AOSD and Reflection: Benefits and Drawbacks to Software Evolution

Report on the WS RAM-SE at ECOOP'06

Walter Cazzola[1], Shigeru Chiba[2], Yvonne Coady[3], and Gunter Saake[4]

[1] DICo - Department of Informatics and Communication,
Università degli Studi di Milano, Milano, Italy
cazzola@dico.unimi.it
[2] Department of Mathematical and Computing Sciences,
Tokyo Institute of Technology, Tokyo, Japan
chiba@is.titech.ac.jp
[3] Department of Computer Science,
University of Victoria, Victoria, Canada
ycoady@cs.uvic.ca
[4] Institute für Technische und Betriebliche Informationssysteme,
Otto-von-Guericke-Universität Magdeburg, Magdeburg, Germany
saake@iti.cs.uni-magdeburg.de

Abstract. Following last two years' RAM-SE (Reflection, AOP and Meta-Data for Software Evolution) workshop at the ECOOP conference, the RAM-SE 2006 workshop was a successful and popular event. As its name implies, the workshop's focus was on the application of reflective, aspect-oriented and data-mining techniques to the broad field of software evolution. Topics and discussions at the workshop included mechanisms for supporting software evolution, technological limits for software evolution and tools and middleware for software evolution.

The workshop's main goal was to bring together researchers working in the field of software evolution with a particular interest in reflection, aspect-oriented programming and meta-data. The workshop was organized as a full day meeting, partly devoted to presentation of submitted position papers and partly devoted to panel discussions about the presented topics and other interesting issues in the field. In this way, the workshop allowed participants to get acquainted with each other's work, and stimulated collaboration. We hope this helped participants in improving their ideas and the quality of their future publications.

The workshop's proceedings, including all accepted position papers can be downloaded from the workshop's web site and a post workshop proceeding, including an extension of the accepted paper is published byt the University of Magdeburg.

In this report, we first provide a session-by-session overview of the presentations, and then present our opinions about future trends in software evolution.

Workshop Description and Objectives

Software evolution and adaptation is a research area, as also the name states, in continuous evolution, that offers stimulating challenges for both academic and industrial

M. Südholt and C. Consel (Eds.): ECOOP 2006 Ws, LNCS 4379, pp. 40–52, 2007.

researchers. The evolution of software systems, to face unexpected situations or just for improving their features, relies on software engineering techniques and methodologies. Nowadays a similar approach is not applicable in all situations e.g., for evolving nonstopping systems or systems whose code is not available.

Features of reflection such as transparency, separation of concerns, and extensibility seem to be perfect tools to aid the dynamic evolution of running systems. Aspect-oriented programming (AOP in the next) can simplify code instrumentation whereas techniques that rely on meta-data can be used to inspect the system and to extract the necessary data for designing the heuristic that the reflective and aspect-oriented mechanism use for managing the evolution.

We feel the necessity to investigate the benefits brought by the use of these techniques on the evolution of object-oriented software systems. In particular we would determine how these techniques can be integrated with more traditional approaches to evolve a system and the benefits we get from their use.

The overall goal of this workshop was that of supporting circulation of ideas between these disciplines. Several interactions were expected to take place between reflection, aspect-oriented programming and meta-data for the software evolution, some of which we cannot even foresee. Both the application of reflective or aspect-oriented techniques and concepts to software evolution are likely to support improvement and deeper understanding of these areas. This workshop has represented a good meeting-point for people working in the software evolution area, and an occasion to present reflective, aspect-oriented, and meta-data based solutions to evolutionary problems, and new ideas straddling these areas, to provide a discussion forum, and to allow new collaboration projects to be established. The workshop was a full day meeting. One part of the workshop was devoted to presentation of papers, and another to panels and to the exchange of ideas among participants.

Workshop Topics and Structure

Every contribution that exploits reflective techniques, aspect-oriented programming and/or meta-data to evolve software systems were welcome. Specific topics of interest for the workshop have included, but were not limited to:

- aspect-oriented middleware and environments for software evolution;
- adaptive software components and evolution as component composition;
- evolution planning and deployment through aspect-oriented techniques and reflective approaches;
- aspect interference and composition for software evolution;
- feature- and subject-oriented adaptation;
- unanticipated software evolution supported by AOSD or reflective techniques;
- MOF, code annotations and other meta-data facilities for software evolution;
- software evolution tangling concerns;
- techniques for refactoring into AOSD and to get the separation of concerns.

To ensure lively discussion at the workshop, the organizing committee has chosen the contributions on the basis of topic similarity that will permit the beginning of new

collaborations. To grant an easy dissemination of the proposed ideas and to favorite an ideas interchange among the participants, accepted contributions are freely downloadable from the workshop web page:

`http://homes.dico.unimi.it/RAM-SE06.html`.

The workshop was a full day meeting organized in four sessions. The first session was devoted to the Awais Rashid's keynote speech on "*Aspects and Evolution: The Case for Versioned Types and Meta-Aspect Protocols*". Each of the remaining sessions has been characterized by a dominant topic that perfectly describes the presented papers and the related discussions. The three dominant topics were: *aspect-oriented modeling for software evolution*, *tools and middleware for software evolution*, and *technological limits to software evolution*. During each session, half time has been devoted to papers presentation, and the rest of the time has been devoted to debate about the on-going works in the area, about relevance of the approaches in the software evolution area and the achieved benefits. The discussion related to each session has been brilliantly lead respectively by Theo D'Hondt, Mario Südholt and Hidehiko Masuhara.

The workshop has been very lively, the debates very stimulating, and the high number of participants (see appendix A) testifies the growing interest in the application of reflective, aspect- and meta-data oriented techniques to software evolution.

Important References

The following publications are important references for people interested in learning more about the topics of this workshop:

- Pattie Maes. Computational Reflection. PhD thesis, Vrije Universiteit Brussel, Brussels, Belgium, 1987.
- Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-Oriented Programming. In *11th European Conference on Object Oriented Programming (ECOOP'97)*, LNCS 1241, pages 220–242, Helsinki, Finland, June 1997. Springer-Verlag.
- The proceedings of the International Conference on Aspect-Oriented Software Development (AOSD) from 2002 to 2006. See also `http://aosd.net/archive/index.php`.
- Several tracks related to aspect-oriented software development and evolution at the International Conference on Software Maintenance (ICSM) and the Working Conference on Reverse Engineering (WCRE), from 2002 onward.
- The software evolution website at the Program Transformation wiki:

  `http://www.program-transformation.org/twiki/bin/view/Transform/SoftwareEvolution`.

1 Workshop Overview: Session by Session

In this section of the report we summarize the workshop. In particular, Shigeru Chiba, in the role of chairman of the invited talk session, will comment the Nierstrasz's talk and the discussions raised from the talk, then a short summary of the the remaining sessions will follow.

Session on Aspects and Evolution

Keynote Speech by Awais Rashid (*Lancaster University, UK*)
Summary by Shigeru Chiba (Session Chair, *Tokyo Institute of Technology*)

In the first morning session, we had a keynote talk by Dr. Awais Rashid:

– Aspects and Evolution: The Case for Versioned Types and Meta-Aspect Protocol.

Abstract. One of the often cited advantages of aspect-oriented programming (AOP) is improved evolvability. No doubt the modularisation support provided by aspects helps to localise changes thus supporting evolution. However, evolution often requires keeping track of changes in order to make them reversible. Furthermore, often such changes (and their reversal) needs to be done online, e.g., in case of business and mission critical systems that can't be taken offline. In this talk, I will discuss whether current AOP mechanisms are suited to such evolution needs. I will highlight the need for first class support for versioned types as well as fully-fledged meta-aspect protocols and present some practical experiences of implementing these in the Vejal aspect language and its associated dynamic evolution framework. The talk will conclude with a roadmap of key research issues that need to be tackled to ensure that the full potential of aspects can be realised with regards to improving the evolvability of software systems.

Dr. Rashid's keynote was very interesting and led active discussion among participants. In particular, the first half of his talk was about a classic problem in AOP, which is the problem of obliviousness and quantification. Although it was first proposed by Filman and Friedman [10] that the primary properties of AOP are obliviousness and quantification, this issue has been actively discussed by many researchers. Dr. Rashid's claim was that obliviousness and quantification are not mandatory properties of AOP but they are only desirable properties. According to his talk, the obliviousness property often makes modular reasoning difficult. Hence, open modules and XPIs have been proposed as technique for limiting the obliviousness property and enabling easier modular reasoning. This fact shows that this property is not necessary (therefore, it can be limited). The quantification property is also useful but most of aspects, for example, in the database transaction domain, are not heterogeneous aspects but homogeneous aspects. Homogeneous aspects do not need the complex functionality of the quantification property. His claim was appealing and led a number of comments, objections, and supports from the floor.

Session on Aspect-Oriented Modeling for Software Evolution

The second session was related to the use of the aspect-oriented modeling to support the software evolution, Theo D'Hondt (Vrije Universiteit Brussel, Belgium) was the chairman. Three papers have been presented:

[7] Improving AOP Systems' Evolvability by Decoupling Advices from Base Code. *Alan Cyment*, *Nicolas Kicillof*, *Rubén Altman*, and *Fernando Asteasuain* (University of Buenos Aires, Argentina).

Alan Cyment gave the talk.

[16] Making Aspect Oriented System Evolution Safer. *Miguel Ángel Pérez Toledano, Amparo Navasa Martinez, Juan M. Murillo Rodriguez,* (University of Extremadura, Spain) and *Carlos Canal* (University of Málaga).

Miguel Ángel Pérez Toledano gave the talk.

[6] Design-Based Pointcuts Robustness Against Software Evolution. *Walter Cazzola* (DICo Università degli Studi di Milano, Italy), *Sonia Pini*, and *Massimo Ancona* (DISI Università degli Studi di Genova, Italy).

Sonia Pini gave the talk.

Session on Tools and Middleware for Software Evolution

The papers in this session covered the topic of adaptive middleware to support software evolution. Mario Südholt (École de Mines de Nantes) has chaired the session. Three papers have been presented:

[1] Evolution of an Adaptive Middleware Exploiting Architectural Reflection. *Francesca Arcelli,* and *Claudia Raibulet* (Università degli Studi di Milano-Bicocca, Italy).

Claudia Raibulet gave the talk.

[3] An Aspect-Oriented Adaptation Framework for Dynamic Component Evolution. *Javier Cámara Moreno, Carlos Canal, Javier Cubo* (University of Málaga, Spain), and *Juan M. Murillo Rodriguez* (University of Extremadura, Spain).

Javier Cámara Moreno gave the talk.

[12] An Aspect-Aware Outline Viewer. *Michihiro Horie*, and *Shigeru Chiba* (Tokyo Institute of Technology, Japan).

Michihiro Horie has given the talk.

Session on Technological Limits for Software Evolution

The papers in this session explore the technological limits of the AOP and reflective techniques to support software evolution. Hidehiko Masuhara (University of Tokyo, Japan) has lead the session. Four papers have been presented:

[18] Solving Aspectual Semantic Conflicts in Resource Aware Systems. *Arturo Zambrano, Tomás Vera,* and *Silvia Gordillo* (University of La Plata, Argentina).

Arturo Zambrano has given the talk.

[8] Statement Annotations for Fine-Grained Advising. *Mark Eaddy*, and *Alfred Aho* (Columbia University, USA).

Mark Eaddy gave the talk.

[9] Dynamic Refactorings: Improving the Program Structure at Run-time. *Peter Ebraert* and *Theo D'Hondt* (Vrije Universiteit Brussel, Belgium).

Peter Ebraert gave the talk.

[13] Implementing Bounded Aspect Quantification in AspectJ. *Christian Kästner, Sven Apel,* and *Gunter Saake* (Otto-von-Guericke-Universität Magdeburg, Germany).

Christian Kästner gave the talk.

2 Software Evolution Trends: The Organizers' Opinion

The authors, with this report, would like to go beyond the mere presentation of statistical and generic information related to the workshop and to its course. They try to speculate about the current state of art of the research in the field and to evaluate the role of reflection, AOSD and meta-data in the software evolution.

Can or Cannot the AOP Support the Software Evolution?

Comment by Walter Cazzola (*Università di Milano, Italy*)

In [2], software evolution is defined as a kind of software maintenance that takes place only when the initial development was successful. The goal consists of adapting the application to the ever changing user requirements and operating environment.

Software systems are often asked for promptly evolving to face critical situations such as to repair security bugs, to avoid the failure of critical devices and to patch the logic of a critical system. It is fairly evident the necessity of improving the software adaptability and its promptness without impacting on the activity of the system itself. This statement brings forth to the need for a system to manage itself to some extent, to dynamically inspect component interfaces, to augment its application-specific functionality with additional properties, and so on.

Independently of the mechanism adopted for planning the evolution, it requires a mechanism that permits of concreting the evolution on the running system. In particular this mechanism should be able of i) extruding the code interested by the evolution from the whole system code, ii) carrying out the patches required by the planned evolution on the located code. Often, both these steps must occur without compromising the system stability and the services availability (that is, without stopping the system providing them).

AOP [14] provides some mechanisms (join points, pointcut and aspect weaving) that allow of modifying the behavior and the structure of an application. The AOP technology better addresses functionality that crosscut the whole implementation of the application. Evolution is a typical functionality that crosscut the code of many objects in the system. Moreover, the AOP technology seems suitable to deal with the detection of the code to evolve and with the instrumentation of such a code.

From AOP characteristics, it is fairly evident that AOP has the potential for providing the necessary tools for instrumenting the code of a software system, especially when

aspects can be plugged and unplugged at run-time. Pointcuts should be used to pick out a region of the code involved by the evolution, whereas the advices should be used to define how the selected region of code should evolve. Weaving such an aspect on the running system should either inject new code or manipulate the existing code, allowing the dynamic evolution of the system.

Unfortunately, to define pointcuts that point out the code interested by the evolution is a hard task because such modifications can be scattered and spread all around the code and not confined to a well-defined area that can be taken back to a method call. Moreover the changes could entail only part of a statement, e.g., the exit condition of a loop or an expression, and not the entire statement.

The AOP technology could be the right approach to deal with the evolution concern but some scenarios are difficult to administrate with the current aspect-oriented frameworks. The main issues that obstacle the use of the AOP approaches are: the *granularity* of the requested manipulation and the *locality* of the code to manipulate. The necessary granularity for the pointcut is too fine, traditional join point models refer to method invocation in several way whereas we want to be able to manipulate a single statements or a group of statements in somehow related. This means that we can manipulate the method execution at its beginning and at its ending but we cannot alter its computational flow requiring the execution of another statement between two statements of its body.

These problems are due to the poor expressiveness of the pointcut definition languages and of the related join point models provided by most of the actual AOP frameworks. Nowadays AOP languages provide very basic pointcut definition languages that heavily rely on the structure and syntax of the software application neglecting its semantics. The developer has to identify and to specify in the correct way the pointcut by using, what we call the linguistic pattern matching mechanism; it permits of locating where an advice should be applied by describing the join points as a mix of references to linguistic constructs (e.g., method calls) and of generic references to their position, e.g., before or after it occurs. Therefore, it is difficult to define generic, reusable and comprehensible pointcuts that are not tailored on a specific application. Moreover, current join point model is too abstract. Join points are associated to a method call whereas a finer model should be adopted to permit of altering each single statement.

Therefore, to benefit from the AOP technology, this one and the underlying join point models, have to support a finer granularity of the join point model with a pointcut declaration language that abstracts from the syntax and deals with the semantics/behavior of the application. A few attempts in this direction are under development [5, 11, 15].

Unanticipated Software Evolution: Does It Exist?

Comment by Shigeru Chiba (*Tokyo Institute of Technology, Japan*)

An interesting issue on software evolution is what kind of evolution we must support. For example, if we use AOP or reflection, we can extend our applications so that they can fit new requirements. However, if we anticipate future changes of requirements when we design the applications, we can prepare against those future changes. We will define extra interfaces for better modularity. We may apply some design patterns such as the visitor pattern. If we can perfectly anticipate future changes, we will not need

AOP or reflection. We will be able to prepare within confines of existing technologies such as object-oriented programming.

Some readers might say the value of AOP and reflection is that they provide better syntax. Although preparation by existing technologies is often complicated, preparation by AOP or reflection is much simpler or unnecessary. However, I think that the real value of AOP and reflection is that they can be used for implementing unanticipated evolution. If we use AOP or reflection, we do not have to prepare against future changes at all when we design and implement a first version of our applications. On the other hand, existing technologies such as object oriented programming do not address unanticipated evolution. To adapt applications to new requirements, those technologies force developers to edit and modify source code of the applications. I think that the study in this research field should consider differences between anticipated evolution and unanticipated one.

AOP vs. Reflection for Evolution

Comment by Gunter Saake (*University of Magdeburg, Germany*)

Evolution of software artifacts means continuous adaptation to a changing environment and changing requirements. However, this process can be performed on different abstraction levels and in different kinds of environments. Important dimensions for adaptation are the following:

- Anticipated evolution can be foreseen at software production time. Typical techniques can be built-in parametrization and reconfiguration methods, e.g., based on components or design patterns. Unanticipated adaptation means the reaction on unforeseen changes, which is of course much harder.
- The adaption process can be performed manually as part of the maintenance process. Automatic adaptation is possible in some situations but requires the detection of environmental changes and planning and validation of adaption steps by the software system itself.
- Adaptation can take place on different abstraction levels, from the model level (for example UML) down to the code level.

These dimensions are not independent and usually they overlap. For example, planning of an adaptation requires in most case a semantic-rich representation of the system and it is hard to plan on the syntactic code-level.

How do AOSD and reflection fit to this general picture? Figure 1 shows the current situation for AOP and reflection techniques.

AOP techniques can directly manipulate (syntactic) code. However, since common AOP techniques are based on the syntactic structure of the program it is hard to use them for semantic based tasks which are used for a fully automatic adaption. AOP is general enough to allow manual adaptation to unforeseen changes.

Reflection and meta-programming on the other hand is based on an internal model of the software. Current technology can use reflection for automatic reconfiguration at runtime, which can be used for reaction on anticipated changes. Unanticipated changes are hard to cope with basic reflection techniques, because that would require besides the

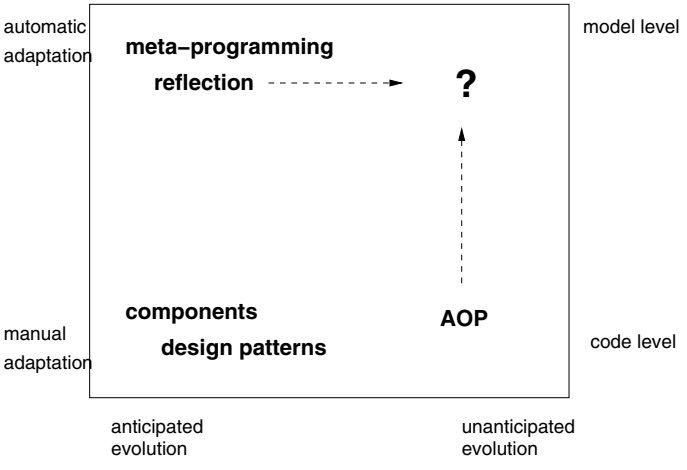

Fig. 1. AOP versus Reflection

internal model of the software also a complete model of the environment which has to be general enough to capture all unforeseen changes. Also it is hard to use these techniques for the code level.

So, AOP and reflection are currently playing in different corners of the adaptation techniques landscape. What is their interplay?

First of all, AOP techniques can be used to prepare software for reflection. Pointcuts can identify points of interaction between meta-level and base-level in the code. This can be used to built in the necessary communication between the base-level objects and the meta-level objects.

Second, a combination of techniques from both approaches may shift one of them a little bit toward the "hard" corner (automatic reaction to unanticipated changes [4]). Pointcuts related to semantic properties as described for example in [15] in combination with dynamic weaving may become a (semi) automatic technique for performing semantic adaptation resulting of a plan generated by a reflective analysis of the changed environment.

Natural Selection and System Microevolution: A New Modularity for RAM-SE'07?

Comment by Yvonne Coady (*University of Victoria, Canada*)

Microevolution can be thought of on a small scale, within a single population, while *macroevolution* transcends the boundaries of a single species. Despite their differences, evolution on both of these scales relies on the same mechanisms of evolutionary change, such as *natural selection*. Natural selection is the differential survival of a subset of a population over other subsets. In terms of software, we can apply theories of microevolution within single, open source systems, and can see how natural selection drives

evolution. For example, if we consider a single population such as Linux, natural selection determines which kernel extensions survive as the system evolves. But what factors determine survival of the fittest? Why do some extensions survive while others do not?

Companies and governments alike are relying upon evolution of Linux to reduce the cost and time-to-market of developing WiFi routers, cell phones, and telecommunications equipment and of running services on specialized servers, clusters, and high-performance supercomputers. One important benefit of starting from the same Linux source for all of these systems is that developers have access to this open source project and can easily create variants that are directly tailored for their application domain. Major evolutionary variants of a mainline Linux kernel are typically created by incrementally adding kernel extensions by fine grained instrumentation of source.

Current best practice is to implementing such extensions is by directly instrumenting the code-base of a mainline Linux kernel. Upon completing the development of an extension, developers extract the instrumentation as a whole with tools such as `diff` to create a "patch". They can then share the resulting patch with others, who in turn integrate the extension into their version of the kernel using tools such as `patch`.

It is interesting to note that patch sets must be approved before they can be incorporated into a mainline version of the kernel. As a result, all instrumentation must be reasoned about and tested comprehensively. The problem is that, though well localized, patch sets themselves are almost impossible to reason about at a high-level, as their granularity of interaction is expressed in terms of line numbers with very little surrounding context.

Given that there are a number of highly invasive patch sets that have yet to be mainlined, it is indeed reasonable to assume that it is ultimately their lack of explicit representation of interaction with the rest of the system that may be an ultimate determining factor in survival. That is, highly invasive patches are exterminated due to natural selection, meaning that *modularity is indeed an evolutionary factor in system microevolution*.

For most of these extensions however, the answer does not come in the form of aspects as we know them today. This is due to the aggressive nature of the required refactoring necessary to reveal suitable join points given current join point models. Aggressive refactoring is not an option in this context, within this domain, as it would make the entire patch set less likely to be adopted into a mainline kernel.

Others have suggested a need for a finer-grained join point model, and this challenge problem of microevolution seems to echo that same call. At the same time however, it is necessary that these new mechanisms provide substantially more semantic leverage than that which is currently available in tools such as `diff` and `patch`. The question as to how a new modularity can be developed to simultaneously provide (1) fine-grained points of interaction, along with (2) a high enough level of abstraction upon which semantic conflicts can be reasoned about, is an interesting challenge problem for participants at RAM-SE'07!

3 Final Remarks

The workshop's main goal was to bring together researchers interested in the field and to allow them to get to know each other and each other's work. The workshop lived

up to its expectations, with high-quality submissions and presentations, and lively and stimulating discussions. We hope participants found the workshop interesting and useful, and encourage them to finalize their position papers and submit them as full papers to international conferences interested in the topics of this workshop.

Acknowledgements. We wish to thank Awais Rashid, Theo D'Hondt, Mario Südholt and Hidehiko Masuhara both for their interest in the workshop, and for their help during the workshop as chairmen and speakers. We wish also to thank all the researchers that have participated to the workshop.

We have also to thank the Department of Informatics and Communication of the University of Milan, the Department of Mathematical and Computing Sciences of the Tokyo institute of Technology and the Institute für Technische und Betriebliche Informationssysteme, Otto-von-Guericke-Universität Magdeburg for their various supports.

References

1. Francesca Arcelli and Claudia Raibulet. Evolution of an Adaptive Middleware Exploiting Architectural Reflection. In Walter Cazzola, Shigeru Chiba, Yvonne Coady, and Gunter Saake, editors, *Proceedings of ECOOP'2006 Workshop on Reflection, AOP and Meta-Data for Software Evolution (RAM-SE'06)*, pages 49–58, Nantes, France, July 2006.
2. Keith H. Bennett and Václav T. Rajlich. Software Maintenance and Evolution: a Roadmap. In Anthony Finkelstein, editor, *The Future of Software Engineering*, pages 75–87. ACM Press, 2000.
3. Javier Cámara Moreno, Carlos Canal, Javier Cubo, and Juan M. Murillo Rodriguez. An Aspect-Oriented Adaptation Framework for Dynamic Component Evolution. In Walter Cazzola, Shigeru Chiba, Yvonne Coady, and Gunter Saake, editors, *Proceedings of ECOOP'2006 Workshop on Reflection, AOP and Meta-Data for Software Evolution (RAM-SE'06)*, pages 59–70, Nantes, France, July 2006.
4. Walter Cazzola, Ahmed Ghoneim, and Gunter Saake. Software Evolution through Dynamic Adaptation of Its OO Design. In Hans-Dieter Ehrich, John-Jules Meyer, and Mark D. Ryan, editors, *Objects, Agents and Features: Structuring Mechanisms for Contemporary Software*, Lecture Notes in Computer Science 2975, pages 69–84. Springer-Verlag, Heidelberg, Germany, July 2004.
5. Walter Cazzola, Sonia Pini, and Massimo Ancona. AOP for Software Evolution: A Design Oriented Approach. In *Proceedings of the 10th Annual ACM Symposium on Applied Computing (SAC'05)*, pages 1356–1360, Santa Fe, New Mexico, USA, on 13th-17th of March 2005. ACM Press.
6. Walter Cazzola, Sonia Pini, and Massimo Ancona. Design-Based Pointcuts Robustness Against Software Evolution. In Walter Cazzola, Shigeru Chiba, Yvonne Coady, and Gunter Saake, editors, *Proceedings of ECOOP'2006 Workshop on Reflection, AOP and Meta-Data for Software Evolution (RAM-SE'06)*, pages 35–45, Nantes, France, July 2006.
7. Alan Cyment, Nicolas Kicillof, Rubén Altman, and Fernando Asteasuain. Improving AOP Systems' Evolvability by Decoupling Advices from Base Code. In Walter Cazzola, Shigeru Chiba, Yvonne Coady, and Gunter Saake, editors, *Proceedings of ECOOP'2006 Workshop on Reflection, AOP and Meta-Data for Software Evolution (RAM-SE'06)*, pages 9–21, Nantes, France, July 2006.

8. Mark Eaddy and Alfred Aho. Statement Annotations for Fine-Grained Advising. In Walter Cazzola, Shigeru Chiba, Yvonne Coady, and Gunter Saake, editors, *Proceedings of ECOOP'2006 Workshop on Reflection, AOP and Meta-Data for Software Evolution (RAM-SE'06)*, pages 89–99, Nantes, France, July 2006.
9. Peter Ebraert and Theo D'Hondt. Dynamic Refactorings: Improving the Program Structure at Run-time. In Walter Cazzola, Shigeru Chiba, Yvonne Coady, and Gunter Saake, editors, *Proceedings of ECOOP'2006 Workshop on Reflection, AOP and Meta-Data for Software Evolution (RAM-SE'06)*, pages 101–110, Nantes, France, July 2006.
10. Robert E. Filman and Daniel P. Friedman. Aspect-Oriented Programming is Quantification and Obliviousness. In *Proceedings of OOPSLA 2000 Workshop on Advanced Separation of Concerns*, Minneapolis, USA, October 2000.
11. Jeff Gray, Janos Sztipanovits, Douglas C. Schmidt, Ted Bapty, Sandeep Neema, and Aniruddha Gokhale. Two-Level Aspect Weaving to Support Evolution in Model-Driven Synthesis. In Robert E. Filman, Tzilla Elrad, Siobhàn Clarke, and Mehmet Akşit, editors, *Aspect-Oriented Software Development*, chapter 30, pages 681–709. Addison-Wesley, October 2004.
12. Michihiro Horie and Shigeru Chiba. An Aspect-Aware Outline Viewer. In Walter Cazzola, Shigeru Chiba, Yvonne Coady, and Gunter Saake, editors, *Proceedings of ECOOP'2006 Workshop on Reflection, AOP and Meta-Data for Software Evolution (RAM-SE'06)*, pages 71–75, Nantes, France, July 2006.
13. Christian Kästner, Sven Apel, and Gunter Saake. Implementing Bounded Aspect Quantification in AspectJ. In Walter Cazzola, Shigeru Chiba, Yvonne Coady, and Gunter Saake, editors, *Proceedings of ECOOP'2006 Workshop on Reflection, AOP and Meta-Data for Software Evolution (RAM-SE'06)*, pages 111–122, Nantes, France, July 2006.
14. Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-Oriented Programming. In *11th European Conference on Object Oriented Programming (ECOOP'97)*, Lecture Notes in Computer Science 1241, pages 220–242, Helsinki, Finland, June 1997. Springer-Verlag.
15. Klaus Ostermann, Mira Mezini, and Christoph Bockisch. Expressive Pointcuts for Increased Modularity. In Andrew P. Black, editor, *Proceedings of the 19th European Conference on Object-Oriented Programming (ECOOP'05)*, LNCS 3586, pages 214–240, Glasgow, Scotland, July 2005. Springer.
16. Miguel Ángel Pérez Toledano, Amparo Navasa Martinez, Juan M. Murillo Rodriguez, and Carlos Canal. Making Aspect Oriented System Evolution Safer. In Walter Cazzola, Shigeru Chiba, Yvonne Coady, and Gunter Saake, editors, *Proceedings of ECOOP'2006 Workshop on Reflection, AOP and Meta-Data for Software Evolution (RAM-SE'06)*, pages 23–34, Nantes, France, July 2006.
17. Awais Rashid. Aspects and Evolution: The Case for Versioned Types and Meta-Aspect Protocols. In Walter Cazzola, Shigeru Chiba, Yvonne Coady, and Gunter Saake, editors, *Proceedings of ECOOP'2006 Workshop on Reflection, AOP and Meta-Data for Software Evolution (RAM-SE'06)*, pages 3–5, Nantes, France, July 2006.
18. Arturo Zambrano, Tomás Vera, and Silvia Gordillo. Solving Aspectual Semantic Conflicts in Resource Aware Systems. In Walter Cazzola, Shigeru Chiba, Yvonne Coady, and Gunter Saake, editors, *Proceedings of ECOOP'2006 Workshop on Reflection, AOP and Meta-Data for Software Evolution (RAM-SE'06)*, pages 79–88, Nantes, France, July 2006.

A Workshop Attendee

The success of the workshop is mainly due to the people that have attended it and to their effort to participate to the discussions. The following is the list of the attendees in alphabetical order.

Name	Affiliation	Country	e-mail
Altman, Rubén	Universidad de Buenos Aires	Argentina	ruben.altman@miva.com
Bernard, Emmanuel	jBoss Europe	France	
Beurton-aimar Marie	LaBRI, Université de Bordeaux 1	France	aimar@labri.u-bordeaux.fr
Cámara Moreno, Javier	Universidad de Málaga	Spain	jcamara@lcc.uma.es
Cazzola, Walter	Università degli Studi di Milano	Italy	cazzola@dico.unimi.it
Chiba, Shigeru	Tokyo Institute of Technology	Japan	chiba@is.titech.ac.jp
Cyment, Alan	Universidad de Buenos Aires	Argentina	acyment@yahoo.com
David, Pierre-Charles	France Télécom R&D	France	pierrecharles.david@francetelecom.com
D'Hondt, Theo	Vrij Universiteit Brussel	Belgium	tjdhondt@vub.ac.be
Dubochet, Gilles	École Polytechnique Fédérale de Lausanne	Switzerland	gilles.dubochet@epfl.ch
Eaddy, Mark	Columbia University	USA	eaddy@cs.columbia.edu
Ebraert, Peter	Vrij Universiteit Brussel	Belgium	pebraert@vub.ac.be
Horie, Michihiro	Tokyo Institute of Technology	Japan	horie@csg.is.titech.ac.jp
Kästner, Christian	University of Magdeburg	Germany	christian.k@stner.de
Masuhara, Hidehiko	University of Tokyo	Japan	masuhara@graco.c.u-tokyo.ac.jp
Meister, Lior	Rafael	Israel	meister@rafael.co.il
Nguyen, Ha	École des Mines de Nantes	France	ha.nguyen@emn.fr
Pérez Toledano, Miguel Ángel	University of Extremadura	Spain	toledano@unex.es
Pini, Sonia	Università degli Studi di Genova	Italy	pini@disi.unige.it
Raibulet, Claudia	Università di Milano Bicocca	Italy	raibulet@disco.unimib.it
Rashid, Awais	Lancaster University	United Kingdom	marash@comp.lancs.ac.uk
Saake, Gunter	University of Magdeburg	Germany	saake@iti.cs.uni-magdeburg.de
Shakil Khan, Safoora	Lancaster University	United Kingdom	safoorashakil@hotmail.com
Stein, Krogdahl	University of Oslo	Norway	stein.krogdahl@ifi.uio.no
Südholt, Mario	École des Mines de Nantes	France	sudholt@emn.fr
Tsadock, Carmit	Rafael	Israel	
Zambrano, Arturo	Universidad de La Plata	Argentina	arturo@sol.info.unlp.edu.ar