Software Evolution: A Trip Through Reflective, Aspect, and Meta-data Oriented Techniques

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

[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] Institute für Technische und Betriebliche Informationssysteme,
Otto-von-Guericke-Universität Magdeburg,
Magdeburg, Germany
saake@iti.cs.uni-magdeburg.de

Abstract. Previous workshops related to aspect oriented software development, reflection organized at previous ECOOP conferences (e.g., RMA'00[1].and AOMMeT'01[2].) and conferences on the same topics (Reflection'01 and AOSD since 2002) have pointed out the growing interest on these topics and their relevance in the software evolution as techniques for code instrumentation. Very similar conclusions can be drawn by reading the contributions to the workshops on unanticipated software evolution (USE 2002 and USE 2003[3].).

Following the example provided by these venues, the RAM-SE (Reflection, AOP and Meta-Data for Software Evolution) workshop has provided an opportunity for researchers with a broad range of interests in reflective techniques and aspect-oriented software development to discuss recent developments of such a techniques in application to the software evolution.

The workshop main goal was to encourage people to present works in progress. These works could cover all the spectrum from theory to practice. To ensure creativity, originality, and audience interests, participants have been selected by the workshop organizers on the basis of 5-page position paper. We hope that the workshop will help them to mature their ideas and to improve the quality of their future publications based on the presented work.

The workshop proceedings are available as research report C-186 of the Department of Mathematical and Computing Sciences of the Tokyo Institute of Technology and freely downlodable from the workshop web site[4].

[1] Details at http://www.disi.unige.it/RMA2000.html
[2] Details at http://ecoop2001.inf.elte.hu/workshop/AOMMeT-ws.html
[3] Details at http://www.joint.org/use/
[4] RAM-SE04 Web Site: http://homes.dico.unimi.it/RAM-SE04.html

J. Malenfant and B.M. Østvold (Eds.): ECOOP 2004, LNCS 3344, pp. 118–132, 2004.

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 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.

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 together 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:

- reflective middleware and environments for software evolution;
- adaptative software components;
- feature-oriented adaptation;
- aspect interference and composition for software evolution;
- evolution and adaptability;

- MOF, code annotations and other meta-data facilities for software evolution;
- intercession and introspection;
- software evolution tangling 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-SE04.html`.

The proceedings of the event is also available as research report C-186 of the Dept. of Mathematical and Computing Sciences of the Tokyo Institute of Technology.

The workshop was a full day meeting organized in four sessions. Each session has been characterized by a dominant topic that perfectly describes the presented papers and the related discussions. The four dominant topics were: *reflective middleware for software evolution*, *software evolution and refactoring*, *join points and crosscutting concerns for software evolution*, and *parametric aspects and generic aspect languages*. 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 Yvonne Coady, Joseph W. Yoder, Günter Kniesel 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

To an occasional reader who would like to deepen his(her) knowledge about the topics of this workshop (that is, to learn more about reflection, aspect-oriented programming and software evolution), we suggest to read the following basic contributions:

- Pattie Maes. Concepts and Experiments in Computational Reflection. In *Proceedings of the 2nd Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'87)*, pages 147–156, Orlando, Florida, USA, October 1987. ACM.
- 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.
- 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.

Whereas, to learn more about the use of reflective or aspect-oriented techniques in the software evolution and maintenance we suggest to look at the following proceedings and books:

- Walter Cazzola, Robert J. Stroud, and Francesco Tisato, editors, *Reflection and Software Engineering*, LNCS 1826. Springer, Heidelberg, Germany, June 2000.
- Akinori Yonezawa and Satoshi Matsuoka, editors. *Proceedings of 3rd International Conference on Metalevel Architectures and Separation of Crosscutting Concerns (Reflection'2001)*, LNCS 2192. Kyoto, Japan, September 2001. Springer.
- The Proceedings of the AOSD Conferences from 2002 to 2004. Available from `http://aosd.net/archive/index.php`.

Besides, to keep up to date with the evolution of the software evolution research area we suggest to consult the following page:

- Program-Transformation.org:

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

which collects a lot of useful links related to software evolution (in a general sense).

1 Workshop Overview: Session by Session

In this session of the report we gather together the opinions of the session chairmen (and woman) relatively to the session and the panel they have lead, and their considerations about the future trends.

Session on Reflective Middleware for Software Evolution

Summary by Yvonne Coady (Session Chair, *University of Victoria, Canada*)

There were three papers in the session, all related to reflective middleware for the software evolution.

[2] Reflections on Programming with Grid Toolkits. *Emiliano Tramontana* (Università di Catania, Italy) and *Ian Welch* (Victoria University of Wellington, New Zealand).

Emiliano Tramontana gave the talk.

This paper presents an approach for developing Grid applications. In this approach, developers need not include any code handling toolkit specific concerns nor adaptation for changing conditions of the environment. This proposed solution consists of two aspects. Firstly, applications are developed in a centralised manner and automatically transformed to be distributed into the Grid, where they use a suitable communication primitive. Secondly, an open implementation of Globus is used to enable dynamic changes at run-time of the communication infrastructure and the service container.

Panel session: During the panel session Emiliano was asked if the development of the client and the server side of a Grid system took place all at once or in a coordinated way. His answer was that they are developed separately and the approach proposes to have only a coordinated way to add/remove nonfunctional concerns to/from both sides.

Follow-up questioning then asked whether just the proposed reflective Grid, alone, could satisfy both the need for adapting the application and the middleware. Emiliano's answer was that distribution would still need to be added into the application before its execution could be affected/adapted by the reflective Grid, especially when the application is initially sent for execution to a host that does not provide all the reflective middleware.

[3] Using Aspects to Make Adaptive Object-Models Adaptable. *Ayla Dantas, Paulo Borba* (Federal University of Pernambuco, Brazil) and *Joe Yoder* and *Ralph Johnson* (University of Illinois at Urbana-Champaign, USA).

Paulo Borba gave the talk.

This paper argues that AOP and adaptive object-models (AOM) play complementary roles for structuring adaptive applications. Whereas AOM support flexible dynamic adaptations by representing business rules and entities as data, AOP modularizes the crosscutting adaptation code. In addition to the results presented in the paper, further metrics were presented to quantifiably compare the approaches.

Panel session: The first part of the discussion was about the importance of metrics to provide a meaningful comparison, and the particularly nice job the authors did in the presentation. Follow-up discussion included issues such as the use of generic aspects for implementing adaptation concerns. The general consensus at the end of the discussion was that people thought they would be indeed useful to have reusable adaptation aspects.

[4] RAMSES: a Reflective Middleware for Software Evolution. *Walter Cazzola* (Università di Milano, Italy) and *Ahmed Ghoneim* and *Gunter Saake* (University of Magdeburg, Germany).

Ahmed Ghoneim gave the talk.

This paper presents a middleware for dynamically evolving and validating consistency of software systems against run-time changes. This middleware is based on a reflective architecture that provides objects with the ability to dynamically change their behavior by using design information. The evolution takes place in two steps: first a meta-object (the evolutionary meta-object) plans a possible evolution against the detected external events then another meta-object (the consistency checker meta-object) validates the feasibility of the proposed plan before really evolving the system.

Panel session: The first question involved the use of XMI schemas and their use. XMI is XML Meta Interchange, and this approach extracts XMI schemes from the UML models, and then uses this data at run-time. A second question involved the ability to drive both evolution and consistency. Both the meta-objects in this approach (evolutionary and consistency checker) inherit engine, which includes a set of rules. Script rules are used for modifying the reified XMI for run-time events and check the modified XMI.

Session on Software Evolution and Refactoring

Summary by Joe Yoder (Session Chair, *The Refactory Inc.*)

This session presented four talks on how to handle dynamic aspects and adaptability with AOP. These papers outlined some interesting approaches to adapt to changing

requirements which include frameworks for dynamically refactoring separation of concerns and dynamic ways to deal with runtime adaptability.

The outlines of the talks are as follows:

[5] *Ruzanna Chitchyan* (Lancaster University, UK) presented the work on "AOP and Reflection for Dynamic Hyperslices". This paper described a model for dynamic hyperslices which uses a particular aspect-oriented approach - Hyperspaces - for decomposition and reflection as a means for composition of software modules. This model allows for structured, dynamic, incremental change introduction and rollback, thus, supporting run-time evolution yet preserving component modularity. The applicability of the model is illustrated through a schema adaptation scenario.

[6] *Peter Ebraert* (Vrij Universiteit Brussel) presented the work on "A Reflective Approach to Dynamic Software Evolution". The paper outlines a solution that allows systems to remain active while they are evolving. The approach goes out from the principle of separated concerns and has two steps. In the first step, they make sure that the system's evolvable concerns are cleanly separated by proposing aspect mining and static refactorings for separating those concerns. In the second step, they allow every concern to evolve separately. A preliminary reflective framework that allows dynamic evolution of separate concerns is outlined.

[7] *Yvonne Coady* (University of Victoria) presented the work on "OASIS: Organic Aspects for System Infrastructure Software Easing Evolution and Adaptation through Natural Decomposition". The OASIS project explores the potential of aspects to naturally shape crosscutting system concerns as they grow and change. This paper describes ongoing work to modularize evolving concerns within high-performance state-of-the-art systems software, and outlines some of the major challenges that lie ahead within this domain.

[8] *Yoshiki Sato* (Tokyo Institute of Technology) presented the work on "Negligent Class Loaders for Software Evolution". This paper presents a negligent class loader, which can relax the version barrier between class loaders for software evolution. The version barrier is a mechanism that prevents an object of a version of a class from being assigned to a variable of another version of that class. In Java, if a class definition (i.e. class file) is loaded by different class loaders, different versions of the class are created and regarded as distinct types. If two class definitions with the same class name are loaded by different loaders, two versions of the class are created and they can coexist while they are regarded as distinct types.

The following highlight the main questions the presenters and the audience discussed after the presentations:

- How do you ensure consistency when making changes at run-time? Sometimes the model needs to clearly define the adaptability parameters up front, thus managing consistency.
- How do you deal with performance issues while dynamically loading new versions with the class loader? Some well known techniques were discussed such as static code translation, just-in-time hook insertion and modified JVM.

In summary, providing dynamic ways to adapt to changing requirements has some great potential benefits for software developers. Using some well-known reflection techniques in conjunction with AOP can really assist with this by separating what changes

from what doesn't and by allowing ways to *refactor* your systems by dynamically applying new weavings for the evolving separation of concerns during run-time. Also, by using AOP to separate adaptable concerns shows promise for assisting with run-time adaptations.

Session on Join Points and Crosscutting Concerns for SW Evolution

Summary by Günter Kniesel (Session Chair, *University of Bonn, Germany*)

This and next session were dedicated to the relation between software evolution and aspect oriented software development (AOSD). This session focused on the influence of join point models, a central concept of AOSD, on evolution and evolvability. The three papers in this session addressed this issue from different perspectives:

[9] *Nicolas Pessemier* (INRIA, France) presented the paper "Components, ADL & AOP: Towards a Common Approach". In his talk he motivated the need for unification and explained the approach taken in the FRACTAL project. FRACTAL integrates the notion of components with ports and port binding from the domain of architecture description languages with the AOSD specific notion of join points.

[10] *Naoyasu Ubayashi* (Kyushu Institute of Technology, Japan) presented the paper "An AOP Implementation Framework for Extending Join Point Models". He argued that current join point models are too rigid and extensible join point models are needed instead to foster unanticipated evolution. Then he introduced a reflective API for defining join points, which gives programmers complete control over join points and the kind of weaving actions to be performed at a specific join point.

[11] *Sonia Pini* (University of Genova, Italy) presented the paper "Evolving Pointcut Definition to Get Software Evolution". She explained the need for a formal model of join points and showed that no satisfactory one is available so far for dynamic join points. Then she introduced the use of UML statechart diagrams as a precise specification of dynamic join points. She showed that evolving the diagram concisely captures evolution scenarios that are hard to express otherwise (in AspectJ, for instance).

In the subsequent discussion the first paper attracted no critical remarks regarding the chosen approach or its utility. The asked questions only requested clarification of technical details of the used join point model.

For the second paper its reliance on a reflective API within an AOP approach triggered a lively discussion. Some attendants argued that such a combination should be avoided since AOP had been motivated in the first place by the desire to provide a simpler abstraction than full reflection. Others pointed out that it may be worthwhile to go "back to the roots" and review "old" design decisions in the light of new experience.

The third paper was received with a mix of positive surprise about the simplicity and elegance of the statechart based join point specification and some skepticism about the compositionality of different specifications and the scalability of the approach.

Session on Parametric Aspects and Generic Aspect Languages

Summary by Hidehiko Masuhara (Session Chair, *University of Tokyo, Japan*)

This session had three talks on AOP languages and frameworks to develop software systems that are more robust to software evolution.

By supporting separation of crosscutting concerns, aspect-oriented programming languages, AspectJ in particular, are known to be useful to develop more reusable programs. For example, Hannemann and Kiczales presented that some of the GoF design patterns can be provided as reusable aspects in AspectJ [12]. However, the experiences also revealed that current AOP languages are not sufficient for providing certain kinds of reusable aspects. The three talks in this session addressed the problem by introducing aspects whose pointcuts, introductions and advices are parameterized. These generic definitions can be tailored to different applications by instantiating some of their parameters. The outlines of the talks are as follows (in the order presented):

[13] *Jordi Alvarez Canal* (Universitat Oberta de Catalunya, Spain) presented the work on "Parametric Aspects: A Proposal", in which parametric aspects can take type parameters, and are useful to define abstract factory patterns in a domain-independent way and simple Enterprise Java Beans.

[14] *Philip Greenwood* (Lancaster University, UK) presented the work on "Dynamic Framed Aspects for Dynamic Software Evolution", in which aspects support dynamic changes in software systems. The approach is based on the Framed Aspects to parameterize the aspects for a specific use.

[15] *Tobias Rho* (University of Bonn, Germany) presented the work on "Evolvable Pattern Implementations need Generic Aspects", which points out that the evolution of design patterns is rarely supported in existing AOP languages. He also introduced the language LogicAJ, an extension of AspectJ in which homogeneously generic aspects are supported. LogicAJ is based on the use of logic meta variables as placeholders for arbitrary program elements, a concept borrowed from logic programming languages.

After the talks, the presenters and the audience discussed the following questions:

- How those proposals are different from each other? While all three talks proposed some form of generic aspects they differ in terms of their primary target, syntax, generality, and so forth. LogicAJ is an attempt to provide general framework to cover various kinds of generic aspects; Parametric Aspects offer simpler syntax; and Framed Aspects are more interested in supporting product lines, rather than supporting reusable design patterns.
- How those proposals support for software evolution? Generally, they all improve modularity of software systems so that independently evolving parts can be separated from others.
- How those proposals ensure the correctness of the generated aspects? Since all those proposals generate base code from generic definitions, it might be possible to generate incorrect code. It would be better if the system could check the safety of a generic aspect and the code that instantiates the aspect into specific context

so that incorrect code is never generated[5]. All of the proposals currently check the correctness after they generated base code for specific contexts. LogicAJ aims at checking the correctness before generating aspects by extending its logical framework to static types. Both parametric aspects and framed aspects are willing to offer some means to check before generation as well.

To summarize the session, generic or parametric AOP languages and frameworks are promising means to develop more evolvable and better modularized software systems. At the same time there are also interesting challenges such as supporting rich kind of evolution and statically ensuring correctness.

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.

The Role of Reflection in Software Evolution

Comment by Walter Cazzola (*Università di Milano*)

In [16], *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, and often in an unexpected way, user requirements and operating environment.

Software evolution, as well as software maintenance, is characterized by its huge cost and slow speed of implementation. Often, software evolution implies a redesign of the whole system, the development of new features and their integration in the existing and/or running systems (this last step often implies a reboot of the system). The redesign and develop steps correspond to an economic effort from the software producer that often does not have an immediate benefit from this extra work whereas the integration step involves also the user of the system.

The chimera of the current and future trends in this discipline is to beat down the cost of evolving a system. The most recognized approach consists of minimizing the impact of the software evolution on the activity of the user and of improving the software adaptability. This statement brings forth the need for a system to manage itself to some extent, to inspect components' interfaces dynamically, to augment its application-specific functionality with additional properties, and so on. To deal with these issues many researchers are developing middleware for supporting the software evolution without affecting the activity or the property of the system that has to be evolved. Few examples of this kind of middleware have been presented also to our workshop, see [3], [7] and [4].

[5] Note that it can be checked in some situations as we see in the programming languages with polymorphic types, such as ML.

Reflection is a discipline that is steadily attracting attention within the community of object-oriented researchers and practitioners. From a pragmatic point of view, several reflective techniques and technologies lend themselves to be employed in addressing the software evolution issue. On a more conceptual level, several key reflective principles could play an interesting role as general software design and evolution principles. Even more fundamentally, reflection may provide a cleaner conceptual framework than that underlying the rather 'ad-hoc' solutions embedded in most commercial platforms and technologies, and so on. The transparent nature of reflection makes it well suited to address problems such as evolution of legacy systems, customizable software, product families, and more. The properties of transparency, separation of concerns, and extensibility supported by reflection have largely been recognized as useful for software development and design. These features seem perfect tools to aid the dynamic evolution of running systems providing the basic mechanisms for adapting (i.e., evolving) a system without directly altering or stopping the existing system.

A reflective architecture represents the perfect structure that allows running systems to adapt themselves to unexpected external events, i.e., to consistently evolve. In this kind of reflective architecture, the system running in the base-level should be the one prone to be adapted, whereas software evolution should be the nonfunctional feature realized by the meta-level system. Reflection plays a fundamental role allowing the meta-level system of inspecting and instrumenting the code of the base-level system in a transparent way and independently of the knowledge of such code. This approach, therefore, permits of developing the software evolution as a separate and independent system that could be connected at any time to the system to be adapted without any specific requirement.

In theory, this kind of middleware can provide many benefits (e.g., dynamic patching to critical failures without stopping the system and the consequently the postponement of the system redesigning) but it is not to simple and immediate to realize. To evolve the base-level system and maintain it consistent and stable, the meta-level system must face many problems. The most important are: (1) to determine which events cause the need for evolving the base-level system (2) how to react on events and the related evolutionary actions (3) how to validate the consistency and the stability of the evolved system and eventually how to undo the evolution, (4) to determine which information are need to the evolution and/or are involved by the evolution. Therefore, the future research in reflective middleware for software evolution should face these open issues and many others.

The Role of AOP in Software Evolution

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

AOP (Aspect Oriented Programming) gives us a new concept called aspects, which improve our ability for modeling and designing software. In traditional OOP (Object Oriented Programming), we must decompose software into a number of objects. A group of objects that have similar functionality is categorized as a class. We have been developing and maintaining software by using a class as a minimum unit of maintenance. However, in practice, there are usually several different means of decomposing software into objects. Developers could take multiple viewpoints for decomposition; every viewpoint would cause different decomposition. A problem is that there is no single

best viewpoint for decomposition. Some viewpoints would be good for developing or maintaining some parts of software and others would be good for doing other parts.

AOP enables us to decompose software from various viewpoints – *aspects* – as well as the dominant viewpoint represented by classes. For example, in the AspectJ language, some functionality that must be spread over several classes in a traditional OOP language can be separated into an independent module called an *aspect*. In AOP, such functionality is called a crosscutting concern.

Since AOP enables better modeling and designing of software, it is absolutely useful technology for software evolution. Well modeled and designed software is easy to maintain for evolution. If a crosscutting concern is separated as an independent module, that concern can be maintained without touching other modules. This fact will reduce maintenance costs of software evolution.

An issue actively discussed during the workshop was the maintainability of aspects themselves. Speakers pointed out that the implementation of some aspects heavily depends on other class-based modules and hence those aspects cannot be maintained independently of the other modules. In other words, aspects are in separate files but they must be edited when the classes that the aspects depend on are edited. Such aspects cannot be regarded as a truly separated module. This problem can be avoided by using abstract aspects to a certain degree, but the use of abstract aspects is a limited solution.

A better solution of this problem is to introduce parametric aspects. In fact, one session of this workshop was allocated for discussing this topic. Roughly to say, parametric aspects are generic templates that are used to generate concrete aspects in the given contexts. For example, Rho presented their language called LogicAJ, in which developers can write generic aspects using logic variables. He showed that some design patterns can be implemented as aspects in LogicAJ but the definitions of these aspects can be independent of the definitions of the classes that are woven with the aspects.

Although parametric aspects are powerful language constructs, there is still a question; are there any unique issues or techniques for parametric aspects against parametric classes? For example, the C++ template system is powerful and well-studied. The Java generics system is also. Are parametric aspect systems just straightforward variations of such parametric class systems? If not, what are design issues unique to generics for AOP? In OOP, classes can be regarded as types. Parametric classes have been studied by the researchers of type theory. Is it possible that we apply the results of such study to parametric aspects? If not, is there any theoretical background of parametric aspects?

The Role of Meta-data in Software Evolution

Comment by Gunter Saake (*University of Magdeburg*)

Future software systems should be robust and adaptive to a changing environment and to evolving requirements. This adaptiveness requires a kind of *self-awareness* — a software system has to reason about itself to be able to react on external stimuli requiring a modification.

Current software technology does not allow to build general systems reacting on arbitrary, unforeseen changes which require unanticipated modifications of system

structure and behavior. However, for restricted scenarios we can use for example reflection to react on explicit events of the environments with a modified behavior.

A key concept of such software systems are explicit meta-data. Meta-data are used to describe (relevant parts of) system architecture, system objects and behavior in a form which can be processed by a software system. Self-awareness of a system is only possible if the system has a kind of system model of itself and such a system model can be described as meta-data of that system.

This use of meta-data opens some research problems:

- How to represent meta-data? A promising approach is to use standards where appropriate, for example UML notation and XML coding. Are domain-specific languages more appropriate than general frameworks?
- What is the best abstraction level for the system model? It should be detailed enough to represent the implemented system correctly, but abstract enough to have manageable reasoning rules. One solution is to use a design model for reasoning instead of the implementation model. In this case, one has to propagate changes from the design to the implementation each time an adaptation is performed.
- Can meta-data be automatically extracted from source code and documentation? When should the extraction be done? What about continuous extraction versus incremental update of meta-data? How to filter the extracted meta-data? Can all this be done with reasonable performance? Is it possible to performing a kind of meta-data mining process on the running system itself?
- What kind of reasoning is necessary to compute a modification at run-time? A modification has to react on the changed requirements, but system functionality has to be preserved. The reasoning process therefore is driven by a certain goal with some strong constraints on the resulting solution.
- Using extracted meta-data, can we be sure about the consistency of the meta-data w.r.t. the actual system? How to detect discrepancies? How to minimize the amount of extracted information?
- Can we find a general meta-data presentation for different kinds of software systems, or do we need application specific frameworks?
- Is meta-data management done by the system itself, or does the system use services of a meta-data repository? If so, which services are core services for self-adaptive systems?
- Besides the dynamic evolution using reflection, we still have a more static kind of evolution through new releases of the software and so on. These two kinds of evolution have to be synchronized. That is, we need a backpropagation of evolution steps through reflection onto design documents, source code and documentation.

Meta-data for software evolution was not a core topic of the workshop but appears in several presentations notwithstanding that I confide that in the future, meta-data will become more and more important in the software evolution research area. Two presentations make explicit use of meta-data and reflection in their approaches. Cazzola, Ghoneim and Saake [4] propose a reflective middleware, where meta-data based on design models in UML are used for evolution. This meta-data is represented in XMI format. For reflection, two meta-objects play together: the evolutionary object plans evolution steps, and the consistency checker object proves correctness of evolution plans. Ebraert

and Tourwé [6] represent the object structure of the system explicit on the meta-level. For evolution, the meta-level evolves and the changes are propagated to the base level.

3 Final Remarks

This workshop main goal was to encourage people to present works in progress in the area of the application of reflective and aspect-oriented techniques applied to software evolution. The workshop was lively and the debates were very stimulating. We hope that the workshop has helped researchers to mature their idea and we encourage the accepted papers to be submitted to the attention of more important venues.

Acknowledgements. We wish to thank Yvonne Coady, Günter Kniesel, Hidehiko Masuhara, and Joseph Yoder both for their interest in the workshop, and for their help during the workshop and in writing part of this report. 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.

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
Alvarez Canal, Jordi	Universitat Oberta de Catalunya	Spain	jalvarezc@uoc.edu
Borba, Paulo	Federal University of Pernambuco	Brazil	phmb@cin.ufpe.br
Cazzola, Walter	Università degli Studi di Milano	Italy	cazzola@dico.unimi.it
Chiba, Shigeru	Tokyo Institute of Technology	Japan	chiba@is.titech.ac.jp
Chitchyan, Ruzanna	Lancaster University	United Kingdom	r.chitchyan@lancaster.ac.uk
Coady, Yvonne	University of Victoria	Canada	ycoady@cs.uvic.ca
Ebraert, Peter	Vrij Universiteit Brussel	Belgium	pebraert@vub.ac.be
Ghoneim, Ahmed	University of Magdeburg	Germany	ghoneim@iti.cs.uni-magdeburg.de
Greenwood, Phil	Lancaster University	United Kingdom	greenwop@comp.lancs.ac.uk
Kniesel, Günter	University of Bonn	Germany	gk@cs.uni-bonn.de
Masuhara, Hidehiko	University of Tokyo	Japan	masuhara@graco.c.u-tokyo.ac.jp
Monfort, Valérie	MDTVision	France	v-monfort@mdtvision.com
Nyström, Sven-Olof	Uppsala University	Sweden	svenolof@user.it.uu.se
Pessemier, Nicolas	INRIA	France	nicolas.pessemier@lifl.fr
Pini, Sonia	Università degli Studi di Genova	Italy	pini@disi.unige.it
Rho, Tobias	University of Bonn	Germany	rho@cs.uni-bonn.de
Saake, Gunter	University of Magdeburg	Germany	saake@iti.cs.uni-magdeburg.de
Sato, Yoshiki	Tokyo Institute of Technology	Japan	yoshiki@csg.is.titech.ac.jp
Seinturier, Lionel	INRIA	France	lionel.seinturier@lifl.fr
Störzer, Maximilian	Universität Passau	Germany	stoerzer@fmi.uni-passau.de
Tanter, Eric	University of Chile	Chile	etanter@dcc.uchile.cl
Tramontana, Emiliano	Università di Catania	Italy	tramonta@dmi.unict.it
Ubayashi, Naoyasu	Kyushu Institute of Technology	Japan	ubayashi@ai.kyutech.ac.jp
Yahiaoui, Nesrine	Université de Versailles	France	nesrine.yahiaoui@edf.fr
Yoder, Joseph W.	The Refactory Inc.	U.S.A.	joeyoder@joeyoder.com
Yonezawa, Akinori	University of Tokyo	Japan	yonezawa@yl.is.s.u-tokyo.ac.jp

References

1. Cazzola, W., Chiba, S., Saake, G., eds.: Proceedings of the 1st ECOOP Workshop on Reflection, AOP and Meta-Data for Software Evolution (RAM-SE'04). Research Report C-196 of the Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology. Preprint No. 10/2004 of Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg (2004)
2. Tramontana, E., Welch, I.: Reflections on Programming with Grid Toolkits. In Cazzola, W., Chiba, S., Saake, G., eds.: Proceedings of ECOOP'2004 Workshop on Reflection, AOP and Meta-Data for Software Evolution (RAM-SE'04), Oslo, Norway (2004) 3–8
3. Dantas, A., Yoder, J.W., Borba, P., Johnson, R.: Using Aspects to Make Adaptive Object-Models Adaptable. In Cazzola, W., Chiba, S., Saake, G., eds.: Proceedings of ECOOP'2004 Workshop on Reflection, AOP and Meta-Data for Software Evolution (RAM-SE'04), Oslo, Norway (2004) 9–19
4. Cazzola, W., Ghoneim, A., Saake, G.: RAMSES: a Reflective Middleware for Software Evolution. In Cazzola, W., Chiba, S., Saake, G., eds.: Proceedings of ECOOP'2004 Workshop on Reflection, AOP and Meta-Data for Software Evolution (RAM-SE'04), Oslo, Norway (2004) 21–26
5. Chitchyan, R., Sommerville, I.: AOP and Reflection for Dynamic Hyperslices. In Cazzola, W., Chiba, S., Saake, G., eds.: Proceedings of ECOOP'2004 Workshop on Reflection, AOP and Meta-Data for Software Evolution (RAM-SE'04), Oslo, Norway (2004) 29–35
6. Ebraert, P., Tourwé, T.: A Reflective Approach to Dynamic Software Evolution. In Cazzola, W., Chiba, S., Saake, G., eds.: Proceedings of ECOOP'2004 Workshop on Reflection, AOP and Meta-Data for Software Evolution (RAM-SE'04), Oslo, Norway (2004) 37–43
7. Gibbs, C., Coady, Y.: OASIS: Organic Aspects for System Infrastructure Software Easing Evolution and Adaptation through Natural Decomposition. In Cazzola, W., Chiba, S., Saake, G., eds.: Proceedings of ECOOP'2004 Workshop on Reflection, AOP and Meta-Data for Software Evolution (RAM-SE'04), Oslo, Norway (2004) 45–52
8. Sato, Y., Chiba, S.: Negligent Class Loaders for Software Evolution. In Cazzola, W., Chiba, S., Saake, G., eds.: Proceedings of ECOOP'2004 Workshop on Reflection, AOP and Meta-Data for Software Evolution (RAM-SE'04), Oslo, Norway (2004) 53–58
9. Pessemier, N., Seinturier, L., Duchien, L.: Components, ADL & AOP: Towards a Common Approach. In Cazzola, W., Chiba, S., Saake, G., eds.: Proceedings of ECOOP'2004 Workshop on Reflection, AOP and Meta-Data for Software Evolution (RAM-SE'04), Oslo, Norway (2004) 61–69
10. Ubayashi, N., Masuhara, H., Tamai, T.: An AOP Implementation Framework for Extending Join Point Models. In Cazzola, W., Chiba, S., Saake, G., eds.: Proceedings of ECOOP'2004 Workshop on Reflection, AOP and Meta-Data for Software Evolution (RAM-SE'04), Oslo, Norway (2004) 71–81
11. Cazzola, W., Pini, S., Ancona, M.: Evolving Pointcut Definition to Get Software Evolution. In Cazzola, W., Chiba, S., Saake, G., eds.: Proceedings of ECOOP'2004 Workshop on Reflection, AOP and Meta-Data for Software Evolution (RAM-SE'04), Oslo, Norway (2004) 83–88
12. Hannemann, J., Kiczales, G.: Design pattern implementation in Java and AspectJ. In: Proceedings of the 17th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'02), ACM Press (2002) 161–173
13. Alvarez Canal, J.: Parametric Aspects: A Proposal. In Cazzola, W., Chiba, S., Saake, G., eds.: Proceedings of ECOOP'2004 Workshop on Reflection, AOP and Meta-Data for Software Evolution (RAM-SE'04), Oslo, Norway (2004) 91–99

14. Greenwood, P., Loughran, N., Blair, L., Rashid, A.: Dynamic Framed Aspects for Dynamic Software Evolution. In Cazzola, W., Chiba, S., Saake, G., eds.: Proceedings of ECOOP'2004 Workshop on Reflection, AOP and Meta-Data for Software Evolution (RAM-SE'04), Oslo, Norway (2004) 101–110
15. Kniesel, G., Rho, T., Hanenberg, S.: Evolvable Pattern Implementations Need Generic Aspects. In Cazzola, W., Chiba, S., Saake, G., eds.: Proceedings of ECOOP'2004 Workshop on Reflection, AOP and Meta-Data for Software Evolution (RAM-SE'04), Oslo, Norway (2004) 111–126
16. Bennett, K.H., Rajlich, V.T.: Software Maintenance and Evolution: a Roadmap. In Finkelstein, A., ed.: The Future of Software Engineering. ACM Press (2000) 75–87