Teaching Functional Programming and Erlang: The Galician Experience*

Victor M. Gulias

MADS Group, Department of Computer Science
University of A Coruña, Spain
gulias@dc.fi.udc.es

Abstract. In this paper, we present the experience of teaching functional programming in the Computer Engineering programme of a Galician University, the University of A Coruña. Erlang is introduced as part of an optional course on functional programming as an example of application of functional paradigm to the real world and most of the students really appreciate the beauty of the language when describing concurrent systems. In fact, several students choose Erlang every year as development framework for their Master's Thesis or for conducting research in our doctorate programme.

1 Introduction

In this paper, we present our experience of teaching functional programming in the Computer Engineering programme of a Galician University, the University of A Coruña. The functional paradigm plays an important role in the whole programme and Erlang [AWWV96] has become a relevant part of an optional fourth-year course on functional programming.

Besides academic topics (such as λ-calculus, type systems and so on) and academic languages (Objective Caml, Haskell), Erlang is introduced as part of the course as an example of application of functional programming in the real world and most of the students really appreciate the beauty of the language when describing concurrent systems. In fact, several students choose Erlang every year as implementation language for their Master Thesis or for conducting research in our doctorate programme in front of more popular languages such as Java or C/C++.

The paper is structured as follows. In the next section, we present a general overview of the teaching environment, introducing University of A Coruña and its Faculty of Informatics as well as the role played by declarative programming in the studies. Section 3 presents the general outline of the functional programming course. Section 4 presents some remarks based on of personal experience teaching Erlang. Finally, we conclude.

* Partially supported by ERDF and Spanish MEC TIN2005-08986.

2 The Environment

2.1 University of A Coruña and its Faculty of Informatics

The University of A Coruña (UDC, `http://www.udc.es`) was established in 1989 as scission of an older one, the University of Santiago de Compostela. UDC holds more than 25.000 students spread over seven campuses in two different Galician cities (A Coruña and Ferrol). Faculty of Informatics (`http://www.fic.udc.es`) is located in A Coruña, a city just by the Atlantic coast in the north west of Spain with about 300.000 people in the metropolitan area. It is one of the most important and largest schools at UDC with about 2.500 students (both undergraduate and graduate). Curiously, Faculty of Informatics is older than UDC, being established in 1986 as part of University of Santiago de Compostela.

Nowadays, the Faculty of Informatics holds three different study programmes in computer science –Engineering (five years) and two Technical Engineerings (three years)– as well as several doctorate programmes for graduate students.

2.2 The Role of Declarative Programming in the Studies

Declarative paradigm plays a relevant role in the study programmes offered by Faculty of Informatics. Students of fourth (or fifth) year can choose a specific optional course on Functional Programming (described in detail in section 3). However, students get in touch with functional paradigm previously. There is a mandatory course on declarative programming in the second year (three hours in classroom and two in the laboratory per week during one semester). Here, the students learn two different programming models: logical (Prolog) and functional (Objective Caml, the french dialect of ML family).

Besides this mandatory course, students use Objective Caml as the implementation language for several other courses such as *Programming Technology*, *Automata and Formal Languages*, *Artificial Intelligence*, *Compilers*, etc. In some courses, it is a student's choice the implementation language for practical exercises and people usually agree that it is easier to use a functional language than using other more popular languages such as Java, C/C++ or any of the other nine different languages that students must learn in the first three years. In fact, sometimes you have to smile when other lecturers explicitly forbid the use of functional languages for students' duties because *"that way the proposed exercises are just too easy"*.

2.3 Why Objective Caml?

At the end of the eighties, a research group (LFCIA, `http://www.lfcia.org`), with strong mathematical background and leaded by former dean Jose L. Freire, was exploring applications of category theory. At that time, there was an implementation of an ML dialect based upon this formalism, the *categorical abstract machine* [CCM85] used by the language CAML [Mau91]. Hence, this language was gradually introduced in lectures replacing other options at that time such

as Edinburgh's Standard ML or Miranda. Though the language evolved (Leroy's Caml Light and finally Objective Caml), dropping its categorical foundation, it was the language of choice for teaching functional programming.

Objective Caml [Ler00] (O'Caml for short) has several nice features for teaching: it is efficient (with a native code compiler), portable (both Windows and particularly Linux are usual target platforms), with most of the state-of-the-art functional features (static typing, anonymous functions, parametric polymorphism, powerful module system, partial application, and so on), and a reasonable set of libraries for the development of interesting applications (from the teaching point of view).

3 The Functional Programming Course at UDC

Functional programming is an optional course that is usually taken by students in the fourth year (fall semester). The student load is three hours in the classroom and two in the laboratory per week during 14 weeks (equivalent to 5.0 ECTS). It is a popular optional course, with about 40 people per year (about 25% of fourth year students), even though it has a strange (crazy?) schedule according to European customs (for example, a laboratory session from 8pm to 10pm at friday night). The course is divided into five different parts:

1. **Quick review of functional programming concepts**. Using O'Caml, the first two weeks are used to refresh previous knowledge on functional programming. Topics of interest: (a) Values, types and expressions; (b) Identifiers, definitions and scope; (c) Predicates, conditional expressions and pattern-matching; (d) Recursion; (e) Lists; (f) High-order functions, partial application; (g) Polymorphism; (h) User-defined and abstract datatypes; (i) Modules and Functors.
2. **Introduction to λ-Calculus**. In order to better understand the fundamentals of functional programming, we present the λ-calculus in three-four weeks. Using O'Caml the student implements a small interpreter. After studying different evaluation models, a short presentation of Haskell let us introduce a language with lazy evaluation. Topics: (a) Pure λ-calculus; (b) Substitution and reduction rules; (c) Normalization; (d) Lazy vs. eager evaluation; (e) Fix-point combinators and recursion; (f) Extending the λ-calculus.
3. **Type Systems**. We extend the pure λ-calculus introducing values and an static type system which helps to understand how type inference works. During three-four weeks, the students use O'Caml to add static typing to their own interpreters.
4. **Implementation Details of Functional Compilers**. Depending on students' interest and available lecturers, we spend one-two weeks with (some of) these topics:
   (a) Garbage collection algorithms
   (b) Internal representation and communication with low-level languages

(c) Pattern-matching implementation

5. **Functional programming in the real-world**. That means **Erlang**. At the beginning, it was a short seminar of two hours but students interest suggested to offer a two-three weeks course on concurrent functional programming in Erlang. This module of the course covers most of the topics proposed by basic and continuation Erlang courses, though with a quite different approach due to the student background.

Since 1999, Victor M. Gulias has been in charge of the course even though right now he only conducts the coordination of the whole course and the teaching of the Erlang part.

4 Experience Teaching Erlang

Some general impressions of the students when teaching Erlang:

- After learning a modern functional languages such as O'Caml, Erlang looks somehow primitive: no static typing, no modules, no partial application, etc.
- After a first contact with the compiler in the laboratory, they experiment the "type freedom" effect and start loving the language. Nevertheless, the use of a typed language before seems to be present in their minds: most of them use comments to "informally" state function signatures. In fact, as they get into more complex problems, they miss the type facilities and point out this as the more important problem of Erlang.
- Panic! They are frighten when first hear the term "message passing". They have experimented, in previous courses, the development of parallel applications using infamous low-level libraries such as mpi or pvm. They are amazed that Erlang's runtime system gets in charge of all the marshalling of data. World starts becoming a better place to program distributed applications.
- The main goal of the functional programming course is *abstraction*. They appreciate examples such as the generalization of a server, separating the imperative part (recursion loop and communication) from the definition of the particular services, or the definition of skeletoms such as an abstract divide-and-conquer algorithm.
- The notion of *behaviour* is very well accepted as they are learning (from the same lecturer and at the same time!) classical GoF design patterns [GHJV95] with Java implementation in a mandatory fourth-year *System Design* course. At this moment, students ask why Erlang is not being used in the system design course with (or instead of!) Java.
- Though in the course we try to use familiar design artifacts (such as UML's state or sequence diagrams), students feel that design tools and techniques do not have Erlang in mind.
- Erlang students, when have to implement simple concurrent programs in Java in the system design course, just miss the good days programming in Erlang... but at the end they get better and cleaner solutions than regular students. Even some of them use Erlang to quick implement a solution to better understand the problem.

5 Conclusions

Erlang is an excellent example of a real-world success of functional paradigm and, in the case of University of Coruña, is also a success case among students. The Erlang part of the functional programming course is really appreciated by students and they demand even more time dedicated to Erlang-related topics. "Ericsson", "complex real-time applications", "distributed programming", in summary "Real-world stuff", are by far the best advertisement for Erlang. Students that really get into the language feel that they are going backwards when returning to tradicional but more popular approaches such as JAVA. Some of the students continue working with Erlang by means of their Master's Thesis, where they can conduct a larger Erlang-based project, or joining the doctorate programme where they can access to additional lectures involving Erlang.

In order to measure the impact of the language in the students, we should count several Master's Thesis in the last few years (about 2-3 per year), at least one large research project related with Erlang (VoDKA project [GBF05]) and several on-the-way Ph.D. Thesis. In addition, at least three SMEs has been recently created using Erlang/OTP as platform for the development of their products, which is a notable success indicator if we take into account the poor industry development of Galicia region.

References

[AWWV96] J. L. Armstrong, M. C. Williams, C. Wikström, and S. R. Virding. *Concurrent Programming in Erlang*. Prentice Hall, 2nd edition edition, 1996.

[CCM85] G. Cousineau, P. Curien, and M. Mauny. The categorical abstract machine. In J-P. Jouannaud, editor, *Proceedings Functional Programming languages and Computer Architecture*, volume 201 of *LNCS*, pages 50–64. Springer-Verlag, 1985.

[GBF05] V. Gulias, M. Barreiro, and J. Freire. Vodka: Developing a video-on-demand server using distributed functional programming. *Journal of Functional Programming*, 15(3):403–430, may 2005.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Software*. Addison Wesley, 1995.

[Ler00] X. Leroy. The Objective Caml system: Documentation and user's manual, 2000. With Damien Doligez, Jacques Garrigue, Didier Rémy, and Jérôme Vouillon. Available from `http://caml.inria.fr`.

[Mau91] M. Mauny. Functional programming using CAML. Technical Report RT-0129, Inria, Institut National de Recherche en Informatique et en Automatique, 1991.