RECYCLING PREVIOUS DOCUMENTS FOR DISTANCE EDUCATION

Jean-Michel Hufflen
LIFC (EA CNRS 4269), University of Franche-Comté, 16, route de Gray, 25030 Besançon Cedex, France

Keywords: Presentational education, Distance education, Course text, On-line course, Case study, LaTeX, PDF.

Abstract: Given a course document concerning a teaching unit in Computer Science and written for presentational education, we explain how we took as much advantage as possible of it for the same teaching unit, but adapted to distance education. In particular, whenever we are building a new version, we are able to update this document once, so that changes are automatically applied to both versions, for presentational and distance education. If the original document is clearly written and well structured, the adaptations we propose should be easy to be put into action.

1 INTRODUCTION

As result of greater and greater interest in distance education, most universities have increased such offerings. Since this field aims to deliver education to students who are not physically on site, it is of interest for students who have a full-time job or are very distant, possibly living in another country. As an example of an institution delivering distance education, the CTU[1], part of the University of Franche-Comt, allows students to get the whole of the units of a master in Computer Science. Of course, this university still provides curricula in presentational education, which remains the 'traditional' way of teaching.

We are in charge of a teaching unit for four-year university students in Computer Science. First this unit has been launched in presentational education, then it has also been offered as part of the curriculum in distance education. This article aims to explain how we take as much advantage as possible of documents written for 'traditional' students, in order to recycle them for 'distant' students. These documents were written using LaTeX (Mittelbach et al., 2004), but reading this article only requires basic knowledge of this typesetting system. First we describe the situation when our distance education unit began, then we explain our choices and give an overview of our tools. After a short mention of alternative solutions, we conclude with sumarising the experience we have got.

[1] ***C**entre de **Tl**-**E**nseignement*, that is, *Centre for Teleteaching*.

2 OUR TEACHING UNIT AND ITS DOCUMENTS

Our teaching unit is entitled *Advanced Functional Programming*, PFA for short[2]. Let us recall briefly that *functional programming* emphasises functions' application, whereas *imperative programming*—the paradigm implemented within more 'traditional' languages—emphasises changes in state. Many universities include courses about functional programming, examples being reported in (Thompson and Hill, 1995). Going back to the title, 'advanced' means that this unit is not for beginners in programming, students are supposed to be experienced. Practically, most students attending this unit have already programmed in Java (java, 2008), Scheme (Springer and Friedman, 1989), and C++ (Stroustrup, 1991).

Functional programming languages have a common root as the λ-calculus, a formal system developed in the 1930's (Church, 1941). However, these programming languages are diverse, some—e.g., the Lisp dialects[3]—are dynamically typed, some—e.g.,

[2] ***P**rogrammation **F**onctionnelle **A**vance*, in French.

[3] 'Lisp' stands for '**LIS**t **P**rocessing, because major structures are linked lists. The first version came out in 1958 (McCarthy, 1960) and has many descendants, the most used nowadays being COMMON LISP (Steele et al., 1990) and Scheme.

Hufflen J. (2010).
RECYCLING PREVIOUS DOCUMENTS FOR DISTANCE EDUCATION.
In *Proceedings of the 2nd International Conference on Computer Supported Education*, pages 469-472
DOI: 10.5220/0002801604690472

Standard ML[4] (Paulson, 1996), CAML[5] (Leroy et al., 2004), Haskell[6] (Peyton Jones, 2003)—are strongly typed and include a type inference mechanism.

Our unit's first part is devoted to the λ-calculus' bases (Hufflen, 1998). Then all the practical exercises are performed with only one language, Scheme. The starting point of the most important part: when we begin to program, the language we are learning is always shown as *finite product*. It has precise rules, precise semantics, and is *consistent*. According to the language used, some applications may be easy or difficult to implement. When you put down a statement, running it often results in something predictible. That hides an important point: a language results from some important choices our unit aims to emphasise. For example, if the language is lexical (resp. dynamic), what kinds of applications are easier to implement? Of course, answers of such questions depend on the programming languages considered. Our strategy consists of explaining the choices of Scheme, and we demonstrate alternate solutions using other functional programming languages such as COMMON LISP or Standard ML. After this main part, our unit ends with some advanced features of Scheme: delayed evaluation, continuations, macros.

There is a big document grouping what is taught within this unit, the first version was (Hufflen, 1997). It consists of six chapters. Each chapter includes exercises, given with model solutions. These chapters are followed by several appendices, making precise some extra information or devoted to lab classes done by students. The whole document is approximately 400-page long. It can be viewed as a textbook, even if its diffusion is limited to this unit's students. As mentioned in the introduction, we wrote it using LaTeX, which seems to us to be the best typesetting system for large documents: cross references are widely used throughout this textbook, and there is a rich 'Bibliography' section. Students progressively are given the successive parts of this document, but it is organised as a whole, with precise architecture. Of course, it contains not only texts—in the sense of successive paragraphs—but also many examples of programs and some mathematical formulas, even if it is not really a textbook in Mathematics.

[4] 'ML' stands for '**M**eta**L**anguage' and has been initially developed within the formal proof system LCF (**L**ogic for **C**omputable **F**unctions) (Gordon et al., 1979). Later on, it appears as an actual programming language and its standardisation resulted in the Standard ML language.

[5] **C**ategorical **A**bstract **M**achine **L**anguage.

[6] Named after Haskell Brooks Curry (1900–1982).

3 ADAPTATION TO DISTANCE EDUCATION

3.1 Situation

When the distance master was launched, its curriculum obviously resembled master's in presentational education. But a unit common to these two curricula was not necessarily in the charge of the same teacher. In other words, we have been in charge of the PFA unit within both presentational and distance education, but this arrangement did not hold true for all the units. Besides, we were still in charge of the 'presentational' unit. So we were interested in a method that would allow us to derive the two versions—printed and online—from the same source. Some slight mistakes, especially typing ones, should be fixed, we wished to add more examples. In addition, the version of standard Scheme changed (Kelsey et al., 1998), so we ought to adapt some existing examples. If we considered our text, there were only two differences to be managed. The first difference was located at the introduction to Scheme: since most of presentational students attended a unit for beginners in functional programming for the 2nd-year university degree, this introduction was just some revision. On the contrary, most of distant students do not know Scheme, and a suitable introduction should be more progressive. But this point was not really difficult since LaTeX allows the definition of *conditional texts*; for example[7]:

```
\ifpfaforde...  (For distance education students.)
 \else...       (For presentational education ones.)
\fi
```

The second difference is related to exercises. Presentational students get the successive texts at the end of each chapter, so model solutions may be given after each exercise, especially if this exercise has already been proposed at classes. That cannot be the same for a document devoted to distance education: model solutions should be grouped at the end of each chapter, or provided in separate files. Here also, if these model solutions have been put into separate source files, an '`\if...`' command of LaTeX may allow us to put model solutions at distinct places, according to the document we are building, for presentational or distance education.

[7] To avoid clashes among LaTeX names, the new commands related to our adaptation are prefixed by '`\pfa...`' or '`\ifpfa...`'.

3.2 Difficulty

When distance education was launched, teachers were obviously asked to put on-line documents on the Web. Some teachers put documents using HTML[8]. However, such a choice seemed to us not suitable for scientific documents: the look of resulting Web pages depends on the browser used; in addition, formatting mathematical formulas and program fragments often results in poor-quality output. In our case, this last point was essential about the fragments given in languages other than Scheme. We could perform some demonstrations during the lab classes of presentational students, so they could observe these other programs' behaviour. The same *modus operandi* was impossible for distant students, and it was difficult to ask them to install many compilers or interpreters. So the solution was to ask them for exercises only in Scheme—as done for presentational students—but the examples given throughout our text must be explicit, in order for these students to understand without running them.

As abovementioned, our document was typeset by LaTeX, so an acceptable solution was to use pdfLaTeX, able to produce PDF[9] files. In addition, if the hyperref package is used, PDF files produced by pdfLaTeX can support hyperlinks, as in HTML. But obviously, we could not provide a single document, as a huge PDF file. It is preferable for distant students to get separate medium-sized PDF files, according to the steps of their planning. Besides, let us not forget that these files are downloaded: students cannot be asked to download a huge file again if only some typing mistakes have just been fixed. Splitting this big document into separate files induces a precise organisation of cross-reference links throughout the original version.

3.3 Our Adaptations

Let us assume that the chapters, sections, etc. of the two versions—printed and on-line—are numbered identically. Besides, LaTeX allows each chapter of a document to be associated with its own auxiliary (.aux) file, containing information solving cross-references[10]. So we can compile a chapter for the on-line version by using the auxiliary files of the document's other chapters. A cross-reference put by LaTeX's `\ref` command is implemented in pdfLaTeX as an internal hyperlink, what is fine for cross references within the same chapter. For cross-references to another chapter's part, we define a new command:

`\pfaexternalref[`*`chapter-file`*`]{`label_0`}`

If the big document for presentational education is generated, this works like `\ref{`label_0`}`. If the chapter is generated as part of the on-line text, an link to the PDF file *`chapter-file`* is put. In both cases, the same text is displayed. That means that label_0 is a label identifying a resource belonging to a file used to build the file *`chapter-file`*. This file has been declared by the `\pfaexternaldocument` command, so label_0 is known as a label. Of course, when we started this task, such a choice led us to look for all the occurrences of the `\ref` command and change some into `\pfaexternalref` ones. In practice, that was not difficult, because a good technique is to prefix labels' name by an identifier for the corresponding chapter. So the file name to be put was not difficult to supply. We use a similar technique for cross-references to the bibliography, and to footnotes belonging to another chapter. All these new commands have been grouped into a package.

4 DISCUSS

4.1 Students' Opinion

As far as we know, students' feedback is globally positive. In fact, they quickly perceive that PDF files allow them to watch exactly what teachers want to express, like in a book or blackboard. Our document giving many 'cultural' complements, we had to define typographical signs to mark up what is important and what may be skipped in a first reading, but this task can be performed progressively. It also seems that the hyperlinks pointing to a part of the current chapter are most useful, so pointing to the beginning of another chapter does not cause much trouble.

4.2 Comparison with other Methods

We were obviously interested in reusing our first version written in LaTeX. There are some converters from LaTeX to HTML (Goossens et al., 1999), and they allow a base document to be split into several Web pages. However, these converters are not suitable when we update an existing text since the names of generated Web pages are generated, too; it may be difficult to point just the HTML files that have been changed.

If we proceed from scratch, an interesting method

[8] **H**yper**T**ext **M**arkup **L**anguage. (Musciano and Kennedy, 2002) is a good introduction to it.

[9] **P**ortable **D**ocument **F**ormat, Adobe's format.

[10] That can be done by the commands `\includeonly` and `\include`.

could be to specify our input files using XML[11], which has become a standard for information exchange and provides a rich toolbox. XSLT[12] (W3C, 2007), the language commonly used for transformations of XML texts could be used to derive texts for LaTeX, or in XSL-FO[13] (W3C, 2006), an XML language that aims to describe high-quality print outputs. However, the current XSL-FO processors—generating PDF files—are not complete yet, even if they implement most of this recommendation, so using XSL-FO is interesting for experiment, but not for intensive use by students.

5 CONCLUSIONS

As abovementioned, the first complete version of our course text came out in 1997. Then it has evolved deeply—chapters and appendices have been wholly revised—and continuously, since we have applied some changes each year. We did it successfully, so we can think that our system is reliable. Of course, even if our new commands could be applied throughout any document, it may be noticed that this document must be a LaTeX source text. This not too restrictive for documents in Computer Science or Mathematics, since LaTeX is widely used within these communities. Further expriment should be made about documents concerning other topics.

ACKNOWLEDGEMENTS

I thank the distance education students who addressed me very constructive criticisms. Year after year, they indirectly helped me improve my tools.

REFERENCES

Church, A. (1941). *The Calculi of Lambda-Conversion*. Princeton University Press.

Goossens, M., Rahtz, S., Gurari, E. M., Moore, R., and Sutor, R. S. (1999). *The LaTeX Web Companion*. Addison-Wesley Longman, Inc., Reading, Massachusetts.

Gordon, M. J., Milner, A. J., and Wadsworth, C. P. (1979). *Edinburgh* LCF. Number 78 in LNCS. Springer-Verlag.

Hufflen, J.-M. (1997). Programmation fonctionnelle avancée. notes de cours et exercices. Polycopié. Besançon.

Hufflen, J.-M. (1998). Introduction au λ-calcul (version révisée et étendue). Polycopié. Besançon.

java (2008). *Java Technology*. http://java.sun.com.

Kelsey, R., Clinger, W. D., Rees, J. A., Abelson, H., Adams iv, N. I., Bartley, D. H., Brooks, G., Dybvig, R. K., Friedman, D. P., Halstead, R., Hanson, C., Haynes, C. T., Kohlbecker, Jr, E. E., Oxley, D., Pitman, K. M., Rozas, G. J., Steele, Jr, G. L., Sussman, G. J., and Wand, M. (1998). Revised[5] report on the algorithmic language Scheme. HOSC, 11(1):7–105.

Leroy, X., Doligez, D., Garrigue, J., Rémy, D., and Vouillon, J. (2004). *The Objective Caml System. Release 0.9. Documentation and User's Manual*. http://caml.inria.fr/pub/docs/manual-ocaml/index.html.

McCarthy, J. (1960). Recursive functions of symbolic expressions and their computation by machine, part I. *Communications of the ACM*, 3(4):184–195.

Mittelbach, F., Goossens, M., Braams, J., Carlisle, D., Rowley, C. A., Detig, C., and Schrod, J. (2004). *The LaTeX Companion*. Addison-Wesley Publishing Company, Reading, Massachusetts, 2 edition.

Musciano, C. and Kennedy, B. (2002). HTML & XHTML: *The Definitive Guide*. O'Reilly & Associates, Inc., 5 edition.

Paulson, L. C. (1996). ML *for the Working Programmer*. Cambridge University Press, 2 edition.

Peyton Jones, S., editor (2003). *Haskell 98 Language and Libraries. The Revised Report*. Cambridge University Press.

Ray, E. T. (2001). *Learning* XML. O'Reilly & Associates, Inc.

Springer, G. and Friedman, D. P. (1989). *Scheme and the Art of Programming*. The MIT Press, McGraw-Hill Book Company.

Steele, Jr., G. L., Fahlman, S. E., Gabriel, R. P., Moon, D. A., Weinreb, D. L., Bobrow, D. G., DeMichiel, L. G., Keene, S. E., Kiczales, G., Perdue, C., Pitman, K. M., Waters, R., and White, J. L. (1990). COMMON LISP. *The Language. Second Edition*. Digital Press.

Stroustrup, B. (1991). *The C++ Programming Language*. Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 2 edition.

Thompson, S. and Hill, S. (1995). Functional programming through the curriculum. In *FPLE '95*, pages 85–102, Nijmegen, The Netherlands.

W3C (2006). *Extensible Stylesheet Language (*XSL*). Version 1.1*. http://www.w3.org/TR/2006/REC-xsl11-20061205/. W3C Recommendation. Edited by Anders Berglund.

W3C (2007). XSL *Transformations (*XSLT*). Version 2.0*. http://www.w3.org/TR/2007/WD-xslt20-20070123. W3C Recommendation. Edited by Michael H. Kay.

[11] e**X**tensible **M**arkup **L**anguage. (Ray, 2001) is a good introduction to this meta-language.

[12] e**X**tensible **S**tylesheet **L**anguage **T**ransformations.

[13] e**X**tensible **S**tylesheet **L**anguage—**F**ormatting **O**bjects.