DOSFOP — A Documentation Tool for the Algebraic Programming Language Opal

Klaus Didrich, Torsten Klein

Technische Universität Berlin, Fachbereich Informatik, Institut für Kommunikations- und Softwaretechnik, Franklinstr. 28/29, D – 10587 Berlin
{kd,parrus}@cs.tu-berlin.de, http://uebb.cs.tu-berlin.de/

Abstract We present an approach to the design of a literate programming tool for the algebraic programming language OPAL, which serves as a back-end in the formal program derivation process. In designing our documentation system we not only take technical aspects into account, but also have the acceptance of the documentation system by the software developer in mind.

1 Introduction

The necessity of documenting software products as soon as they have evolved beyond the stage of mere playthings or examples is evident to most software users and even to most software developers. Nevertheless, documentation is often not available or is outdated, either because the development of actual running software is more important and can be more easily checked by the customer than the quality of the documentation or because the job of keeping the documentation up to date is too arduous.

As part of the OPAL environment, the documentation system DOSFOP ("Documentation system for OPAL projects") was developed. While some features of DOSFOP are specific to OPAL, the main objectives of the documentation system are language independent.

2 The OPAL Environment

The OPAL environment [2] is a software engineering environment based on formal methods, that includes formal specifications as well as efficient implementation in a functional style.

The language OPAL [1] is a strongly typed, higher-order, functional language with a distinctive algebraic flavour, as becomes apparent in the fact that specification constructs are available in the language, in the syntactical appearance of OPAL, and last but not least, in the semantics of OPAL. Specifications consist of laws stating freely generated properties for types and first-order propositional theorems in general.

The OPAL environment has grown in recent years, see [2] for a concise description. The documentation tool sketched in this paper (for a full description see [3]) is one of the recent additions.

3 Design Objectives of DOSFOP

Our research efforts have been focused on ideas for software documentation systems that already exist. Donald E. Knuth introduced the concept of "Literate Programming" [7] with the `web` system, which has been used to write documentation systems for many languages like PASCAL, C, FORTRAN or language independent systems like NOWEB. The most important model for DOSFOP is the GRASP system [5] developed at Glasgow University for the functional language HASKELL.

On the basis of experience with all the `web` derivates and new developments in the field of modern programming languages and their environments, we point out some fundamental requirements of documentation systems that are not satisfactorily covered by existing systems:

Support of Large-Scale Documentation In software engineering, one uses modules to structure the software system. These modules form a hierarchy reflecting the logical relations between modules or groups of modules. We expect a documentation system to support the documentation of a software product in a way that reflects its structure.

Exploit Inherent Documentation The documentation system should use the information that is already contained in the sources. Even for programs with self-explanatory variable and function names we believe that indices, reference tables and the like which refer directly to the elements of the source code should be included.

Provide Multiple Forms of Presentation The documentation should not only be available in print but also in hypertext form. Paper is good for documenting static versions of a software product, but we also need the support of a documentation system in the dynamic stages of a development.

In addition to the technical requirements, we also want to take human weaknesses into account, and emphasise that consideration of these aspects finally decides whether a system will be used in real life or not:

Keep the System Flexible Most programmers are very reluctant to use a system if they feel their individuality is not given consideration. So the documentation system should provide a lot of possibilities to customize the outcome.

Lower the Barriers for Initial Use The initial effort required to use the documentation system must be very small. Ideally, the user would provide documentation information in the proper places and the system would generate the documentation without any further activity on the part of the user.

Provide Compatibility with Existing Code In particular, it must be possible to integrate source code that has not been prepared specifically for the documentation system. So quick-and-dirty programs (which is the way many programs originate) can be integrated and then later on be gradually documented.

4 Description of the DOSFOP System

DOSFOP has the source code as input and also information on the modularization and hierarchy, which is recorded in a *project database*. Moreover, DOSFOP does not produce a uniform documentation format; the user is able to customize the result via *global options* and *local options*. So one can configure the documentation for each OPAL structure and OPAL subsystem individually. The handling of the configuration database as well as the translation process from source code to the final documentation product is supported by a graphical user interface.

On the output side, DOSFOP produces an intermediate output file in the TEXINFO language. This TEXINFO file is translated again into a final representation. The advantage of this approach is that we can use existing translation tools. Currently, DVI files (for printed output), INFO files (for the GNU info help system) and HTML files (for WWW-browsers) are supported. Figure 1 provides a graphical representation of the generation of documentation with DOSFOP.

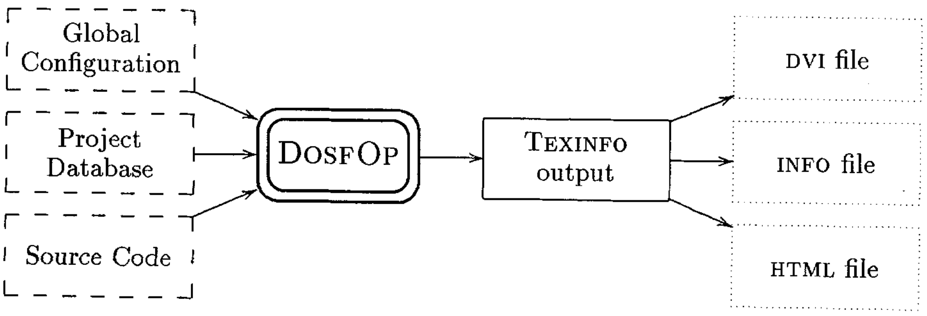

Figure 1: Producing documentation

Documentation of OPAL code is directly included in the source file. After all, this is the idea of literate programming, keeping in mind that a large spatial distance between source code and its documentation entails severe consistency problems. In DOSFOP there are five different kinds of documentation:

- *Ordinary documentation* may contain arbitrary (TEXINFO) text.
- *Tagged documentation* does not appear in the generated documentation unless explicitly specified. This option can be used to generate documentation for different audiences.

- *Documentation sectioning* does not appear in the generated documentation unless explicitly specified. This option can be used to generate documentation for different audiences.
- *References to Properties*, i.e. laws and theorems formally expressed in the specification parts of OPAL, can be referenced by name. The respective formulae are pretty-printed and serve as a mathematical form of documentation.
- *Ignored documentation* finally does not appear at all in the generated documentation.

5 Applications

The DOSFOP system has been successfully used for several projects both at the TU Berlin and Daimler-Benz AG:

- The OPAL standard library (more than 150 structures);
- the ESZ type-checker, part of the ESZ toolkit [6] for editing, typesetting and type-checking Z specifications;
- OPALWIN [4], a library for a window system for OPAL, based on concurrent OPAL;
- DOSFOP itself was of course entirely documented by DOSFOP.

References

1. K. Didrich, A. Fett, C. Gerke, W. Grieskamp, and P. Pepper. OPAL: Design and Implementation of an Algebraic Programming Language. In J. Gutknecht, editor, *Programming Languages and System Architectures*, LNCS 782, pages 228–244. Springer, 1994.
2. K. Didrich, C. Gerke, W. Grieskamp, C. Maeder, and P. Pepper. Towards Integrating Algebraic Programming and Functional Programming: the Opal System. In M. Wirsing and M. Nivat, editors, *Algebraic Methodology and Software Technology*, LNCS 1101, pages 559–562. Springer, 1996.
3. K. Didrich and T. Klein. A Pragmatical Approach to Software Documentation. Technical Report 96–4, TU Berlin, November 1996.
4. Th. Frauenstein, W. Grieskamp, P. Pepper, and M. Südholt. Concurrent Functional Programming of Graphical User Interfaces. Technical Report 95-19, TU Berlin, 1996.
5. The GRASP Team. Glasgow Literate Programming User's Guide, September 1992. Contact: Will Partain.
6. W. Grieskamp. *User's Guide to Editing, Typesetting and Type-Checking Z Specifications with the ESZ Toolkit*, 1996. Bundled with the ESZ distribution.
7. D. E. Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, 1984.