ECOOP 2003 Workshop Report: Seventh Workshop on Tools and Environments for Learning Object-Oriented Concepts

Isabel Michiels[1], Jürgen Börstler[2], Kim B. Bruce[3], and Alejandro Fernández[4]

[1] PROG, Vrije Universiteit Brussel, Belgium
[2] Umeå University, Sweden
[3] Williams College, Massachusetts, USA
[4] Fraunhofer IPSI, Darmstadt, Germany

Abstract. This report summarizes the results of the seventh workshop in a series of workshops on learning object-oriented concepts. The focus of this workshop was on (computer-aided) support for the teaching and learning of basic object-oriented concepts.

1 Introduction

Teaching the object-oriented paradigm in the same way as "traditional" introductory programming courses does not appear to work very well. In the pre-OO world, concepts could be introduced step by step and the grouping of program elements could be handled as an afterthought. There was no need to introduce high-level and abstract structures, like modules or abstract data types, early on. This is very different in the object-oriented paradigm, where the basic concepts are tightly interrelated and seem not to be easily explained in isolation. Instead they need to be handled in groups (like, for example, variable, value, type, object, and class) making teaching and learning more challenging. Grasping the "big picture" may furthermore be hindered by focusing on notational details of specific object-oriented programming languages. Most students therefore have difficulties taking advantage of object-oriented concepts.

This was the seventh in a series of workshops on issues in object-oriented teaching and learning. Reports from all previous workshops in the series are available [1–5, 9]. Further information can be found at the workshop series home page [17].

The objective of the workshop series was to share experiences and discuss ideas, approaches and hypotheses on how to improve the teaching and learning of object-oriented concepts.

Each workshop in the series focussed on a specific theme. This workshop's theme was (computer-aided) support for the teaching and learning of basic object-oriented concepts. The organisers particularly invited submissions on the following topics:

- intelligent learning environments (for teaching object technology)

– frameworks/toolkits/libraries for learning support
– approaches and tools for teaching design early
– different pedagogies
– design early vs. design late
– frameworks/toolkits for the development of teaching/learning applications
– experiences with innovative CS1 curricula
– usage of metaphors, analogies, and illustrative examples
– distance education

2 Workshop Organization

Participation at the workshop was by invitation only. The number of participants was limited to encourage the building of few small interest groups working on specific topics. Potential attendees were required to submit position papers.

Time	Topic
9.00 am - 9.10 am	WELCOME AND INTRODUCTION
9.10 am - 10.30 am	*First Presentation Session*
9.10 am	Animated UML as a 3D-illustration for Teaching OOP by Uwe Thaden
9.35 am	A Framework for Lightweight Object-Oriented Design Tools by Jörg Pleumann
10.00 am	Java Online Pedagogy by Jürgen Wolff von Gudenberg
10.30 am - 11.00 am	COFFEE BREAK
11.00 am - 12.30 am	*Second Presentation Session*
11.00 am	Socio-Cultural Perspectives on Object-Oriented Learning by Ola Berge
11.30 am	Teaching About Creational Design Patterns - General Implementations of an Algebraic Structure by Virginia Niculescu
12.00 am	OO Learning, a Modeling Approach by Arne-Kristian Groven
12.30 am - 1.30 pm	LUNCH BREAK
1.30 pm	Position Statement Presentations
2.00 pm	Discussion Topic Selection
2.20 pm	First Group Discussion Session
3.00 pm - 3.30 pm	COFFEE BREAK
3.30 pm	Second Group Discussion Session
4.45 pm	Wrap-up session

Table 1. Workshop program

Out of the 17 position papers that were submitted, six papers were selected for presentation at the workshop. The authors of nine other papers were invited for participation at the workshop. All contributions were made available on the workshop's website a few weeks before the workshop, in order to give attendees the possibility to prepare the discussions. Presentations were scheduled for the

morning sessions. The afternoon sessions were dedicated to discussions in small working groups.

All attendees were given the opportunity to present their positions briefly (one overhead, at most two minutes time). After that three working groups were formed to discuss in more detail the topics they found most relevant. The full workshop program can be found in Table 1.

The workshop gathered 28 participants from 11 different countries, all of them from academia. A complete list of participants together with their affiliations and e-mail addresses can be found in table 2.

3 Summary of Presentations

This section summarizes the main points of the presented papers and the most important positions of the other participants. More information on the presented papers can be obtained from the workshop's home page [10].

Uwe Thaden (Learning Lab, Hannover, Germany) discussed the usage of 3D animations of UML diagrams to visualize the execution of object-oriented code.

A basic premise for good software development is the ability of programmers to think in the concepts of the programming language used. Thus, today object-oriented programming needs to be taught in an adequate way. The procedural or imperative style of programming corresponds very well with the verbal descriptions of algorithms, concentrating on control flow. Important and typical for the object-oriented paradigm are the dynamic creation and destruction of objects, the links between them, and their interaction through exchanging messages. Program execution is distributed over several objects.

Object-oriented programming was conceived as a way of modeling reality. It seems to be worth examining whether this naturalness can be exploited to explain program execution other than by mapping it to a sequence of instructions to a register machine. The Unified Modeling Language (UML) offers several dynamic diagram types (e.g. sequence- and collaboration diagram) of particular interest for such an approach.

However, paper-bound visualizations of program executions are inevitably static projections that cannot transport program dynamics. The goal of the presented approach is not only to visualize algorithm execution, but to develop a tool to aid the intuitive understanding of an arbitrary Java program execution. The presented (Java) prototype uses an XML structure representing UML diagrams. This structure can be obtained from existing Java programs by means of UML modeling tools ("Together" in this case). The XML structure is then transformed into VRML that can be displayed in web browsers.

Figure 1 shows a screenshot taken from an instantiation animation. The classes in this example are shown at a higher level while the instances are placed on the "ground" to visualize the conceptual difference between class and object in an intuitive way.

Jörg Pleumann (University of Dortmund, Germany) proposed a framework for lightweight object-oriented design tools. Modeling on the basis of formal vi-

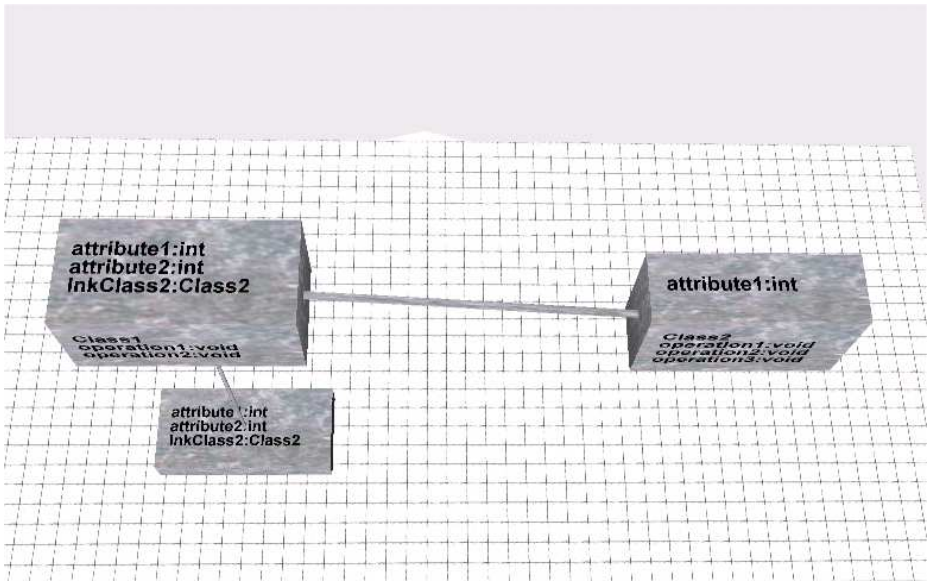

Fig. 1. Example view of an instantiation

sual languages like the UML has become a central activity in today's software development processes. Universities need to teach new software developers how to express their ideas in the form of models and how to understand the models created by others. When the size of models approaches that of non-trivial problems, tool support becomes crucial – even during education. Unfortunately, the typically-used industrial CASE tools have major drawbacks when applied in an educational setting. Aimed at professional work, they are too complex for classroom-use and lack features that support their users in learning a modeling language.

The Java-based framework for so-called "lightweight" modeling tools developed at the University of Dortmund is meant to be a first step towards didactical modeling tools. All tools derived from the framework share the same simple and intuitive user interface and have only small resource demands. The two main extension points of the framework are a generic metamodel and generic graphical figures. New tools are easily developed by "plugging in" the concrete metamodel and graphical elements of a specific modeling language. Three modeling tools have already been built using the framework; a statechart editor and simulator, a software architecture editor and a toolset for the Unified Process. The statechart editor features a multimedia simulation engine interfacing with the user's model. This interface can be used to control a washing machines or a coffee

machine. The tool has already been used successfully in a software engineering class at Dortmund.

Jürgen Wolff von Gudenberg (University of Würzburg, Germany) claimed that programming can only be learned by reading and, first of all, by writing programs. Hence, an online tutorial should provide a high level of interaction. The evaluation of and immediate response to performed tests or submitted programs enhances the value considerably.

The presented JOP (Java Online Praktical) was developed by the universities of Passau and Würzburg (both Germany). It comprises two parts: a Java tutorial and evaluator to check exercises and give feedback to students.

The tutorial is organized as a collection of learning units together with immediately executable and modifiable examples and exercises. It can be noted that interfaces are discussed long before inheritance and polymorphism. I/O and GUIs are the last topic. Examples and exercises are evaluated by electronic tutors. These tutors perform functional as well as structural tests and provide immediate feedback to the users. Examples of tests performed are conformance to coding conventions, proper documentation, usage of required/forbidden classes and the correct execution of supplied functional tests.

The system has been used in different lectures. In those lectures about 2500 programs for 16 different problems have been evaluated. The average size of a program was about 1000 lines of Java code. The work load for examining and marking the programs could be decreased by a factor of four, while the quality and readability of the programs increased significantly. The tests are furthermore quite reliable. Only about two percent of the programs that passed all automatic tests were not accepted by a human inspector.

Ola Berge (Intermedia, University of Oslo, Norway) talked about socio-cultural perspectives on object-oriented learning and how these aspects relate to the COOL (Comprehensive Object-Oriented Learning) project [13]. One of the central objectives of this project is to explore critical aspects associated with learning (and teaching) object-oriented concepts. This objective will be pursued by bringing forth the heritage of Kristen Nygaard and the Norwegian approach to object-orientation. The project aims at developing learning materials based on this work and intends to make those materials available to the object-oriented community in the form of reusable learning objects.

Berge and colleagues apply a socio-cultural perspective on the area of learning object-oriented concepts. It raises a number of issues concerning design of learning objects and learning environments for knowledge construction in this field. The COOL project explores these issues by studying current practices as well as by experimenting with new constellations of artifacts. Early activities include the development of courses that introduces learners and practitioners to fundamental object-oriented concepts through approaches such as “models first” and “objects first.” The courses will be developed evolutionary, where experience from the first courses, planned to be held in Oslo in the summer of 2003, will provide insights for further improvement of the subsequent activities.

The socio-cultural perspectives give directions of how tools (e.g. KarelJ [15] or BlueJ [12]) and other ICTs (Information and Communication Technologies) should be incorporated in the learning activity. Therefore, questions arise on how existing and new learning material should be implemented as learning objects in ICT-based learning environments. Last, but not least, these perspectives provide also guidelines on how to understand the meaning of social interaction with respect to learning.

Virginia Niculescu (Babes-Bolyai University, Cluj-Napoca, Romania) presented an approach to teach creational design patterns based on simple algebraic structures.

Teaching design patterns to students with little experience in OOP is difficult. It is therefore crucial to use examples from well-known domains. Since most CS programs start with (among others) Algebra, this seems a suitable area.

To implement a general algebraic structure over a field of special values, such as null and unity elements, are needed. To build a structure that is independent from the specific field chosen, these special values must be created by specific methods. The goal is to develop a general polynomial class that is independent from the type of its coefficients, i.e. uses an abstract coefficient type (`FieldElement`). When working with polynomials over specific coefficient types it is necessary to handle special values independent of their exact type. For instance, when one constructor of the class polynomial creates a null polynomial, one has to create a null coefficient, even if its exact type is not known.

Niculescu presented three approaches to solve this problem, based on three classic creational design patterns [6]: Factory Method, Abstract Factory, and Prototype. Students can then compare solutions to specific problems using all three approaches. This enables students to understand these design patterns, and the differences between them. Experience confirms that applying patterns in a known domain helps students to better understand the concepts. In the future this work will be extended to cover further algebraic structures, such as matrices, and to build a general algebraic library. This might even help to reinforce some of the mathematical concepts.

Arne-Kristian Groven (Norwegian Computing Centre, Oslo, Norway) argued that OO programming should be regarded as modeling, establishing the basic concepts of the OO perspective early in the study. The dominating current pedagogical approach is often justified by the statement that "teaching must start with sufficiently simple examples." This statement is often (mis-) understood as to use traditional examples developed with the imperative programming paradigm in mind.

The Scandinavian tradition builds upon "sufficiently complex examples" as propagated by Kristen Nygaard. Modeling is about creating a description of phenomena and concepts from a given application domain. In OO modeling these phenomena and concepts are described as classes and objects. A model is itself an abstraction of something for the purpose of understanding it. The expressiveness of programming languages is limited. They are therefore not suitable to describe all aspects of an application domain. It is therefore imperative to

expose students to different programming languages to enrich their vocabulary for modeling different aspects of the application domain.

Groven and colleagues have started to monitor and evaluate OO courses to identify recurring problems and approaches that work or do not work respectively. As a next step they have planned to carefully design interventions in existing courses to study the effects of modeling based approaches. The first results are expected for the beginning of 2004.

4 Working Group Discussions

For the afternoon sessions participants formed three working groups to discuss the following topics. The following subsections summarize the results from these workings groups.

1. Tools
2. OO languages for teaching - why all this Java?
3. What are the real hard problems?

4.1 Tools

This group discussed the usage of tools in various teaching situations. Four main areas for tool support were identified; modeling and design, coding, execution, and evaluation. The strengths and weaknesses of several popular tools were shortly discussed (Alice [11], BlueJ [12], Fujaba [14], JKarel [15], and Praktomat [16]). It was noted that all of them worked excellently in at least one of the four areas above, but none of them did support all four areas.

Working group participants would require the following minimum properties/capabilities from a good OO educational tool:

- clear purpose
- simple & lightweight, but not simplistic
- impose restrictions on programming styles
- visualization of collaborations between objects
- easy transition to "real life" programming tools or environments

Finally, the group expressed also what they wish to see more of in available or future tools:

- traceability; seeing the effects of change
- better integration of tools with each other
- student progression monitoring
- (semi-) automatic evaluation of models, code, and documentation

4.2 OO languages for teaching - why all this Java?

This group debated on the use of OO languages for teaching purposes. The following three questions came to the surface at the beginning of the discussion:

1. Do we need a subset of an existing language?
2. Do we need a brand new language?
3. Does the perfect language perhaps already exist?

Group members agreed on the value of the following language features to support learning of OO:

Static Types and Type Declarations The discussion emphasized that the following properties of static type systems were often valuable in teaching novices:

- safety
- documentation
- tests (powerful documentation!)
- programming environment
- type inference
- beta patterns

Closures There was some discussion as to whether it was useful to have a language that provided support for closures. At least one participant, Bruce, argued that they were not necessary, and that some of the same expressiveness could be provided in a language like Java with inner and anonymous classes. Even then these features were not necessary in a first semester course, though inner classes can be quite helpful in a course on data structures.

Don't fall off a cliff The participants agreed that it could be helpful to use programming environments that provide support for focusing on only parts of the programming language (the part that students are learning at some time). Students shouldn't accidentally use part of a language or an environment which they should not know (yet), in order to avoid confusion. In DrScheme for example, teaching packs are offered depending on the level of the student.

4.3 What are the real hard problems?

The following aphorisms, ideas, and questions came up during the discussion:

- Is it hard to teach or to learn?
- We need to bridge the gap between stories and the program
- Students are not empty bowls
- Students can learn by seeing other students' errors
- As with writing poetry, students should learn to read before learning to write
- A good programmer is lazy – they copy the best practice
- Good design comes from experience – experience comes from bad design

This group started the discussion with a comment of Joe Bergin. He said that if you teach right, learning is easy. But since we still do need to figure out how to teach right, we tried to identify the current problems we have.

Abstraction was considered to be one of the hardest problems – to let students grasp the object-oriented way of thinking and to learn how to generalize a problem. Teaching more than one paradigm was found to be important for broadening a student's mind. Using functional programming was mentioned in this context, because for example Scheme, a dynamically typed, functional language, is an excellent tool for learning about abstraction. Consider, for example, defining higher-order functions in Scheme. This way, you learn to factor out the variabilities of common subproblems.

Polymorphism is a tough topic as well, although others find it a quite natural topic if you succeed in relating polymorphism with things that are already known in the real world. An example of this could be asking a question to two different people, as they might respond in completely different ways.

Another hot topic in this group was a more general remark: other students can benefit by seeing other students' errors. Student assignments based on this idea were discussed, like grading students on their ability to criticize the other's designs, and more than that, also grade them on their ability to improve their work after hearing the criticism.

Another range of hard problems are the ones related to design and modeling: how do you visualize things for students, and how do you provide them with the sort of mental model they need to have? It is clear that even for experienced practitioners, everyone uses their own way of mentally drawing their idea of the design, and even if people use the same notation, people sometimes assign different semantics to it.

5 Conclusions

The objective of this workshop was to discuss current tools and environments for learning object-oriented concepts and to share ideas and experiences about the usage of computer-based tools to teach the basic concepts of object technology.

Ongoing work was presented by a very diverse group of people with very different backgrounds, which resulted in the discussion of a broad range of topics like tool support, environments, courses, teaching approaches, languages for teaching, etc.

Summarizing what was said in the debate groups, we can conclude that:

- Teaching/learning tools should be lightweight and reflect a clear purpose. Support should be provided for visualizing the different aspects of the OO paradigm like object interactions. Integration of different educational tools should be increased, and the transition to *real-life* programming should be easily possible.
- There are many advantages to using an OO language for teaching that supports static types and type declarations. It is also very helpful if support is

provided for focussing only on some aspects of the language (concepts that don't have to be known yet shouldn't be made available) - like teaching packs in DrScheme.
- Having talked about what the hard problems in teaching are, we concluded that abstraction(learning to generalize), polymorphism, teaching design, and providing mental models provide the biggest problems.
- Getting to know different programming paradigms is important; other paradigms, like functional programming, can aid significantlyin clarifying different aspects of programming.

References

1. Börstler, J. (ed.): OOPSLA'97 Workshop Report: Doing Your First OO Project. Technical Report UMINF-97.26, Department of Computing Science, Umeå University, Sweden (1997).
2. Börstler, J. (chpt. ed.): Learning and Teaching Objects Successfully. In: Demeyer, S., Bosch, J. (eds.): Object-Oriented Technology, ECOOP'98 Workshop Reader. Lecture Notes in Computer Science, Vol. 1543. Springer-Verlag, Berlin Heidelberg New York (1998) 333-362.
3. Börstler, J., Fernández, A. (eds.): OOPSLA'99 Workshop Report: Quest for Effective Classroom Examples. Technical Report UMINF-00.03, Department of Computing Science, Umeå University, Sweden (2000).
4. I. Michiels, J. Börstler: Tools and Environments for Understanding Object-Oriented Concepts, ECOOP 2000 Workshop Reader, Lecture Notes in Computer Science, LNCS 1964, Springer (2000), pages 65-77.
5. I. Michiels, J. Börstler and K. Bruce: Sixth Workshop on Pedagogies and Tools for Learning Object-Oriented Concepts, in J. Hernández and A. Moreira, editors, Object Oriented Technology - ECOOP 2002 Workshop Reader, Volume 2548 of Lecture Notes in Computer Science, LNCS 2548, Springer (2002), pages 30-43.
6. E. Gamma , R. Helm , R. Johnson , and J. Vlissides . Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.
7. Pedagogical Patterns pages. `http://www-lifia.info.unlp.edu.ar/ppp/` `http://csis.pace.edu/~bergin/PedPat1.3.html`
8. European Master in Object-Oriented Software Engineering. `http://www.emn.fr/MSc/`
9. OOPSLA01 workshop. `http://www.cs.umu.se/%7Ejubo/Meetings/OOPSLA01/`
10. ECOOP 2003 Workshop homepage. `http://prog.vub.ac.be/~imichiel/ecoop2003/workshop/`
11. Alice homepage. `http://www.alice.org`
12. BlueJ - the interactive Java environment. `http://www.bluej.org`
13. COOL project Workshop homepage. `http://www.intermedia.uio.no/cool/`
14. Fujaba (From UML to Java And Back Again) homepage. `http://www.uni-paderborn.de/cs/fujaba/`
15. JKarel Robot homepages. `http://csis.pace.edu/~bergin/#kjr` and `http://math.otterbein.edu/JKarelRobot/`
16. Praktomat. `http://sourceforge.net/projects/praktomat/`
17. Workshops on OO Education. `http://www.cs.umu.se/research/education/ooEduWS.html`

Name	Affiliation	E-mail Address
Isabel Michiels	*Vrije Universiteit Brussel, Belgium*	imichiel@vub.ac.be
Jürgen Börstler	*Umeå University, Sweden*	jubo@cs.umu.se
Kim Bruce	*Williams College, USA*	kim@cs.williams.edu
Alejandro Fernandez	*Fraunhofer IPSI, Darmstadt, Germany*	casco@ipsi.fhg.de
Ola Berge	*Intermedia, University of Oslo, Norway*	ola.berge@intermedia.uio.no
Joe Bergin	*Pace University, USA*	jbergin@pace.edu
Andrew P. Black	*OGI School of Science and Engineering, Oregon, USA*	black@cse.ogi.edu
Richard Edvin Borge	*University of Oslo, Norway*	richared@ifi.uio.no
Thomas Cleenewerck	*Vrije Universiteit Brussel, Belgium*	tcleenew@vub.ac.be
Pedro J. Clemente	*University of Extremadura, Cáceres, Spain*	jclemente@unex.es
Jessie Dedecker	*Vrije Universiteit Brussel, Belgium*	jededeck@vub.ac.be
Wolfgang De Meuter	*Vrije Universiteit Brussel, Belgium*	wdmeuter@vub.ac.be
Arne-Kristian Groven	*Norwegian Computing Centre, Oslo, Norway*	Arne-Kristian.Groven@nr.no
Håvard Hegna	*Norwegian Computing Centre, Oslo, Norway*	havard.hegna@nr.no
Mario Kusek	*University of Zagreb, Croatia*	mario.kusek@fer.hr
Dennis Medzihradszky	*Dennis Gabor College, Budapest, Hungary*	medzihradszky@ szamalk.hu
Birger Møller-Pedersen	*University of Oslo, Norway*	birger@ifi.uio.no
Marie-Helene Ng Cheong Vee	*Birkbeck College, University of London, UK*	gngch01@dcs.bbk.ac.uk
Virginia Niculescu	*Babes-Bolyai University, Romania*	vniculescu@nessie.cs.-ubbcluj.ro
Jörg Pleumann	*Universität Dortmund, Germany*	pleumann@ls10.cs.uni-dortmund.de
Wilfried Rupflin	*University of Dortmund, Germany*	wilfried.rupflin@uni-dortmund.de
Jens Schröder	*Universität Dortmund, Germany*	schroeder@ls10.cs.uni-dortmund.de
Tyszberowicz Shmuel	*Tel-Aviv University, Israel*	tyshbe@post.tau.ac.il
Marianna Sipos	*Dennis Gabor College, Budapest, Hungary*	sipos@szamalk.edu
Friedrich Steimann	*Learning Lab, Hannover, Germany*	steimann@acm.org
Uwe Thaden	*Learning Lab, Hannover, Germany*	thaden@learninglab.de
Jürgen Wolff von Gudenberg	*University of Würzburg, Germany*	wolff@informatik.uni-wuerzburg.de
Amiram Yehudai	*Tel-Aviv University, Israel*	amiramy@tau.ac.il

Table 2. Workshop participants