one of these ideas, namely type classes, with a few anecdotes and reflections along the way about the process of developing the language.

Type classes are probably Haskell's most distinctive feature. The original idea is very neat and, better still, it led to a long series of subsequent generalisations and innovations. Indeed, although the language is now nineteen years old, Haskell's type system is still in a state of furious development. For example, I am involved in adding type-level functions to Haskell, as I will briefly describe.

I will explain what type classes are, how they differ from the classes of mainstream object oriented languages, why I think they are so cool, and what the hot topics are. I'll give plenty of examples, so you don't need to already know Haskell.

11:00-12:30 Types, Frameworks and Modelling
Viktor Kuncak (École Polytechnique Fédérale de Lausanne, Switzerland)

Coinductive Type Systems for Object-Oriented Languages
Davide Ancona and Giovanni Lagorio (Università di Genova)

We propose a novel approach based on coinductive logic to specify type systems of programming languages. The approach consists in encoding programs in Horn formulas which are interpreted w.r.t. their coinductive Herbrand model.

We illustrate the approach by first specifying a standard type system for a small object-oriented language similar to Featherweight Java. Then we define an idealized type system for a variant of the language where type annotations can be omitted. The type system involves infinite terms and proof trees not representable in a finite way, thus providing a theoretical limit to type inference of object-oriented programs, since only sound approximations of the system can be implemented.

Approximation is naturally captured by the notions of subtyping and subsumption; indeed, rather than increasing the expressive power of the system, as it usually happens, here subtyping is needed for approximating infinite non regular types and proof trees with regular ones.

Checking Framework Interactions with Relationships
Ciera Jaspan and Jonathan Aldrich (Carnegie Mellon University, USA)

Software frameworks impose constraints on how plugins may interact with them. Many of these constraints involve multiple objects, are temporal, and depend on runtime values. Additionally, they are difficult to specify because they are often extrinsic and may break behavioral subtyping. This work introduces relationships as an abstraction for specifying framework constraints in FUSION (Framework Usage SpecificatIONs), and it presents a formal description and implementation of a static analysis to find constraint violations in plugin code. We define three variants of this analysis: one is sound, one is complete, and a pragmatic variant that balances these tradeoffs. We prove soundness and completeness for the appropriate variants, and we show that the pragmatic variant can effectively check constraints from real-world programs.

COPE - Automating Coupled Evolution of Metamodels and Models
Markus Herrmannsdoerfer (Technische Universität München, Germany),
Sebastian Benz (BMW Car IT GmbH, Germany), and
Elmar Juergens (Technische Universität München, Germany)

Model-based development promises to increase productivity by offering modeling languages tailored to a specific domain. Such modeling languages are typically defined by a metamodel. In response to changing requirements and technological progress, the domains and thus the metamodels are subject to change. Manually migrating existing models to a new version of their metamodel is tedious and error-prone. Hence, adequate tool support is required to support the maintenance of modeling languages. This paper introduces COPE, an integrated approach to specify the coupled evolution of metamodels and models to reduce migration effort. With COPE, a language is evolved by incrementally composing modular coupled transformations that adapt the metamodel and specify the corresponding model migrations. This modular approach allows to combine the reuse of recurring transformations with the expressiveness to cater for complex transformations. We demonstrate the applicability of COPE in practice by modeling the coupled evolution of two existing modeling languages.

14:00-15.30 Aliasing and Transactions *Jan Vitek (Purdue University, USA)*

Making Sense of Large Heaps
Nick Mitchell, Edith Schonberg, and Gary Sevitsky (IBM T.J. Watson Research Center, USA)

It is common for large-scale Java applications to suffer memory problems, whether inefficient designs that impede scalability, or lifetime bugs such as leaks. Making sense of heaps with many millions of objects is difficult given the extensive layering, framework reuse, and shared ownership in current applications. We present Yeti, a tool that summarizes memory usage to uncover the costs of design decisions, rather

than of lower-level artifacts as in traditional tools, making it possible to quickly identify and remediate problems. Yeti employs three progressive abstractions and corresponding visualizations: it identifies costly groups of objects that collectively perform a function, recovers a logical data model for each, and summarizes the implementation of each model entity and relationship. Yeti is used by development and service teams within IBM, and has been effective in solving numerous problems. Through case studies we demonstrate how these abstractions help solve common categories of problems.

Scaling CFL-Reachability-Based Points-to Analysis Using Context-Sensitive Must-Not-Alias Analysis

Guoqing Xu, Atanas Rountev (Ohio State University, USA), and
Manu Sridharan (IBM T.J. Watson Research Center, USA)

Pointer analyses derived from a Context-Free-Language (CFL) reachability formulation achieve very high precision, but they do not scale well to compute the points-to solution for an entire large program. Our goal is to increase significantly the scalability of the currently most precise points-to analysis for Java. This CFL-reachability analysis depends on determining whether two program variables may be aliases. We propose an efficient but less precise pre-analysis that computes context-sensitive must-not-alias information for all pairs of variables. Later, these results can be used to quickly filter out infeasible CFL-paths during the more precise points-to analysis. Several novel techniques are employed to achieve precision and efficiency, including a new approximate CFL-reachability formulation of alias analysis, as well as a carefully-chosen trade-off in context sensitivity. The approach effectively reduces the search space of the points-to analysis: the modified points-to analysis is more than three times faster than the original analysis.

NePaLTM: Design and Implementation of Nested Parallelism for Transactional Memory Systems

Haris Volos (University of Wisconsin-Madison, USA), Adam Welc, Ali-Reza Adl-Tabatabai, Tatiana Shpeisman, Xinmin Tian, and Ravi Narayanaswamy (Intel Corporation, USA)

Transactional memory (TM) promises to simplify construction of parallel applications by allowing programmers to reason about interactions between concurrently executing code fragments in terms of high-level properties they should possess. However, all currently existing TM systems deliver on this promise only partially by disallowing parallel execution of computations performed inside transactions. This paper fills in that gap by introducing NePaLTM (Nested ParalleLism for Transactional Memory), the first TM system supporting nested parallelism inside transactions. We describe a programming model where TM constructs (atomic blocks) are integrated with OpenMP constructs enabling nested parallelism. We also discuss the design and implementation of a working prototype where atomic blocks can be used for concurrency control at an arbitrary level of nested parallelism. Finally, we present a

performance evaluation of our system by comparing transactions-based concurrency control mechanism for nested parallel computations with a mechanism already provided by OpenMP based on mutual exclusion.

16:00-17:30 Access Control and Verification *Atsushi Igarashi (Kyoto University)*

Implicit Dynamic Frames: Combining Dynamic Frames and Separation Logic
Jan Smans, Bart Jacobs, and Frank Piessens (Katholieke Universiteit Leuven, Belgium)

The dynamic frames approach has proven to be a powerful formalism for specifying and verifying object-oriented programs. However, it requires writing and checking many frame annotations. In this paper, we propose a variant of the dynamic frames approach that eliminates the need to explicitly write and check frame annotations. Reminiscent of separation logic's frame rule, programmers write access assertions inside pre- and postconditions instead of writing frame annotations. From the precondition, one can then infer an upper bound on the set of locations writable or readable by the corresponding method. We implemented our approach in a tool, and used it to automatically verify several challenging programs, including subject-observer, iterator and linked list.

Fine-Grained Access Control with Object-Sensitive Roles
Jeffrey Fischer, Daniel Marino, Rupak Majumdar, and Todd Millstein (University of California, Los Angeles, USA)

Role-based access control (RBAC) is a common paradigm to ensure that users have sufficient rights to perform various system operations. In many cases, traditional RBAC cannot easily express application-level security requirements. For instance, in a medical records system it is difficult to express that doctors should only update records for their own patients. Further, traditional RBAC frameworks like Java's Enterprise Edition rely solely on dynamic checks, making application code fragile and difficult to ensure correct. We introduce Object-sensitive RBAC (ORBAC), a generalized RBAC model for object-oriented languages. ORBAC resolves the expressiveness limitations of RBAC by allowing roles to be parameterized by properties of the business objects being manipulated. We formalize and prove sound a dependent type system that statically validates a program's conformance to an ORBAC policy. We have implemented our type system for Java and have used it to validate fine-grained access control in the OpenMRS medical records system.

Practical API Protocol Checking with Access Permissions
Kevin Bierhoff, Nels E. Beckman, and Jonathan Aldrich (Carnegie Mellon University, USA)

Reusable APIs often define usage protocols. We previously developed a sound modular type system that checks compliance with typestate-based protocols while affording a great deal of aliasing flexibility. We also developed Plural, a prototype tool

that embodies our approach as an automated static analysis and includes several extensions we found useful in practice. This paper evaluates our approach along the following dimensions: (1) We report on experience in specifying relevant usage rules for a large Java standard API with our approach. We also specify several other Java APIs and identify recurring patterns. (2) We summarize two case studies in verifying third-party open-source code bases with few false positives using our tool. We discuss how tool shortcomings can be addressed either with code refactorings or extensions to the tool itself. These results indicate that our approach can be used to specify and enforce real API protocols in practice.

Thursday, July 9th (Sala del «Minor Consiglio»)

9:00-9:30 Awarding of Dahl-Nygaard Prize

9:30-10:30 Dahl-Nygaard Prize Talk *Eric Jul (University of Copenhagen, Denmark)*

Self and Self: Whys and Wherefores
David Ungar (IBM Research, USA)

Generational garbage collection, prototype-based languages, dynamic optimization, cartoon animation for legibility, all tremendous fun, none done alone. What were they? How did they happen? Why did they matter? Looking back, what is worth learning about these experiences beyond the technical innovations? Combining hindsight with others' wisdom, it is possible to abstract some thoughts that may be useful in other situations: when (not) to listen to wise council; whom to follow into the cafeteria at lunch time; the benefit of striking a balance between one's own vision and those of ones collaborators; which chance events might alter one's course; and how one's best work can sometimes arise from things that, on the surface, have nothing to do with work at all. At a deeper level still, the notion that values, principles, and practices arise in that particular order serves to unify the work and the experiences, and perhaps points the way forward as we all strive to invent the future.

11:00-12:30 Modularity *Erik Ernst (University of Aarhus, Denmark)*

Adding State and Visibility Control to Traits Using Lexical Nesting
Tom Van Cutsem (VUB, Belgium), Alexandre Bergel (INRIA Lille, France), Stéphane Ducasse (INRIA Lille, France), and Wolfgang De Meuter (VUB, Belgium)

Traits are reusable building blocks that can be composed to share methods across unrelated class hierarchies. Original traits are stateless and cannot express visibility control for methods. Two extensions, stateful traits and freezable traits, have been proposed to overcome these limitations. However, these extensions introduce complexity and have not yet been combined to simultaneously add both state and visibility control to traits.

This paper revisits the addition of state and visibility control to traits. Rather than extending the original traits model with additional operations, we allow traits to be lexically nested within other modules. Traits can then have (shared) state and visibility control by hiding variables or methods in their lexical scope. Although the Traits' "flattening property" must be revisited, the combination of traits with lexical nesting results in a simple and expressive trait model. We discuss an implementation of the model in AmbientTalk and specify its operational semantics.

Featherweight Jigsaw — A Minimal Core Calculus for Modular Composition of Classes

Giovanni Lagorio, Marco Servetto, and Elena Zucca (Università di Genova, Italy)

We present FJig, a simple calculus where basic building blocks are classes in the style of Featherweight Java. However, inheritance has been generalized to the more flexible notion proposed in Bracha's Jigsaw framework.

We keep the nominal type approach of Java-like languages, however, a class is not necessarily a structural subtype of any class used in its defining expression.

The calculus allows the encoding of a large variety of different mechanisms for software composition, including standard inheritance, mixin classes, traits and hiding. Hence, FJig can be used as a unifying framework for analyzing existing mechanisms and proposing extensions.

Two different semantics are provided: flattening and direct. The difference is analogous to that between two intuitive models to understand inheritance: where inherited methods are copied into heir classes, and where member lookup ascends the inheritance chain. We address equivalence of these two views for a more sophisticated composition mechanism.

Modular Visitor Components: A Practical Solution to the Expression Families Problem

Bruno Oliveira (Oxford University Computing Laboratory, United Kingdom)

The expression families problem can be defined as the problem of achieving reusability and composability across the components involved in a family of related datatypes and corresponding operations over those datatypes. Like the traditional expression problem, adding new components (either variants or operations) should be possible while preserving modular and static type-safety. Moreover, different combinations of components should have different type identities and the subtyping relationships between components should be preserved. By generalizing previous work exploring the connection between type-theoretic encodings of datatypes and visitors, we propose two solutions for this problem in Scala using modular visitor components. These components can be grouped into features that can be easily composed in a feature-oriented programming style to obtain customized datatypes and operations.

14:00-15:30 Mining and Extracting *Yossi Gil (Google Haifa and Technion, Israel)*

Debugging Method Names
Einar W. Høst (Norsk Regnesentral, Norway) and
Bjarte M. Østvold (Norsk Regnesentral, Norway)

Meaningful method names are crucial for the readability and maintainability of software. Existing naming conventions focus on syntactic details, leaving programmers with little or no support in assuring meaningful names. In this paper, we show that naming conventions can go much further: we can mechanically check whether or not a method name and implementation are likely to be good matches for each other. The vast amount of software written in Java defines an implicit convention for pairing names and implementations. We exploit this to extract rules for method names, which are used to identify "naming bugs" in well-known Java applications. We also present an approach for automatic suggestion of more suitable names in the presence of mismatch between name and implementation.

MAPO: Mining and Recommending API Usage Patterns
Hao Zhong (Peking University, China), Tao Xie (North Carolina State University, USA), Lu Zhang (Peking University, China), Jian Pei (Simon Fraser University, Canada), and Hong Mei (Peking University, China)

API methods provided by software libraries are often complex and not well documented. To get familiar with how API methods are used, programmers often exploit a code search tool for related snippets. However, the returned snippets are often large in number, which place a barrier for programmers to locate useful ones. To help programmers overcome this barrier, we developed MAPO for mining API usage patterns. A mined pattern describes that in a certain usage scenario, some API methods are frequently invoked together and their usages follow some sequential rules. Our experimental result shows that with these patterns MAPO can help programmers locate useful code snippets more effectively than two state-of-the-art code search tools. To investigate whether MAPO can assist programmers in programming tasks, we further conducted an empirical study. The result shows that using MAPO, programmers can produce code with fewer bugs when an API usage is relatively complex.

Supporting Framework Use via Automatically Extracted Concept-Implementation Templates
Abbas Heydarnoori, Krzysztof Czarnecki, and Thiago Tonelli Bartolomei (University of Waterloo, Canada)

Application frameworks provide reusable concepts that are instantiated in application code through potentially complex implementation steps such as subclassing, implementing callbacks, and making calls. Existing applications contain valuable examples of such steps, except that locating them in the application code is often

challenging. We propose the notion of concept implementation templates, which summarize the necessary implementation steps, and an approach to automatic extraction of such templates from traces of sample applications. We demonstrate the feasibility of the template extraction with high precision and recall through an empirical study with twelve realistic concepts from four widely-used frameworks. Finally, we report on a user experiment with twelve subjects in which the choice of templates vs. documentation had much less impact on development time than the concept complexity.

16:00-17:30 Refactoring *Oscar Nierstrasz (University of Bern, Switzerland)*

Stepping Stones over the Refactoring Rubicon
Max Schaefer, Mathieu Verbaere, Torbjorn Ekman, and Oege de Moor (Oxford University, UK)

Refactoring tools allow the programmer to pretend they are working with a richer language where the behaviour of a program is automatically preserved during restructuring. In this paper we show that this metaphor of an extended language yields a very general and useful implementation technique for refactorings: a refactoring is implemented by embedding the source program into an extended language on which the refactoring operations are easier to perform,and then translating the refactored program back into the original language. Using the well-known Extract Method refactoring as an example, we show that this approach allows a very fine-grained decomposition of the overall refactoring into a series of micro-refactorings that can be understood, implemented, and tested independently. We thus can easily write implementations of complex refactorings that rival and even outperform industrial strength refactoring tools in terms of correctness, but are much shorter and easier to understand.

Program Metamorphosis
Christoph Reichenbach, Devin Coughlin, and Amer Diwan (University of Colorado, USA)

Modern development environments support refactoring by providing atomically behaviour-preserving transformations. While useful, these transformations are limited in three ways: (i) atomicity forces transformations to be complex and opaque, (ii) the behaviour preservation requirement disallows deliberate behaviour evolution, and (iii) atomicity limits code reuse opportunities for refactoring implementers.

We present 'program metamorphosis', a novel approach for program evolution and refactoring that addresses the above limitations by breaking refactorings into smaller steps that need not preserve behaviour individually. Instead, we ensure that sequences of transformations preserve behaviour together, and simultaneously permit selective behavioural change.

To evaluate program metamorphosis, we have implemented a prototype plugin for Eclipse. Our analysis and experiments show that (1) our plugin provides correctness guarantees on par with those of Eclipse's own refactorings, (2) both our plugin

and our approach address the aforementioned limitations, and (3) our approach fully subsumes traditional refactoring.

From Public to Private to Absent: Refactoring Java Programs under Constrained Accessibility
Friedrich Steimann and Andreas Thies (Fernuniversität in Hagen, Germany)

Contemporary refactoring tools for Java aiding in the restructuring of programs have problems with respecting access modifiers such as public and private: while some tools provide hints that referenced elements may become inaccessible due to the intended restructuring, none we have tested prevent changes that alter the meaning of a program, and none take steps that counteract such alterations. To address these problems, we formalize accessibility in Java as a set of constraint rules, and show how the constraints obtained from applying these rules to a program and an intended refactoring allow us to check the preconditions of the refactoring, as well as to compute the changes of access modifiers necessary to preserve the behaviour of the refactored program. We have implemented our framework as a proof of concept in Eclipse, and demonstrated how it improves applicability and success of an important refactoring in a number of sample programs. That our approach is not limited to Java is shown by comparison with the constraint rules for C# and Eiffel.

Friday, July 10th (Sala del «Minor Consiglio»)

9:30-10:00 Invited Talk *Doug Lea (State University of New York at Oswego, USA)*

Java on 1000 Cores: Tales of Hardware/Software Co-design
Cliff Click (Azul Systems, USA)

Azul Systems designs and builds systems for running business logic applications written in Java. Unlike scientific computing, business logic code tends to be very large and complex (> 1MLOC is *common*), display very irregular data access patterns, and make heavy use of threads and locks. The common unit of parallelism is the transaction or thread-level task. Business logic programs tend to have high allocation rates which scale up with the amount of work accomplished, and they are sensitive to Garbage Collection max-pause-times. Typical JVM implementations for heaps greater than 4 Gigabytes have unacceptable pause times and this forces many applications to run clustered.

Our systems support heaps up to 600 Gigabytes and allocation rates up to 35 Gig/s with pause times in the dozen-millisecond range. We have large core counts (up to 864) for running parallel tasks; our memory is Uniform Memory Access (as opposed to the more common NUMA), cache-coherent, and has supercomputer-level bandwidth. The cores are our own design; simple 3-address RISCs with read- and write-barriers to support GC, hardware transactional memory, zero-cost high-rez profiling, and some more modest Java-specific tweaks.

This talk is about the business environment which drove the design of the hardware (e.g. why put in HTM support? why our own CPU design and not e.g. MIPS or X86?), some early company history with designing our own chips (1st silicon back from the fab had problems like the bits in the odd-numbered registers bleeding into the even-numbered registers), and finally some wisdom and observations from a tightly integrated hardware/software co-design effort.

10.30-12:30 Concurrency, Exceptions and Initialization
Kathryn Gray (University of Cambridge, UK)

Loci: Simple Thread-Locality for Java
Tobias Wrigstad, Filip Pizlo, Fadi Meawad, Lei Zhao, and Jan Vitek (Purdue University, USA)

This paper presents a simple type system for thread-local data in Java. Classes and types are annotated to express thread-locality and unintended leaks are detected at compile-time. The system, called Loci, is minimal, modular and compatible with legacy code. The only change to the language is the addition of two new metadata annotations. We implemented Loci as an Eclipse plug-in and used it to evaluate our design on a number of benchmarks. We found that Loci is compatible with how Java programs are written and that the annotation overhead is light thanks to a judicious choice of defaults.

Failboxes: Provably Safe Exception Handling
Bart Jacobs and Frank Piessens (Katholieke Universiteit Leuven, Belgium)

The primary goal of exception mechanisms is to help ensure dependency safety: that when an operation fails, code that depends on the operation's successful completion is not executed. The exception mechanisms of current mainstream programming languages make it hard to achieve this, particularly when objects manipulated inside a try block outlive the try block.

We propose a language mechanism called failboxes. Programmers may create failboxes dynamically and execute blocks of code in them. Once any such block fails, all subsequent attempts to execute code in the failbox will fail. To achieve dependency safety, programmers simply need to ensure that if one operation depends on another operation, then both are executed in the same failbox. Furthermore, failboxes help fix the unsafe interaction between locks and exceptions; they enable safe cancellation and robust resource cleanup; and they help prevent liveness issues when a thread is waiting on a failed thread.

Are We Ready for a Safer Construction Environment?
Yossi Gil (Google, Haifa, Israel) and
Tali Shragai (The Technion, Israel)

If a constructor of a base class invokes a method overridden in the derived, then (in many languages) this overridden method is invoked on a "half baked" object, without any language aid to warn the method against this.

This situation may lead to confusing semantics, complicated coupling between the base and the derived, and difficulty in introducing features such as nonnull and readonly specification into the language.

This work is an empirical investigation of calling dynamically bound methods in the current programming practice in Java.

In a data set comprising over 60K classes, we found that although the potential for such a situation is non-negligible, this potential is realized scarcely, occurring in 1.5% of all constructors, inheriting from 0.5% of all constructors.

We found that over 80% of these incidents fall into eight "patterns", which can be relatively easily transformed into equivalent safer code.

Type-based Object Immutability with Flexible Initialization
Christian Haack and Erik Poll (Radboud University Nijmegen, The Netherlands)

We present a type system for checking object immutability, read-only references, and class immutability in an open or closed world. To allow object initialization outside object constructors (which is often needed in practice), immutable objects are initialized in lexically scoped regions. The system is simple and direct; its only type qualifiers specify immutability properties. No auxiliary annotations, e.g., ownership types, are needed, yet good support for deep immutability is provided. To express object confinement, as required for class immutability in an open world, we use qualifier polymorphism. The system has two versions: one with explicit specification commands that delimit the object initialization phase, and one where such commands are implicit and inferred. In the latter version, all annotations are compatible with Java's extended annotation syntax, as proposed in JSR 308.

14:00-15.30 Concurrency and Distribution
Gary Leavens (University of Central Florida, USA)

Security Monitor Inlining for Multithreaded Java
Mads Dam (KTH, Sweden), Bart Jacobs (K.U.Leuven, Belgium),
Andreas Lundblad (KTH, Sweden), and Frank Piessens (K.U.Leuven, Belgium)

Program monitoring is a well-established and efficient approach to security policy enforcement. An implementation of program monitoring that is particularly appealing for application-level policy enforcement is monitor inlining: the application is rewritten to push monitoring and policy enforcement code into the application itself. The intention is that the inserted code enforces compliance with the policy (security),

and otherwise interferes with the application as little as possible (conservativity and transparency).

For sequential Java-like languages, provably correct inlining algorithms have been proposed, but for the multithreaded setting, this is still an open problem. We show that no inliner for multithreaded Java can be both secure and transparent. It is however possible to identify a broad class of policies for which all three correctness criteria can be obtained. We propose an inliner that is correct for such policies, implement it for Java, and show that it is practical by reporting on some benchmarks.

EventJava: An Extension of Java for Event Correlation
Patrick Eugster and K. R. Jayaram (Purdue University, USA)

Event correlation has become the cornerstone of many reactive applications, particularly in distributed systems. However, support for programming with complex events is still rather specific and rudimentary. This paper presents EventJava, an extension of Java with generic support for event-based distributed programming. EventJava seamlessly integrates events with methods, and broadcasting with unicasting of events; it supports reactions to combinations of events, and predicates guarding those reactions. EventJava is implemented as a framework to allow for customization of event semantics, matching, and dispatching. We present its implementation, based on a compiler transforming specific primitives to Java, along with a reference implementation of the framework. We discuss ordering properties of EventJava through a formalization of its core as an extension of Featherweight Java. In a performance evaluation, we show that EventJava compares favorably to a highly tuned database-backed event correlation engine as well as to a comparably lightweight concurrency mechanism.

Remote Batch Invocation for Compositional Object Services
Ali Ibrahim (University of Texas at Austin, USA), Yang Jiao (Virginia Tech, USA), Eli Tilevich (Virginia Tech, USA), and William R. Cook (University of Texas at Austin, USA)

Because Remote Procedure Calls do not compose efficiently, designers of distributed object systems use Data Transfer and Remote Facade patterns to create large-granularity interfaces, hard-coded for particular client use cases. As an alternative to RPC-based distributed objects, this paper presents Remote Batch Invocation (RBI), language support for explicit client-defined batches. A Remote Batch statement combines remote and local execution: all the remote code is executed in a single round-trip to the server, where all data sent to the server and results from the batch are communicated in bulk. RBI supports remote blocks, iteration and conditionals, and local handling of remote exceptions. RBI is efficient even for fine-grained interfaces, eliminating the need for hand-optimized server interfaces. We demonstrate RBI with an extension to Java, using RMI internally as the transport layer. RBI supports large-granularity, stateless server interactions, characteristic of service-oriented computing.

Posters & Demos Session at Palazzo Ducale

ECOOP 2009 will hold combined poster and demonstration sessions. The poster & demo Session is an opportunity for presenting late-breaking results, ongoing research projects, and speculative or innovative work in progress, in an informal and interactive setting. Posters and demos are intended to provide authors and participants with the ability to connect with each other and to engage in discussions about the work. We will be pleased to have you attend our two posters & demos sessions at Loggiato Minore on Wednesday 8th and Thursday 9th 17:30-18:30.

Wednesday 8th and Thursday 9th (17:30-18:30, Loggiato Minore)

Implementing Aggregation in Java
George B. Wilson and Azadeh Ebrahimi (Anglia Ruskin University, UK)

The Unified Modeling Language (UML) is a standard design tool for object oriented environments. However, despite of the richness of UML to model critical systems, the whole-part relationship (aggregation) is not unambiguously supported by the lower level programming syntax available in many object-oriented languages. This work does not attempt to elaborate or redefine the ambiguity of aggregation but rather suggests a simple strategy that might be adopted by the programmer to implement this relationship specifically (but not exclusively) in the Java programming language.

Typestate Checking in Concurrent Programs with Synchronized Blocks
Nels E. Beckman and Jonathan Aldrich (Carnegie Mellon University, USA)

In previous work we described a modular static analysis based on access permission annotations, which describe the ways in which a reference can be aliased, for preventing the improper use of object protocols in concurrent programs. That system was based on atomic blocks, a mutual exclusion primitive not yet in wide use. Now we extend that system to programs written using synchronized blocks, which are in wide use today. This system can verify concurrent programs without any concurrency-specific annotations.

Javeleon - Extending NetBeans with Dynamic Update of Active Modules
Allan Gregersen (University of Southern Denmark, Denmark)

This demo looks at full dynamic module updates on the NetBeans^{TM}platform, showing how extensive changes to the code of running application modules can be made on the fly. A demonstration of changes to the class inheritance hierarchies of active classes reveals a so-far-unseen power in the field of dynamic update at the application level. In addition, standard development practices are unaffected and programmers do not have to provide additional information to the underlying update

mechanism, which leaves the update almost completely transparent, although still very flexible.

COPE - Automating Coupled Evolution of Metamodels and Models
Markus Herrmannsdoerfer (Technische Universität Munchen)

Have you ever evolved your metamodel and your models were no longer valid afterwards? Or have you avoided to evolve your metamodel in order not to invalidate your models? Or have you even deteriorated your metamodel so that it remains downwards compatible to previous versions in order to avoid these problems? This poster and demo introduces COPE, a tool that eases the migration of models in response to an evolving metamodel. COPE explicitly records the history of the metamodel as a sequence of operations and allows to attach information on how to migrate models. The combination of metamodel adaptation and model migration is referred to as COuPled Evolution of metamodels and models. The attached information can be used to automatically migrate models to the new version of the metamodel. To further reduce migration effort, COPE allows to reuse combinations of metamodel adaptation and model migration steps across metamodels. COPE is implemented based on the Eclipse Modeling Framework (EMF) which is probably the most widely used metamodeling framework. In order not to disturb EMF users in their habits, COPE seamlessly integrates into the existing metamodel editor.

A Demo of Azul Systems' JVM Profiling Tool
Cliff Click (Azul Systems, USA)

Every JVM running on Azul Systems' gear comes with RTPM — a *Real-Time Performance Monitor*. RTPM is a zero-overhead always-on profiler, and is accessed through any browser. RTPM shows the inner state of the JVM in great detail through the standard browser interface. RTPM shows e.g., hot code, hot locks, live running thread stacks, the current heap state, live object counts, I/O rates, GC pauses & allocate rates. For hot code, RTPM samples every thread 1000 times/sec (with no overhead because of hardware support), displays the hot JIT'd code with sample counts, profiling data, compiler & inlining decisions. For hot locks, RTPM collects stack traces of blocking threads and can display the hot-paths leading to contended locks. For live thread stacks, RTPM can show full stack traces of all threads, down to the bytecode level (generally with Java variables named). The stack-trace is fully live, and every variable & heap object can be inspected. The heap information includes allocation counts, live-object counts, and points-to information. RTPM can monitor these and many more aspects of a JVM. I will demo using RTPM to inspect & diagnose performance problems in a live webserver.

More information can be found on Azul Systems' Engineer-2-Engineer webpages: `http://www.azulsystems.com/e2e`.

General Information

Exhibition

Throughout the conference, the ECOOP 2009 sponsors and exhibitors will have exhibition stands in the *Loggiato Maggiore*.

Birds-of-a-Feather Sessions

A Birds-of-a-Feather Session on *Software engineering at Google* has been organized by Peter Dickman and Yossi Gil, on Thursday July 9th 17:30-18:15 (Sala «Minor Consiglio»).

Further Birds-of-a-Feather sessions can be arranged on request, and will be advertised on the information board near the conference desk (Loggiato Maggiore).

Wireless Access

WLAN access will be available, both at Hotel Bristol and at Palazzo Ducale, Piano Nobile. Instruction on its use will be given during registration.

Information Board

Near the conference desk there will be a message and information board where you can leave messages and where the conference staff will post messages for participants and current information.

Luggage Facilities

Luggage can be stored in the wardrobe room at the left of Sala del Minor Consiglio entrance. Our staff will look after your possessions but we are unable to accept any responsibility for loss or damage.

Smoking Policy

A smoking ban applies in all conference buildings.

Parking Facilities

Parking near the conference sites is possible only at payment parking places in *Piazza Piccapietra* and *Piazza Dante*.

Catering Arrangements

Coffee Breaks

Coffee breaks will be daily at **10:30-11:00** and at **15.30-16.00**.

On **Monday** and **Tuesday**, coffee breaks will be at Hotel Bristol, **Sala Paganini A**, 2nd Floor.

On **Wednesday**, **Thursday**, and **Friday**, coffee breaks will be served in the **"Loggiato Maggiore"**, Piano Nobile (3rd Floor), Palazzo Ducale.

Lunches

Lunch will be served daily **12:30-14:00** at **"Le Terrazze del Ducale"**, on the terrace of Palazzo Ducale.

Evening Social Events

Monday, July 6th, 19:00-22:00 – Workshop Reception

The Workshop Reception will be held at "Le Terrazze del Ducale", on the terrace of Palazzo Ducale, offering a spectacular aerial view of Genova's old center. Snacks and drinks will be served.

Wednesday, July 8th, 19:00-22:00 – Welcome Party

At Aula Magna of the University Rectorate in via Balbi, where some University buildings are part of the Rolli system, a set of mid 17th century palaces which are included in the UNESCO World Heritage list. Snacks and drinks will be served.

The University Rectorate can be reached from the conference site by foot (10-minutes walk). It is a very nice walk through the main streets of Genova. Alternatively, you may take buses number 20, 30, or 35 in Piazza de Ferrari (Carlo Felice side) and get off at the *Via Balbi-Università* bus stop.

Address: via Balbi, 5 – 16126 Genova
Meeting Point: Palazzo Ducale entrance on Piazza de Ferrari, 18:45

Thursday, July 9th, 19:30-23:30 – Conference Banquet

At Villa Lo Zerbino, a 16th century mansion surrounded by a wonderful park designed by the architect of Versailles in Paris. A guided visit of the Villa and its prestigious painting collection is included. Appetizers and sweets buffets will be in the park (weather permitting) while dinner will be served in the wonderful painted rooms on the first floor.

We are pleased to have a banquet speech by William Cook, entitled *Objectives & Objections*.

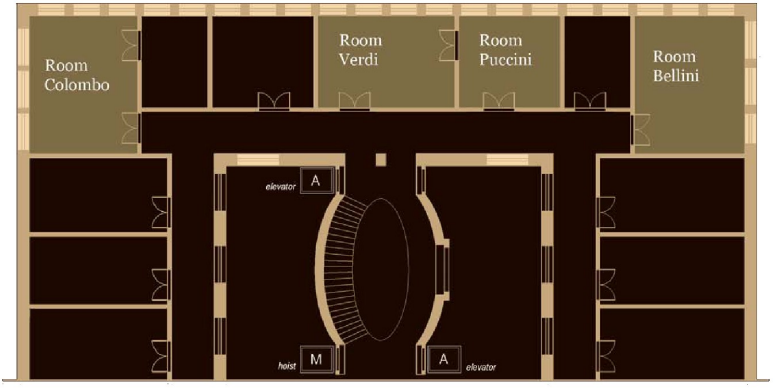
Villa Lo Zerbino can be reached from the conference site by foot (20-minutes walk). Alternatively, you may take bus number 36 in Piazza de Ferrari (Carlo Felice side) and get off at the *Piazza Manin* bus stop.

Address: Passo dello Zerbino, 1 – 16122 Genova
Meeting Point: Palazzo Ducale entrance on Piazza de Ferrari, 19:00

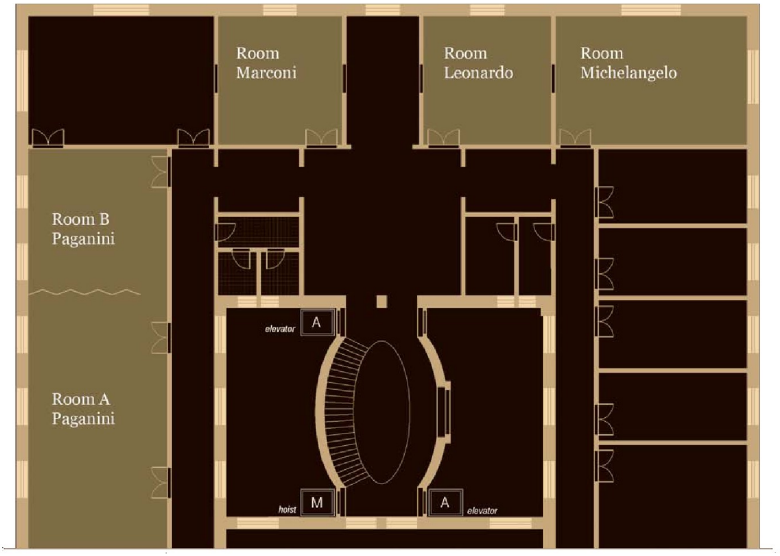
Conference Site Information

Hotel Bristol — Workshops, July, 6th-7th

First Floor



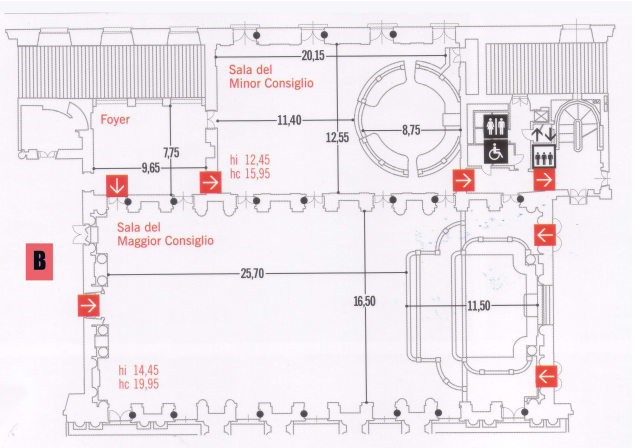
Second Floor



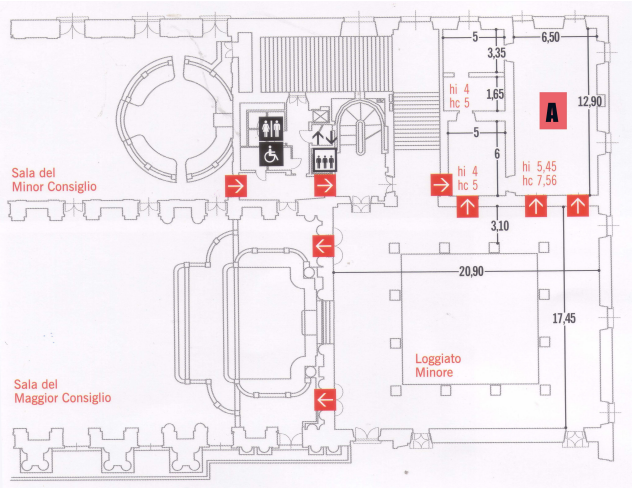
Third Floor: Mazzini Room

Palazzo Ducale — Main Conference, July, 8th-10th

Third Floor — Piano Nobile



B Loggiato Maggiore



A Sala Camino

Genova

Genova, the capital of Liguria, stretches along the bay of the same name from Voltri to the west as far as Nervi to the east, while the hinterland area takes in the lower parts of the Polcevera and Bisagno Valleys.

The original nucleus of the city, which already existed in pre-Roman times, developed around the Mandraccio wharf area and on Castello Hill, which overlooks it.

In the ninth century, the Genoese built the first town walls and laid the foundations for the development of shipping and sea-trading, which would eventually make the Republic of Genova a Mediterranean sea power and create a dominion stretching across the entire region of Liguria. From the nineteenth century onwards, the great city port was flanked by large industrial areas. The old town district is one of the largest in Europe, and hosts some remarkable artistic and architectural treasures, including the Palazzi dei Rolli, fifty or so homes of the aristocracy entered on the UNESCO World Heritage List.

In addition to offering a wealth of cultural attractions, Genova is a fascinating destination for tourists, with its scenic vantage points, sea promenades, aristocratic villas and of course the Riviera to the east and west, both easy to reach: Porto Venere and Le Cinque Terre (also UNESCO World Heritage Sites), Tigullio, Portofino and Camogli to the east and Alassio, Sanremo, Bordighera to the west.

Liguria

Liguria is a narrow strip of land, enclosed between the sea and the Alps and the Apennines mountains, it is a winding arched extension from Ventimiglia to La Spezia and is one of the smallest regions in Italy.

It is limited in size, but not in the variety of its vegetation and wildlife which is amongst the most diversified and interesting in Italy. The coast-line, which is geographically divided between the Western Riviera and the Eastern Riviera at the sides of the important center of Genova, from the scenic point of view is characterized by an alternating series of magnificent high coast-lines and flat, sandy coast-lines, whilst in the interior the steep hills meet up with the Apennines peaks.

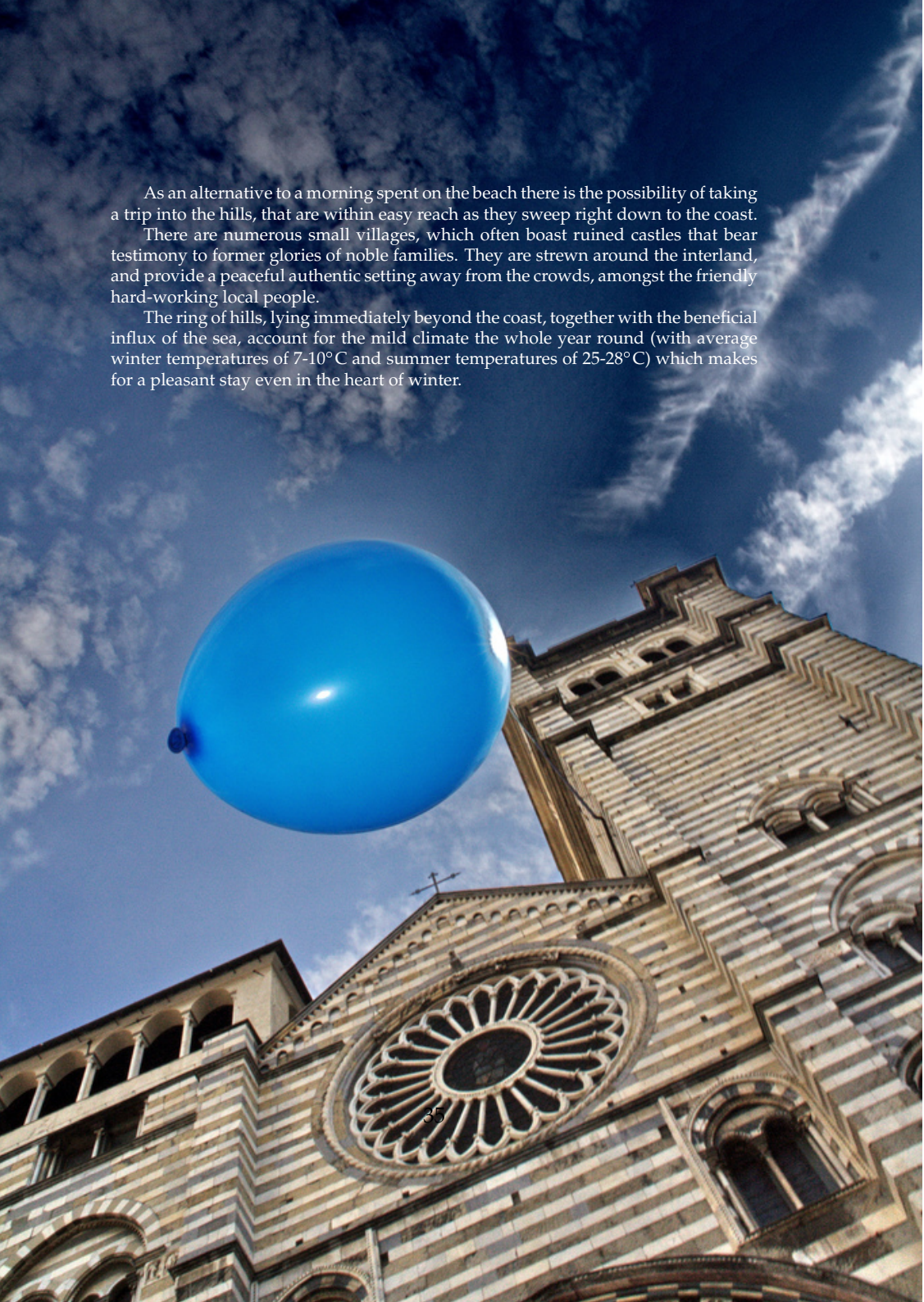
Liguria has an abundance of natural beauty and the various names given to it such as “Paradise Gulf”, “Siren bay”, “Bay of silence”, “Bay of fairy tales”, “Sea’s echo” are all a testimony to the magnificent beauty of these marine landscapes.

The host of hotels and seaside facilities ensure that the tourist can enjoy the very best kind of holiday.

As an alternative to a morning spent on the beach there is the possibility of taking a trip into the hills, that are within easy reach as they sweep right down to the coast.

There are numerous small villages, which often boast ruined castles that bear testimony to former glories of noble families. They are strewn around the interland, and provide a peaceful authentic setting away from the crowds, amongst the friendly hard-working local people.

The ring of hills, lying immediately beyond the coast, together with the beneficial influx of the sea, account for the mild climate the whole year round (with average winter temperatures of 7-10° C and summer temperatures of 25-28° C) which makes for a pleasant stay even in the heart of winter.



Executive Committee

Sophia Drossopoulou	Imperial College, United Kingdom	*Program Chair*
Giovanna Guerrini	DISI, Università di Genova, Italy	*Conference Chairs*
Elena Zucca	DISI, Università di Genova, Italy	
Davide Ancona	DISI, Università di Genova, Italy	*Organizing Chairs*
Walter Cazzola	DICo, Università di Milano, Italy	

Organizing Committee

Ferruccio Damiani	Università di Torino, Italy	*Workshop Chairs*
Mario Südholt	Ècole des Mines de Nantes, France	
Antonio Cisternino	Università di Pisa, Italy	*Summer School*
Paola Giannini	Univ. del Piemonte Orientale, Italy	*Committe*
James Noble	Victoria University of Wellington, NZ	
Lorenzo Bettini	Universià di Torino, Italy	*Poster &*
Giovanni Lagorio	DISI, Università di Genova, Italy	*Demo Chairs*
Giovanni Rimassa	Whitestein Technologies, Zürich, CH	*Exhibition Chairs*
Mirko Viroli	Università di Bologna, Italy	
Dave Clarke	K.U. Leuven, Belgium	*Publicity Chair*
Marco Servetto	DISI, Università di Genova, Italy	*Stud. Vol. Chair*
Vittoria Gianuzzi	DISI, Università di Genova, Italy	*Sponsor Chair*
Antonio Cuni	DISI, Università di Genova, Italy	*Web Master*

Student Volunteers

This year, 12 Student Volunteers from across the globe will work side-by-side with the locals to make this the best ECOOP experience ever. Please feel free to approach the Student Volunteers at any time during the conference or the social events with any questions or comments you may have. They will make every effort to help you. They also know the best places to go and can give you ideas and hints on how to make the most of the conference and your stay in Genoa.

Nels Beckman	Carnegie Mellon University	USA
Jia Dai	Politecnico di Torino	Italy
Sebastian Götz	Dresden University of Technology	Germany
Mayleen Lacouture	École des Mines de Nantes	France
Paley Li	Victoria University of Wellington	New Zealand
Ismael Mejía	École des Mines de Nantes	France
José Felipe Mejia Bernal	Politecnico di Torino	Italy
Radu Muschevici	Katholieke Universiteit Leuven	Belgium
Roberto Perillo	Aeronautical Institute of Technology	Brazil
Gregor Richards	Purdue University	USA
Ilya Sergey	Katholieke Universiteit Leuven	Belgium
Muhammad Uzair	INRIA, Sophia Antipolis	France

Program Committee

Elisa Baniassad	The Chinese University of Hong Kong, China
Françoise Baude	University of Nice, Sophia Antipolis, France
Bernhard Beckert	University of Koblenz, Germany
Lodewijk Bergmans	University of Twente, The Netherlands
John Tang Boyland	University of Wisconsin-Milwaukee, USA
William Cook	University of Texas at Austin, USA
Eric Eide	University of Utah, USA
Erik Ernst	University of Aarhus, Denmark
Cormac Flanagan	University of California at Santa Cruz, USA
Yossi Gil	Google Haifa and Technion, Israel
Neal Glew	Intel, USA
Kathryn E. Gray	University of Cambridge, UK
Görel Hedin	Lund University, Sweden
Atsushi Igarashi	Kyoto University, Japan
Richard Jones	University of Kent, UK
Viktor Kuncak	EPFL, Switzerland
Doug Lea	State University of New York at Oswego, USA
Gary T. Leavens	University of Central Florida, USA
Oscar Nierstrasz	University of Bern, Switzerland
James Noble	University of Wellington, New Zealand
Nathaniel Nystrom	IBM Research, USA
Awais Rashid	Lancaster University, UK
Diomidis Spinellis	Athens University of Economics and Business, Greece
Peter Sewell	University of Cambridge, UK
Laurence Tratt	Bournemouth University, UK
Jan Vitek	Purdue University, USA
Matthias Zenger	Google, Switzerland
Elena Zucca	University of Genova, Italy


ERCIM
European Research Consortium for Informatics and Mathematics
IBM Research
Google
Microsoft Research
acm
SIGPLAN
SIGSOFT
Google
YAHOO! RESEARCH
DISI
DICo