Functional-logic programming as an antidote to Java in computer science education (extended abstract)

Peter Van Roy[1,2]

[1] Université catholique de Louvain
[2] Swedish Institute of Computer Science
pvr@info.ucl.ac.be, http://www.info.ucl.ac.be/people/cvvanroy/

We are writing a textbook on computer programming that brings the student a comprehensive and up-to-date presentation of all major programming concepts and techniques. The concepts and techniques are organized into a series of computation models, or less accurately, programming paradigms. We define the precise concept of *computation model* to capture the intuitive concept of *programming paradigm*. All models are fully implemented for practical programming (in the Mozart Programming System [4]), are formally defined, and incorporate the latest research ideas. In spirit, the textbook is a successor to Abelson & Sussman [1].

1 Limits of object-oriented programming

The current trend in computer science education is to restrict the student to one or two models. The most extreme case is where a single rather complex model and language, namely object-oriented programming in Java, is used as a general-purpose approach with which all problems should be solved. This trend is driven by market forces and has no scientific basis. One goal of the book is to be a counterweight to this trend, showing programming in its full richness as a scientific discipline. Many problems that are hard to solve in object-oriented languages become simple when multiple computation models are available.

2 How concepts lead to "paradigms"

The book introduces computation models progressively in a uniform framework. The simplicity of the framework lets us explore the deep relationships between the models. We examine the uses and limits of each model. We show how and why to use different models together in the same program. This happens in a completely natural way when programming (as a kind of epiphenomenon), depending on which concepts one uses for different parts of the program.

The simplest paradigm in the uniform framework is the *declarative model*, which models both strict functional programming and deterministic logic programming. We show how the declarative model can be naturally extended with dataflow concurrency, lazy execution, search (for Prolog-style nondeterministic logic programming), encapsulated state, and object-orientation. We give a complete formal semantics of the extended model, showing how all these concepts fit together. For the programmer, this allows to choose the right paradigm for each part of a program and reason in that paradigm. For the student, this means that the extended model is a suitable tool for learning programming concepts. All concepts work together smoothly with no unpleasant surprises.

3 Practical advantages

Using a simple declarative model as base gives many practical programming advantages. This is profusely illustrated in the book. Four examples of intrinsic interest are:

- A practical approach to concurrency that is much simpler than the standard monitor-based approach.
- A mixed declarative/imperative approach to user interface design that is well-suited for context-sensitive interfaces [2].
- An efficient approach to network-transparent distributed programming [3].
- An advanced approach to constraint programming that is based on encapsulated search [5].

All these examples clearly illustrate the gains to be had by using multiple models in one program. In addition to examples such as these, the book has an in-depth discussion of the uses and limits of declarative programming.

4 Teaching plans

The latest draft of the book is always available on the Web [6]. While this draft has much material, it is still subject to major modifications as we incorporate feedback from courses that use it. The book will be teach-tested this academic year in three universities, in both second-year and fourth-year programming courses.

Acknowledgements

This research is partially financed by the Walloon Region of Belgium. We thank all the contributors and developers of the Mozart Programming System.

References

1. Harold Abelson, Gerald Jay Sussman, and Julie Sussman. *Structure and Interpretation of Computer Programs, Second Edition.* MIT Press, Cambridge, Mass, 1996.

2. Donatien Grolaux, Peter Van Roy, and Jean Vanderdonckt. `QTk` – a mixed declarative/procedural approach for designing executable user interfaces. In *8th IFIP Working Conference on Engineering for Human-Computer Interaction (EHCI'01)*, Lecture Notes in Computer Science, Toronto, Canada, May 2001. Springer-Verlag. Short paper.
3. Seif Haridi, Peter Van Roy, Per Brand, and Christian Schulte. Programming languages for distributed applications. *New Generation Computing*, 16(3):223–261, May 1998.
4. Mozart Consortium. The Mozart Programming System version 1.2.0, May 2001. Available at `http://www.mozart-oz.org/`.
5. Christian Schulte. *Programming Constraint Inference Services*. PhD thesis, Universität des Saarlandes, Fachbereich Informatik, Saarbrücken, Germany, 2000.
6. Peter Van Roy and Seif Haridi. *Concepts, Techniques, and Models of Computer Programming – with Practical Applications in Distributed Computing and Intelligent Agents*. 2002. Work in progress. Draft available at `http://www.info.ucl.ac.be/people/PVR/book.html`. Expected publishing date 2002.