Reflection and Meta-Level Architectures: State of the Art, and Future Trends

Walter Cazzola[1], Shigeru Chiba[2], and Thomas Ledoux[3]

[1] DISCO - Department of Informatics, Systems, and Communication, University of Milano - Bicocca, Milano, Italy
cazzola@disco.unimib.it
[2] Institute of Information Science and Electronics, University of Tsukuba, Tsukuba, Japan
chiba@is.tsukuba.ac.jp
[3] Département Informatique, École des Mines de Nantes, Nantes, France
Thomas.Ledoux@emn.fr

Abstract Previous workshops on reflection both in ECOOP and in OOPSLA have pointed out the growing interest and importance of Reflection and Metalevel Architectures in the fields of programming languages and systems (ECOOP'98, OOPSLA'98), software engineering (OOPSLA'99) and middleware (Middleware 2000).
Following these workshops but also the conference Reflection'99 held in Saint-Malo (France), this workshop has provided an opportunity for researchers with a broad range of interests in meta-level architectures and reflective techniques to discuss recent developments in this field. It has also provided a good test-bed for preparing them to submit their works to Reflection'01.
The workshop main goal is 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 idea and improve the quality of their future publications based on the presented work.

Workshop Objectives

Over the last 15 years, Reflection and Metalevel Architectures have attracted the attention of researchers throughout computer science.

Reflective and meta-level techniques have now matured to the point where they are being used to address real-world problems in such areas as: programming languages, operating systems, databases, software engineering, distributed computing, middleware expert systems and web-computing. For example, reflective features such as separation of concerns and flexibility provide a "plug and play" environment for enabling the runtime modification of policies in middleware.

The main goal of this workshop is to address the issues arising in the definition and construction of reflective systems and to demonstrate their practical applications. To enable lively and productive discussions, participants had to present a brief introduction to their work in this area.

Objective of this workshop is to favor a working dialogue between participants in order to improve their current work and to prepare the third international conference on Reflection and Metalevel Architectures (Reflection'01).

Workshop Topics and Structure

Presentations have been accepted on topics including, but not limited to, the following:

– Reflective features of object-oriented languages (C++, Java, Smalltalk, and so on)
– Practical experience with reflective programming
– Reflective system architecture (operating systems, middleware, embedded, mobile computing, and so on)
– Reflective implementation of non-functional requirements (real-time, fault-tolerance and security issues)
– Implementation techniques (open compilers, specializers, analysis, and so on)
– Reflective software engineering (adaptive software components, MOP, AOP, meta-models, and so on)

The workshop has been organized in four sessions. Each session is characterized by a dominant topic which describes the presented papers and the related discussions. The four dominant topics are: *software engineering*, *meta-level architecture*, *middleware*, and *program translation*. 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 the role of reflection and its trend relatively to the dominant topic of the session. The discussion related to each session has been brilliantly lead respectively by Gilad Bracha, Pierre Cointe, Takuo Watanabe, and Satoshi Matsuoka.

The workshop has been very lively, the debates very stimulating, and the high number of registered attendee from several countries (see appendix A) testifies the growing interest in reflection and its potentiality and applications.

Important References

To an occasional reader who would like to learn more about reflection and related topics, we suggest to read the basic papers on the topic:

– Brian C. Smith. Reflection and Semantics in Lisp. In *Proceedings of ACM Symposium on Principles of Programming Languages*, pages 23–35, 1984.
– Pattie Maes. Concepts and Experiments in Computational Reflection. In *Proceedings of the 2nd OOPSLA'87*, pages 147–156, October 1987.

and to consult the following books on the topic:

– Gregor Kiczales, Jim des Rivières, and Daniel G. Bobrow. *The Art of the Metaobject Protocol*. MIT Press, Cambridge, Massachusetts, 1991.
– Chris Zimmerman, editor, *Advances in Object-Oriented Metalevel Architectures and Reflection*. CRC Press, Inc., Boca Raton, Florida 33431, 1996.

- Pierre Cointe, editor, Proceedings of the 2nd International Conference on Meta-Level Architectures and Reflection (Reflection'99), Springer Verlag - LNCS 1616, Saint-Malo, France, July 1999.
- Walter Cazzola, Robert J. Stroud, and Francesco Tisato, editors, *Reflection and Software Engineering*, LNCS 1826. Springer, Heidelberg, Germany, June 2000.

Besides, to keep up to date with the reflection area evolution we can consult the proceedings of the late workshops and conferences on reflection, and the following pages:

- Reflection links:

  `http://computer.org/channels/ds/middleware/RMreflection.htm`,

  and
- Reflective midddleware links:

  `http://computer.org/channels/ds/middleware/RM.htm`

which collect a lot of useful links related to reflection (in a general sense), and applied to middlewares.

Beyond to present statistical information about the workshop and general information about the current state of art of the research in the reflection area, this report gathers together the opinions of the session chairs relatively to the session they lead, and the opinions of the workshop organizers about the overall workshop and the results the workshop achieved with respect to the initial objectives.

1 Workshop Overview: Session by Session

Session on Software Engineering:
Summary by Gilad Bracha (Session Chair, *Sun Java Software*)

Two talks were given in this session:

[7] Using Reflective Features to Support Mobile Users. *Vania Marangozova and Fabienne Boyer* (INRIA, France)

Vania Marangozova gave the talk.

[9] Towards a Reflective Component Based Middleware Architecture. *Nikos Parlavantzas, Geoff Coulson, Mike Clarke, and Gordon Blair* (Lancaster University, United Kingdom)

Nikos Parlavantzas gave the talk.

[7] described a application that used an extension of Java with the ability to dynamically adapt objects behavior. Both Jacques Malenfant and I commented that the paper did not (but absolutely should) refer to the concept of delegation, which was clearly related to the language extension used.

[9] Described a middleware architecture based on OpenCOM, a COM extension. The middleware so constructed supports reflection and is itself constructed out of components.

In the discussion, I (Gilad Bracha) played devil's advocate. I challenged everyone to argue for the pragmatic value of the more advanced uses of reflection that researchers have proposed in recent years. Is there, for example, a "killer application" for their approach. After all, the basic reflective techniques of querying an object for its class and methods, fields etcetera have been used for at least 20 years. So has the capacity to reflectively modify a program by changing methods, object schemas etcetera. These capabilities are showing up in industry now and have clear value. Are more exotic uses simply past the point of diminishing returns?

For example, how valuable is the ability to trap calls to specific objects (in languages that do not naturally support this)? And if this is valuable, is reflection the correct way to support it? In languages that support delegation one can easily do this dynamically.

Various participants responded with comments on the value of malleability in general. In particular, Fábio Costa argued that the more "exotic uses" are those related to behavioural reflection, i.e., the ability to do reflection (inspection and intercession) into the underlying mechanisms used to carry out method calls, message handling, and so on. Such mechanisms normally refer to the non-functional properties associated with the method execution. By allowing the manipulation of non-functional properties using a reflective meta-interface, we achieve a degree of separation of concerns which is highly desirable.

There are, however, alternative ways of achieving this same goal, such as AOP, but reflection gives the ability to handle these issues in a dynamic way (especially, if they are not known a priori).

My response was that while this is true, we must be able to show massive (order-of-magnitude) benefits in order to displace an established (or well marketed) technology. We have repeatedly seen that being twice as good is not nearly good enough. If leading edge work requires a significantly different model than current practice, it can only have an impact if benefits are overwhelmingly compelling.

The discussion of this matter was interleaved with a related point, which I also posited rather provocatively. Are people stretching the definition of reflection too broadly?

Some of the examples given in the papers seemed to be just good object-oriented engineering, and had little to do with reflection as I saw it. For example, is delegation inherently a reflective mechanism? I would say no, but it is not clear if there is universal agreement. After all [7] is a paper on using reflection that presents a Java language extension closely related to delegation. If so, we can take this further and ask: are higher-order functions inherently reflective? If they are, so is any object oriented program.

As another simple example, the fact that a machine can report on its status is indeed reflective, but is nothing new. In particular, while the machine is reflective, the program that reports on, say, the status of the hardware is not reflective at all.

Many in the audience felt that my view of reflection was too narrow, and that reflection should be thought of as an a pattern, a way to structure a software system in order to achieve dynamic adaptability and separation of concerns. Good object-oriented engineering is a means to build a reflective system, but this does not make it less reflective.

One hopes that the controversy helped sharpen the arguments in favor of reflection, and might contribute to a widely agreed upon definition of reflection.

Session on Meta-level Architecture:

Summary by Pierre Cointe (Session Chair, *Ècole des Mines de Nantes*)

Four talks has been given in this session. Most of them (the first three) addressed the issue of introducing reflective facilities in Java in order to deal with mobility and/or security.

[12] A Reflective Framework for Reliable Mobile Agent Systems. *Takuo Watanabe, Amano Noriki and Kenji Shinbori* (JAIST, Ishikawa, Japan)

Takuo Watanabe gave the talk.

[10] Iguana/J: Towards a Dynamic and Efficient Reflective Architecture for Java. *Barry Redmond and Vinny Cahill* (Trinity College, Dublin, Ireland)

Barry Redmond gave the talk.

[11] Security and Meta Programming in Java. *Julien Vayssière* (INRIA-CNRS-I3S, Nice, France)

[1] Reflective Actors for Mobile Agents Programming. *Jean-Paul Arcangeli, Laetitia Bray, Annie Marcoux, Christine Maurel and Frédéric Migeon* (Université Paul Sabatier, Toulouse, France)

Frédéric Migeon gave the talk.

[12] described the implementation of a Java experimental framework to model mobile agents (and in the spirit of Lead++ providing dynamic adaptability). Reflection and AOP are used to separate non-functional features from the rest of the application. Mobility is considered as an aspect and every object has its own meta-object. One purely syntactical question was about the difference between a reflective framework (as used in the title of the paper) and a MOP.

[10] described the current work of implementing Iguana in Java. A first implementation of Iguana has been done in C++ and this current research explores issues in implementing Iguana for an interpreted language. The runtime intercession mechanism will be realized by providing a native code library which extends the virtual machine.

[11] discussed the match between Java security architecture and Java meta-programming facilities. First, Vayssière provided a very interesting survey about Java MOPs including OpenJava, Dalang/Kava, JavaAssist, MetaXa, Guarana and ProActive (but not Iguana!). This survey was based on the compile/load/run-time separations. Then Vayssière discussed how to protect Java security policies from reflection and, how to use Java MOPs to implement new security policies.

[1] discussed how to use actors (according to Hewitt|Agha definition) to program mobile agents and to provide a good paradigm to deal with mobility. Reflective architecture has also been introduced to separate functional aspects from operational ones.

Due to the number of talks and the to lack of time, there were only few questions. Most of them were about looking for a (Java) killer architecture dealing with security and mobility.

Session on Middleware: Session Chair Takuo Watanabe *JAIST, Japan*. Summary by Walter Cazzola and Shigeru Chiba

A lot of position papers has been submitted related to this topic, but only three significant papers has been selected for presentation:

[3] Reflective Implementation of non-functional Properties with the JavaPod Component Platform. *Eric Bruneton, and Michel Riveill* (INRIA, France)

Eric Bruneton gave the talk.

This work is focused on the separation and composition of functional and nonfunctional properties in a distributed framework. Issues left open from this talk are related to the role of reflection in their work, properties composition is achieved through a complex form of delegation.

[4] The Role of Meta-Information Management in Reflective Middleware. *Fábio Costa, and Gordon S. Blair* (Lancaster University, United Kingdom)

Fábio Costa gave the talk.

He presented the use of reflection for exposing various aspects of middleware configuration. Their architecture is based on a multi-model reflection framework originally developed by Okamura et al for the AL-1/D language. They identified four meta-models in their ORB system and designed reflective interface for those models. They also discussed the integration of their reflective features with the meta-object facility, which is part of the CORBA standard.

[6] Using Reflection for Composable Message Semantics. *Markus Hof* (Johannes Kepler University Linz, Austria)

Hof in his work has described how render remote invocations first class citizens, going a step further towards full polymorphism: the invocation of the same method can have different semantics on two objects of the same class.

During the discussion time, questions and comments from the floor were focused on whether reflection is the unique approach to the applications presented by the speakers. Some radical comments were that the technique used by JavaPod is not reflection but delegation and that the architecture used by Hof's system is traditional message filtering. These comments are not really true because their systems provide some *meta* information, which is not available at the base level. However, the speakers' responses did not seem strong enough to convince people, especially from the outside of the reflection community, of the usefulness of reflection in the middleware domain.

Session on Program Transformation:

Summary by Satoshi Matsuoka (Session Chair, *Tokyo Institute of Technology*)

There were three papers in the session, all relevant to user-induced program transformation, or in a more 'reflective' terminology "compile-time reflection".

[2] Declarative Meta-Programming for a Language Extensibility Mechanism. *Johan Brichau* (Vrije Universiteit Brussel, Belgium).

This paper basically proposes to utilize declarative programming techniques for program transformation|compile-time reflection. More specifically, prolog-like predicates are utilized to program the program transformation over a parse tree in a declarative manner. Although the idea of using declarative programming for program transformation is not new, its application to compile-time reflection, in particular to provide easy-to-use and possibly composable transformation, sheds some technical interest. Here are some of the technical points of interest for the paper|talk:

- Pro: Declarative specification of program transformers instead of procedural compile-time MOP is good.
- Question: Difference with traditional prolog|logic meta-programming? Other compile-time reflective systems that also facilitate parse-tree transformation? Are the abstractions too low level? What about conflicting transformations? How about some traditional analysis? Do you want a different metalanguage or the same, since when this system is applied to C++ or Java it is really not 'reflective' in a true sense, since one will be using different languages.

[8] Jasper: Type Safe Compile-Time Reflection Language Extensions and MOP Based Templates for Java. *Dmitry Nizhegorodov* (Oracle Corporation, USA)

Jasper is essentially an attempt to introduce the sophistication of modern Lisp-style macros into Java, in a reflective manner (so that macros can be programmed in Java as well.) It has had some history of real industrial application at Oracle, and as such it has an extensive layered architecture, which also distinguishes itself from previous Java compile-time reflective systems such as OpenJava. It also introduces type-safe template mechanism into Java, with the ability to control the expansion in a sophisticated manner. Here are some of the technical points of interest for the paper|talk:

– Pros: Layered architecture providing different levels of meta-programming abstractions, well organized. Good ideas from Lisp Macros
– Question: User experience—what layer would users like to use the most? Relationship with other Java reflective systems? What about the macro processing speed?

[5] On the Lightweight and Selective Introduction of Reflective Capabilities in Applications. *Rémi Douence, and Mario Südolt* (École des Mines de Nantes, France)

Rémi Douence gave the talk.

The final paper does not really focus on allowing user-defined program transformations, but rather on introducing reflective 'hooks' into existing Java code for computational reflection via program transformation. While this can be done rather straightforwardly and potentially useful, there are some potential technical problems as described below:

– Pro: Selective reification is lightweight.
– Question: Difference between previous systems, that have performed "selective reification" in the past, such as ABCL/2, and in particular Okamura & Ishikawa's AL-1 and AL-1/D? Is this not too straightforward, potentially introducing overhead? Will this destroy the basic class abstraction & encapsulation (e.g. field access) such that we lose separate compilation, etc.?

In the discussions period, one of the main issues was in line with what had been raised in Gil Bracha's session—what kind of real-life impact each system will have, in terms of their advantages over existing solutions, and|or previous reflective systems. In this respect Jasper holds advantages over the other two systems, since it is already applied to a real product and the others are still quite experimental. As such many questions focused on how Jasper is being used in a real setting, and it was presented that Jasper templates and macros are used heavily as is with traditional C++ templates or Lisp macros. Thus, the utility question was answered in a very positive fashion, but as to what style of compile-time meta-programming would be considered ideal or advantageous remained an open question.

2 Reflection Trends: the Organizers' Opinion

Summary by Shigeru Chiba *(University of Tsukuba)*

In this summary, I would like to describe some issues that I thought were interesting during the workshop.

Middleware

Reflection is an active research issue in the middleware field. This workshop received a number of paper submissions from this field.

During this workshop, I was wondering what reflection means in the middleware field. In the language field, reflection means the ability to query an object for its class, methods, fields, and so on (introspection), and to modify a program by changing methods and class definitions (program transformation). This ability, which allows us to access non 1st-class language constructs (Jacques Malenfant reminded us of this classic definition. Although many participants did not seem to like this definition, I think this is the clearest definition at least in the language field), has clear value even in industry.

One of the issues confusing me was differences between reflective middleware and modular middleware. Here, modularity means that middleware components can be replaced on demand and connected to other components for extension. Is reflection a technique for building modular middleware? I think the answer should be yes but I do not think that the middleware papers in this workshop presented a reflective technique for implementing modular middleware. Rather, they seemed to use "reflection" just as an alias of the word "modularity." Many techniques presented in those papers were categorized into traditional delegation and filter.

I thought that the classic definition in the language field could be also useful in the middleware field. If we find non 1st-class data that we want to access at runtime, we could develop something other than delegation and filter in the middleware field. Remember that even transparent method interception, which is a typical reflective technique, reifies non 1st-class data such as parameter types (marshaling). Fábio Costa's presentation [4] gave us another hint. He mentioned the use of reflection for managing type repository, which is a non 1st-class data structure. Version management based on interface types might be a good application of reflection in the middleware field.

According to Jacques Malenfant's comment, reflection is useful if an *unexpected* extension is needed in future. Is surveying the history of middleware helpful for reflection research?

Security

Julien Vayssière's presentation "Security and Meta Programming in Java" [11] was good survey. He rose two distinct issues: (1) how we protect systems from reflective computation, and (2) how reflection can be used for protecting systems. The former issue is relatively classic and well known but the latter issue is still open.

As for the former issue, he mentioned that existing reflective Java systems, such as OpenJava, Kava, and MetaXa, can potentially cause security problems. I agree to this argument since, in principle, reflection is a technique for accessing protected internal data structures. However, I do not think that those security problems are really *problems* in practice since the reflective capability is provided for only secure programs, such as a configuration program written by the users. Malicious programs cannot use the reflective capability. Reflective computation can cause a security problem only if malicious programs break the protection mechanism and use the reflective capability. We should rather discuss how to prevent malicious programs from using the reflective capability.

For protecting systems from security flaws caused by programmers' mistakes, load-time reflection is a reasonable trade-off point in Java. It is more adaptable than compile-time reflection since it is executed later. On the other hand, it is more secure than run-time reflection using a customized JVM since it can exploit a bytecode verifier. Even if the result of reflective computation is an invalid program, the bytecode verifier denies loading it into the virtual machine. However, some applications need runtime reflection. They need program transformation not at the bytecode level but at the assembly level and thus they need the reflective capability to customize the virtual machine at runtime. The best reflective architecture depends on applications.

The latter issue raised by Julien Vayssière was more interesting. Although he did not talk much about this issue, I think that using reflection for protection mechanisms can be significant applications of reflection. Security seems an orthogonal aspect of program against application logic. Reflection would be a good tool for describing a complex security policy.

Existing applications in the security field are not recognized as killer applications of reflection. For example, as Julien Vayssière pointed out, a metaobject intercepting method invocations is known as a good platform for implementing an access control list (ACL) but that is not a killer application. Why? Probably, the reason is that it does not solve any problem. Although it seems good from the viewpoint of software architecture, it does not provide a faster ACL implementation than traditional ones or make it easier for the users to write a security policy. It does not verify that the security policy includes no security hole. Furthermore, I'm not sure that it is a good architecture even from the implementation viewpoint.

Program Translation

Dmitry Nizhegorodov was a contributor from industry. His presentation "Jasper: Type-Safe MOP-Based Language Extensions and Reflective Template Processing in Java" [8] was about a reflective programming tool *Jasper*, which he is using for developing real products.

Reflective (i.e. programmable) program translators such as Jasper are obviously useful in practice. However, I was wondering that it is really feasible to use such a language-extension tool (without commercial supports) for software development by a large number of engineers. A program written in an extended language is difficult to read for team members who do not know the extensions. Like the excessive use of macros, the excessive use of language extensions is a negative factor of team collaboration.

In the discussion, I asked him this question. His answer was clear; he used Jasper for writing his code but he never shared the source code of Jasper with his team members. He said that he shared only the code produced by Jasper with his team members. Since the final stage of Jasper is a pretty printer, his team members can see only a formatted program written in the regular Java language.

Although Jasper is an extended Java compiler, his way of using Jasper reminds me of Emacs commands, which automate typical processes of text editing and assist us to write a program. The commands are used only during writing a program and the resulting program does not include any commands as the output of Jasper does not include any syntax extensions. The commands can be personalized; team members do not have to use the same set of Emacs commands. Syntax extensions by Jasper are similar to Emacs commands. Sharing complex syntax extensions among team members is not realistic if the team size is big, but personally using those extensions should be acceptable. I thought that Jasper showed one of the ways of promoting the use of reflective program translators in industry.

Summary by Walter Cazzola *(University of Milano Bicocca)*

In this summary, I would like to try weighing up what has been emerged from the workshop debates and focusing our attention on what I consider will be the future trends for research in object-oriented reflection.

Software Engineering

Reflection, thanks to its features, such as, *separation of concerns*, and *transparency*, is commonly recognized as a valuable instrument for developing, extending, and maintaining software.

Structuring software in mostly independent layers which will be successively combined together with little coding efforts, encourages developers to either develop different aspects of the application in different moments or entrust the development of a specific aspect to a different team. Such an approach improves:

- software reuse, — because to combine and to adapt existing modules with|to the new applications, it is simpler than using modularity,
- software stability, — the development of critical software's aspects may be entrusted to specialized teams or easily retrieved from well-known and tested library, and
- software maintenance, — each aspect of the system is realized by a close (i.e., without, or with very little, dependencies with other subsystems) subsystem, whose code is more compact than the code of the whole system.

Thanks to transparency it is also possible to reuse black box components.

Thus, reflection, if well applied, helps to overcome typical problems related to software component integration, which hinder the spreading of software reuse. A lot of work has already been done to define reflective languages and reflective tools for coding reflective systems. Unfortunately, very little has been done to define, or to extend, a methodology analogous to UML which helps in integrating reflective concepts since the software development early stages.

I think that reflection's future trends involve to shift up its concepts from the linguistic level to a methodological level, as it has already happened for object-oriented technology. To testify such a need there is the growing interest from people coming from software engineering field in reflection which has lead to some attempts (e.g., Suzuki and Yamamoto, Extending UML for Modeling Reflective Software Components, ≪`UML'99`≫) of modeling reflection in UML.

Functional and Nonfunctional Requirements

A typical open issue, related to consider reflection as a methodology and not simply as a development tool, is represented by the meaning of *functional* and *nonfunctional features*.

A basic criterion used to layerize an application consists in determining which features are not functional to the application requirements. Unfortunately a universal definition or a recognized taxonomy about what is or is not a functional feature doesn't exist. A formal definition of functional or nonfunctional feature would help in establishing what can be realized in the meta-level and would encourage to build libraries of meta-level entities up.

Some interesting ideas have been proposed by the Bruneton, and Riveill's work [3]. But they only consider how to implement nonfunctional features, without giving an algorithm to determine what is or not is (or they consider to be) functional to the application.

In my personal opinion the border between functional and nonfunctional features cannot be defined, because it depends on the application, i.e., what we consider nonfunctional for an application, can be considered functional to another. One idea to face this problem could be to outline a group of properties describing the concept of functional feature, and through such properties, to determine whether a feature is or is not functional, application by application.

At the workshop, few works considered these issues, and proposed some solutions. In his work, Emiliano Tramontana, who unfortunately shouldn't attend the workshop, has considered software's evolution and maintenance through reflective techniques.

Summary by Thomas Ledoux (École des Mines de Nantes)

Reflection: How and Why?

How? According to Gilad Bracha and Shigeru Chiba, a few papers submitted to the workshop had little to do with reflection. Rather than using purely reflective techniques, they used mechanisms such as delegation or message filtering. Then, they provided good modularity but they did not deal with 1st class constructs.

However, I want to be positive because some presentations gave a good idea of what is a reflective system (see [4,10,12]). For example, the Iguana/J architecture is a promising approach [10]. It deals with selective reification of behavior, run-time reflection and run-time efficiency.

Now, the Java language and/or the middleware domain are a common denominator between several works in the reflection area. We can bet that technical and architectural issues will progress in the next years.

Why? During the workshop, a recurrent question was "what is the killer application of reflection?". Gilad Bracha challenged everyone to argue for the pragmatic value of the most advanced uses of reflection that researchers have proposed in recent years. What are the real interests of reflection?

First, for SUN (cf. Java Reflection API) and lots of people, *introspection* has now clear value. Any interesting system has to query information on itself.

Then, flexibity and *adaptability* are other benefits provided by reflection. Recent work and workshop presentations (see [7,9]) showed these advantages in the middleware domain. Why should middleware be reflective? Because reflection makes system more adaptable to its environment and better able to cope with change. In a middleware context, the need to cope with change particularly arises when middleware based applications are deployed in dynamic environments such as multimedia, group communication, real-time and mobile computing environments (cf. the recent workshop on Reflective Middleware).

Finally, reflection allows *separation of concerns* which is highly desirable to build open systems (cf. summary by Walter Cazzola). Integrating definitions of functional and non-functional properties via reflective concepts at the design level is a new and an interesting issue. An example of need of separation of concerns is the industrial components model EJB. An EJB container deals with code weaving of functional and non-functional properties. Thus, the needs of separation of concerns are crucial and reflection, if well applied, helps to structure a software system.

In short, reflection has many good features. According to Gilad Bracha, reflective community has to prove the usefulness of reflection by demonstrating it as the best way to reduce time-to-market.

Composition of Concerns

One remaining main open issue is the composition of concerns. Distinguishing different abstraction levels, reflection helps separation of concerns. Some presentations emphasized the separation of functional and non-functional properties (see [3,6,12]). Such an approach improves reusability and provides a "plug and play" environment for enabling adaptability.

However, separation is a good idea if we can compose the separated concerns together!

In compile-time reflection, the composition of functional and non-functional properties is materialized by transformation rules and is based on a kind of join points (likewise in the AOP approach). In run-time reflection, this composition is materialized, in general, by trap calls. So, the composition between functional and non-functional properties owns a clear semantic.

However, the composition of several non-functional properties has to be studied more in deep. How to combine different non-functional properties in a clean way by keeping the original semantic of each property? The order of composition can have a serious impact on the result. For example, the composition of the concern "encoding" with the concern "tracing" supposes a special order. Interferences between non-functional properties lead to unexpected (meta)-behavior and discredit the advantages provided by the separation of concerns.

This problem was not mentioned at the workshop. At the design level, [6] suggests a composition model with the Strategy and Decorator design patterns. At the implementation level, [3] proposes a minimal composition model which should be clarified in the future. Researchers in the reflection area should be inspired by work on other fields such as mixin-based inheritance or AOP.

3 Final Remarks

This workshop main goal was to encourage people to present works in progress in the area of reflection and meta-level architectures. 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 conference Reflection'01 (`http://www.openjit.org/reflection2001`).

Acknowledgements. We wish to thank Gilad Bracha, Pierre Cointe, Satoshi Matsuoka, and Takuo Watanabe 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 researchers which partecipated to the workshop.

A Workshop Attendee

Lastname	Firstname	Company & Country	e-mail
Amano	Noriki	JAIST, Japan	n-amano@jaist.ac.jp
Arcier	Borice	ESSI/INRIA, France	arcier@essi.fr
Blay	Mireille	ESSI, France	blay@essi.fr
Bracha	Gilad	Sun Microsystems, USA	gilad.bracha@eng.sun.com
Brichau	Johan	Vrije Universiteit Brussel (VUB), Belgium	johan.brichau@vub.ac.be
Briot	Jean-Pierre	Laboratoire d'Informatique P6, France	jean-pierre.briot@lip6.fr
Bruneton	Eric	INRIA - Projet Sirac, France	eric.bruneton@inrialpes.fr
Cazzola	Walter	University of Milano Bicocca, Italy	cazzola@disi.unige.it
Charra	Olivier	INRIA - Projet Sirac, France	olivier.charra@inrialpes.fr
Chiba	Shigeru	University of Tsukuba, Japan	chiba@is.tsukuba.ac.jp
Chignoli	Robert	I3S - CNRS - UNSA, France	chignoli@i3s.unice.fr
Clark	Tony	University of Bradford, United Kingdom	a.n.clark@scm.brad.ac.uk
Cointe	Pierre	École des Mines de Nantes, France	cointe@emn.fr
Costa Moreira	Fábio	Lancaster University, United Kingdom	fmc@comp.lancs.ac.uk
Crescenzo	Pierre	I3S - CNRS - UNSA, France	pierre.crescenzo@unice.fr
Douence	Rémi	École des Mines de Nantes, France	douence@emn.fr
Ducournau	Roland	LIRMM-Université Montpellier II, France	ducour@lirmm.fr
Forax	Rémi	Université de Marne la Vallee, France	forax@univ-mlv.fr
Hof	Markus	University of Linz, Austria	hof@ssw.uni-linz.ac.at
Istenes	Zoltán	Eötvös University Budapest, Hungary	istenes@inf.elte.hu
Lacôte	Guillaume	Laboratoire d'Informatique P6, France	guillaume.lacote@poleia.lip6.fr
Ledoux	Thomas	École des Mines de Nantes, France	thomas.ledoux@emn.fr
Malanfant	Jacques	Université de Bretagne Sud, France	jacques.malenfant@univ-ubs.fr
Marangozova	Vania	INRIA - Projet Sirac, France	vania.marangozova@imag.fr
Maruyama	Fuyuhiko	Tokyo Institute of Technology, Japan	fuyuhiko.maruyama@is.titech.ac.jp
Matsuoka	Satoshi	Tokyo Institute of Technology, Japan	matsu@is.titech.ac.jp
Michiels	Isabel	Vrije Universiteit Brussel (VUB), Belgium	isabel.michiels@vub.ac.be
Migeon	Frédéric	IRIT - UPS, France	frederic.migeon@irit.fr
Moisan	Sabine	INRIA Sophia-Antipolis, France	sabine.moisan@sophia.inria.fr
Nacsa-Szabó	Rozália	Eötvös University Budapest, Hungary	nacsa@inf.elte.hu
Nizhegorodov	Dmitry	Oracle Corp, USA	dnizhego@us.oracle.com
Ogawa	Hirotaka	Tokyo Institute of Technology, Japan	ogawa@is.titech.ac.jp
Parlavantzas	Nikos	Lancaster University, United Kingdom	n.parlavantzas@lancaster.ac.uk
Peschanski	Frédéric	Laboratoire d'Informatique P6, France	frederic.peschanski@poleia.lip6.fr
Pic	Marc	CEA, France	marc.pic@cea.fr
Redmond	Barry	Trinity College Dublin, Ireland	barry.redmond@cs.tcd.ie
Rozenfarb	Dan	University of Buenos Aires, Argentina	drozenfa@dc.uba.ar
Sohda	Yukihiko	Tokyo Institute of Technology, Japan	sohda@is.titech.ac.jp
Stoops	Luk	Vrije Universiteit Brussel (VUB), Belgium	lstoops@vub.ac.be
Tajes Martinez	Lourdes	University of Oviedo, Spain	tajes@correo.uniovi.es
Tourwé	Tom	Vrije Universiteit Brussel (VUB), Belgium	tom.tourwe@vub.ac.be
Vayssière	Julien	INRIA Sophia-Antipolis, France	julien.vayssiere@sophia.inria.fr
Watanabe	Takuo	JAIST, Japan	takuo@acm.org
Welch	Ian	University of Newcastle upon Tyne, UK	i.s.welch@ncl.ac.uk
Wileden	Jack	University of Massachusetts, USA	jack@cs.umass.edu

References

1. Jean-Paul Arcangeli, Laetitia Bray, Annie Marcoux, Christine Maurel, and Frédéric Migeon. Reflective Actors for Mobile Agents Programming. In Walter Cazzola, Shigeru Chiba, and Thomas Ledoux, editors, *On-Line Proceedings of ECOOP'2000 Workshop on Reflection and Metalevel Architectures*, June 2000. Available at `http://www.disi.unige.it/RMA2000`.
2. Johan Brichau. Declarative Meta-Programming for a Language Extensibility Mechanism. In Walter Cazzola, Shigeru Chiba, and Thomas Ledoux, editors, *On-Line Proceedings of ECOOP'2000 Workshop on Reflection and Metalevel Architectures*, June 2000. Available at `http://www.disi.unige.it/RMA2000`.
3. Eric Bruneton and Michel Riveill. Reflective Implementation of non-functional Properties with the JavaPod Component Platform. In Walter Cazzola, Shigeru Chiba, and Thomas Ledoux, editors, *On-Line Proceedings of ECOOP'2000 Workshop on Reflection and Metalevel Architectures*, June 2000. Available at `http://www.disi.unige.it/RMA2000`.
4. Fábio Costa and Gordon S. Blair. The Role of Meta-Information Management in Reflective Middleware. In Walter Cazzola, Shigeru Chiba, and Thomas Ledoux, editors, *On-Line Proceedings of ECOOP'2000 Workshop on Reflection and Metalevel Architectures*, June 2000. Available at `http://www.disi.unige.it/RMA2000`.
5. Rémi Douence and Mario Südolt. On the Lightweight and Selective Introduction of Reflective Capabilities in Applications. In Walter Cazzola, Shigeru Chiba, and Thomas Ledoux, editors, *On-Line Proceedings of ECOOP'2000 Workshop on Reflection and Metalevel Architectures*, June 2000. Available at `http://www.disi.unige.it/RMA2000`.
6. Markus A. Hof. Using Reflection for Composable Message Semantics. In Walter Cazzola, Shigeru Chiba, and Thomas Ledoux, editors, *On-Line Proceedings of ECOOP'2000 Workshop on Reflection and Metalevel Architectures*, June 2000. Available at `http://www.disi.unige.it/RMA2000`.
7. Vania Marangozova and Fabienne Boyer. Using Reflective Features to Support Mobile Users. In Walter Cazzola, Shigeru Chiba, and Thomas Ledoux, editors, *On-Line Proceedings of ECOOP'2000 Workshop on Reflection and Metalevel Architectures*, June 2000. Available at `http://www.disi.unige.it/RMA2000`.
8. Dmitry Nizhegorodov. Jasper: Type Safe Compile-Time Reflection Language Extensions and MOP Based Templates for Java. In Walter Cazzola, Shigeru Chiba, and Thomas Ledoux, editors, *On-Line Proceedings of ECOOP'2000 Workshop on Reflection and Metalevel Architectures*, June 2000. Available at `http://www.disi.unige.it/RMA2000`.
9. Nikos Parlavantzas, Geoff Coulson, Mike Clarke, and Gordon Blair. Towards a Reflective Component Based Middleware Architecture. In Walter Cazzola, Shigeru Chiba, and Thomas Ledoux, editors, *On-Line Proceedings of ECOOP'2000 Workshop on Reflection and Metalevel Architectures*, June 2000. Available at `http://www.disi.unige.it/RMA2000`.
10. Barry Redmond and Vinny Cahill. Iguana/J: Towards a Dynamic and Efficient Reflective Architecture for Java. In Walter Cazzola, Shigeru Chiba, and Thomas Ledoux, editors, *On-Line Proceedings of ECOOP'2000 Workshop on Reflection and Metalevel Architectures*, June 2000. Available at `http://www.disi.unige.it/RMA2000`.
11. Julien Vayssière. Security and Meta Programming in Java. In Walter Cazzola, Shigeru Chiba, and Thomas Ledoux, editors, *On-Line Proceedings of ECOOP'2000 Workshop on Reflection and Metalevel Architectures*, June 2000. Available at `http://www.disi.unige.it/RMA2000`.
12. Takuo Watanabe, Amano Noriki, and Kenji Shinbori. A Reflective Framework for Reliable Mobile Agent Systems. In Walter Cazzola, Shigeru Chiba, and Thomas Ledoux, editors, *On-Line Proceedings of ECOOP'2000 Workshop on Reflection and Metalevel Architectures*, June 2000. Available at `http://www.disi.unige.it/RMA2000`.