Training in New Programming Technologies: an Experience

Invited Paper Presented at the
European Conference on Education and Computer Science in Europa'92,
Elea Press, pages 117-126, Salerno, Italy, November 1991

Jean-Pierre BRIOT

LITP
Institut Blaise Pascal
4 place Jussieu, 75252 Paris Cedex 05, France
tel: +33 (1) 44 27 40 55
fax: +33 (1) 44 27 62 86
e-mail: `briot@litp.ibp.fr`

Abstract

This paper discusses the introduction of new programming methodologies to computer science and practice. New metaphorical programming models are currently emerging, transferred from various scientific disciplines. The common ground to these new models is the concept of interacting active entities. These entities are called objects and they execute and interact concurrently. This new leading programming methodology is consequently called object-oriented concurrent programming (OOCP at short). We introduce quickly the concepts of OOCP methodology. We then focus on the issue of teaching it. We describe a platform which we designed in order to help describing and experimenting with OOCP. We discuss the experience of using this platform for pedagogical purposes.

1 New Trends in Programming

The use of computers is being ubiquitous in our societies. Early use of computers was restricted to scientific computation and data processing. Application fields for computers is now widely open and application tasks are of higher level. This evolution boosts development of general and high-level programming models. In the following of the section we focus on one of the leading new programming methodology for future computers and applications.

1.1 Metaphorical Programming

Programming computers is currently ongoing very deep evolution. Programming models are moving from mathematical and instruction-based models to *metaphorical* models. By metaphorical programming models, we mean new programming models which are results of transferring scientific theories of real world to provide alternative computation models. A famous example is the case of (artificial) neural networks [PDP 86], abstraction of neurophysiology. Other examples include abstractions from chemistry (reaction/diffusion-based computation models, the chemical abstract machine [Berry and Boudol 90]), genetics (genetic algorithms [Goldberg 89]), ethology (ant nests), economy (market-based models), and other various kinds of self-organizing systems [ECO 88]. One of the main trend in new programming models is the notion of communicating and cooperative agents, called Distributed Artificial Intelligence (DAI at short), taking much input from sociology and organization theories [DAI 88].

1.2 Objects

For most of these alternative models, the basic underlying computation model is based on the notion of active and cooperative entities. This is called *object-oriented programming* (OOP at short). The model is conceptually simple and general enough to be applied widely. It is based on two main concepts: *objects* which identify knowledge (that is data and services), and a unified communication/activation protocol between them, called *message passing*. Other useful concepts are *abstraction* and *inheritance*, which allow to classify and reuse object descriptions.

The first occurrence of programming languages and systems based on these concepts came out in the late 60's (Simula, and then Smalltalk [Goldberg and Robson 83]). Apple's Macintosh, designed and implemented along these concepts, was a key step to popularize this new vision of the computer. This methodology has now been validated by software engineering as a significant improvement in program development, quality, maintenance, and reusability [Meyer 88]. We note that because of the sequential nature of processors, concurrency of activities among objects, although implicit in the metaphor, is usually not addressed. As a result, most of OOP languages remain sequential.

1.3 Concurrency

However *concurrency* is becoming a key issue for future computer systems. It is boosted by the availability of new multi-computers which offer parallel execution of programs. These are promises for increasing efficiency. Other motivations for concurrency are the distribution of knowledge, as illustrated by computer networks, and the multiple interaction aspects of large interactive programs (such as for process control, where input and control may be multiple and distributed).

Concurrent programming abstracts from specific execution rates on specific architectures. Therefore concurrency (logical simultaneity of activities) should be distinguished from parallelism (physical simultaneity). Synchronization is the way to ensure coordination and non-interference of activities by specifying constraints on the ordering of events [Andrews and Schneider 83].

1.4 Object + Concurrency = Actor

Object-oriented concurrent programming (OOCP at short) is the combination of object-oriented programming with concurrent programming [OOCP 87]. In order to achieve optimal integration of OOP and concurrency, it is fruitful to identify objects as the unit of activity and to associate synchronization between objects at message passing level. Consequently each object owns its activity and is built-up to handle multiple interactions.

This leads to concurrency of activities and interactions between objects. These concurrent and active objects are also oftenly called *actors*.

Conventional (sequential) object-oriented programming may be viewed as a technological restriction of the general metaphor of object-oriented (concurrent) programming. OOCP is a very good methodology for decomposing large programs into a collection of cooperative modules, and to execute them onto parallel computer architectures. Advantages may be summarized as following: high-levelness, modularity, dynamicity, and openness. OOCP is a new growing field, but has already main impact on new multi-processor architectures (like the J-Machine [Dally et al. 89]) and various applications like signal processing [Barry 89], process control, office information systems, animation...

2 Teaching

In this section we will discuss the issue of introducing this new programming methodology to students and conventional programmers. This discussion is based on our personal experience in teaching object-oriented (concurrent) programming in courses and seminars at Universities, Engineering Schools, and Companies. We also developed a system to help describing object-oriented concurrent programming, classifying various programming languages, and experimenting with associated methodology.

2.1 Conceptual Programming

Object-oriented programming is more conceptual than conventional programming. It focuses on interaction between agents, abstracting from their representation and implementation details. Consequently we believe that this makes it easier for programmers and non-programmers to catch up with this methodology. This opinion is supported by our experience of introducing the concepts of object-oriented programming to various audiences.

2.2 From Objects to Actors

Object-oriented programming is getting now heavily disseminated in the programming community. A hot issue is how to transfer this expertise into object-oriented concurrent programming. As pointed out in section 1.4, object-oriented concurrent programming is a generalization of object-oriented programming. However the concepts of object-oriented programming need some evolution to fit well with the concept of concurrency.

Another problem is the relative immaturity of the field of OOCP. This results in the abundance of prototype OOCP languages [OOCP 87]. We remarked that it is not always easy to analyze and compare these languages and discuss their design decisions. Comparing various semantics needs to abstract from various syntax and implementation supports. Note that description, classification and classification abilities are supported by object-oriented programming. This led us to design some platform based on OOP (namely Smalltalk-80 [Goldberg and Robson 83]) to model OOCP languages.

2.3 The Actalk Platform

This platform introduces actors into the world of objects. We chose the Smalltalk-80 programming language and environment for reasons discussed below as the OOP basis. Consequently we named our platform *Actalk*, which stands for *act*ors in Small*talk*

[Briot 89]. Actalk provides support for a comprehensive description of object-oriented concurrent programming. It has been used as the experimental basis of numerous courses on OOCP. We will first discuss the design decisions which led to the actual system. We will then survey the status of the system. Then we will discuss its pedagogical purposes. A deeper discussion of the approach leading to Actalk may be found in [Briot 90].

2.3.1 Design Decisions

To design a system for integrating various OOCP languages into a single environment, we had the following goals in mind:

- *uniformity and modularity*
  We wanted to unify various OOCP languages into some common environment and to be able to analyze and define them incrementally. Therefore we chose the object-oriented paradigm as a basis for matching these two first goals. We decided to introduce actors into standard (sequential) OOP, by defining a sub-world of actors embedded into the world of objects, and without changing the underlying object system.
- *minimality and extensibility*
  We wanted some minimal kernel expressing the most general semantics of OOCP languages, and to further extend it in order to simulate various existing languages. Therefore we chose a minimal architecture based on a minimal kernel. We then further extend it by inheritance to simulate and classify various OOCP models.
- *an integrated environment*
  We didn't want to restrict our system to some semantic model and raw implementation, but also to provide a full environment for pragmatic experiment with OOCP. Therefore we chose Smalltalk-80 because it is the most achieved and flexible OOP system with a fully integrated programming environment.

By choosing Smalltalk-80 we provide a minimal implementation of the system, because all entities needed to build actors: objects, classes and messages, and to express concurrency: processes and semaphores, are provided by Smalltalk-80. Thus implementation is both high level and minimal. This allows focusing on the semantics of OOCP and the differences with standard OOP, rather than dealing with low level concerns.

2.4 Pedagogy

Actalk is an integrated environment, embedded into Smalltalk-80, used as a testbed to classify and design OOCP languages. Because of the optimal integration of actors within the world of objects, Actalk is used as a basis for studying relation and combination between objects and actors. Actalk has been successfully used in many courses, seminars and tutorials to describe how to move from object-oriented programming to object-oriented concurrent programming. The minimal implementation of the kernel of Actalk helps making concrete and manageable the concepts of OOCP.

Actalk has also been successfully used to describe some of the most representative OOCP computation models and languages. These extensions use inheritance to customize the basic OOCP semantics defined by the Actalk kernel. Because the kernel and its extensions are related by inheritance, one could easily compare them. Examples of extensions include the Actor model of computation defined by Gul Agha [Agha 86], the ABCL/1 programming language designed by A. Yonezawa and his team [ABCL 89], and the POOL language [America 87]. Simulation of other concurrent languages (such as CSP) have also been conducted by students in the team of Pr. Jean Bézivin at University of Nantes.

Actalk has also been used to build various prototype multi-agent systems in the team of Pr. Jacques Ferber at the Laforia research laboratory in Institute Blaise Pascal, Paris. One of them is the Mages multi-agent testbed [Bouron et al. 90]. These prototype multi-agent systems have been applied to various combinatorial and cooperation problems.

All these experiences convince us of the goods of our approach based on a progressive and layered description of OOCP concepts, and practical experiment with them, thanks to the Actalk platform (built on the wonderful environment provided by Smalltalk-80). We are currently in the process of writing a book on object-oriented concurrent programming based on this experience.

Conclusion

We introduced the Object-Oriented Concurrent Programming methodology (OOCP), which helps decomposing programs and running them onto new multi-computers. We discussed the issue of introducing this new leading methodology to programmers and non-programmers. Therefore we described how we designed a modelization platform, named Actalk. This platform has been validated as an useful tool for description and experiment with OOCP. We described how it is used for pedagogical purposes. We expect to get further experience in the process of teaching and experimenting with this new programming methodology.

This article is dedicated to a Gemini named Dominique. Thanks to Vincenzo Loia for giving us opportunity to think over this issue of teaching new programming methodologies.

References

[ABCL 89] "ABCL: an Object-Oriented Concurrent System", edited by A. Yonezawa, Mit-Press, 1990.

[Agha 86] G. Agha, "Actors: A Model of Concurrent Computation in Distributed Systems", Series in Artificial Intelligence, MIT Press, 1986.

[America 87] P. America, "POOL-T: A Parallel Object-Oriented Language", Pages 199-220, in [OOCP 87].

[Andrews and Schneider 83] G.R. Andrews and F.B. Schneider, "Concepts and Notations for Concurrent Programming", Computing Surveys, Vol. 15, No 1, March 1983.

[Barry 89] B. Barry, "Prototyping a Real-Time Embedded System in Smalltalk", Proceedings of the Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'89), Special Issue of Sigplan Notices, Vol. 22, No 12, pages 255-265, October 1989.

[Berry and Boudol 90] G. Berry and G. Boudol, "The Chemical Abstract Machine", ACM Conference on Principle of Programming Languages (POPL'90), San-Francisco CA, USA, 1990.

[Bouron et al. 90] T. Bouron, J. Ferber, and F. Samuel, "Mages: a Multi-Agent Testbed for Heterogeneous Agents", Proceedings of MAAMAW'90, Decentralized AI, Vol. II, North-Holland, 1991.

[Briot 89] J.-P. Briot, "Actalk: a Testbed for Classifying and Designing Actor Languages in the Smalltalk-80 Environment", Proceedings of the European Conference on Object-Oriented Programming (ECOOP'89), pages 109-129, Cambridge University Press, 1989.

[Briot 90] J.-P. Briot, "OOCP = OOP + C", Proceedings of the 3rd Conference on Object-Oriented Languages and Systems (Tools Pacific'90), pages 417-421, Sydney, Australia, November 1990.

[DAI 88] "Readings in Distributed Artificial Intelligence", edited by A. Bond and L. Gasser, Morgan Kaufman, 1988.

[Dally et al. 89] W.J. Dally et al., "The J-Machine: a Fine-Grain Concurrent Computer", Proceedings of Information Processing Congress (IFIP'89), pages 1147-1153, August 1989.

[ECO 88] "The Ecology of Computation", edited by B.A. Huberman, North-Holland, 1988.

[Goldberg and Robson 83] A. Goldberg and D. Robson, "Smalltalk-80: The Language and its Implementation", Series in Computer Science, Addison-Wesley, 1983.

[Goldberg 89] D.E. Goldberg, "Genetic Algorithms", Addison-Wesley, 1989.

[Meyer 88] B. Meyer, "Object-Oriented Software Construction", Series in Computer Science, Prentice-Hall, 1988.

[OOCP 87] "Object-Oriented Concurrent Programming", edited by A. Yonezawa and M. Tokoro, MIT Press, 1987.

[PDP 86] "Parallel Distributed Processing", edited by D.E. McClelland and J.L. Rumelhart, Vol. 1 and 2, MIT Press, 1986.