Eelco Visser as a Typographic Designer

Paul Klint
Centrum Wiskunde & Informatica (CWI), Amsterdam, The Netherlands

Abstract

While supervising both Eelco's master's thesis and his dissertation I have witnessed an evolution in his taste and style in many aspects of design, ranging from visual and typographical design to software design and even presentation and article design. I give some personal reflections on this evolution.

2012 ACM Subject Classification Software and its engineering → Documentation

Keywords and phrases Typesetting formal specifications, Typographic design, personal recollection

Digital Object Identifier 10.4230/OASIcs.EVCS.2023.15

1 Prologue

The terminology to describe the relationship between a supervisor and a student is troublesome. The phrase "my student" always reminds me of modern slavery, which it probably is, but should one put that in writing or talk about it loudly? The word "supervisor" suggests an asymmetric relationship where the supervisor is all mighty and all knowledgeable and this is also frequently besides the truth.

Having set this straight, I have had the privilege to "supervise" Eelco on various occasions. The first was a side project about the semantics of Eiffel while he was still a computer science student at the University of Amsterdam; it was completed by the end of 1992 [6]. It was a substantial (125 page) document that could have easily served as a Master's Thesis, but not so in Eelco's opinion. Important stylistic properties: a text-only, red front page and inside ASF+SDF specifications printed in a mono-spaced font.

2 Thesis: Thou Shalt Typeset Specifications

Eelco and I had many interests in common, among them context-free parsing, typesetting, and literate programming. My own dissertation (1982, published in 1985 as [3]) was probably the first computer typeset dissertation in The Netherlands and when I a gave a copy to Edsger Dijkstra he said: "This is the first document where I cannot see that it has been produced by a computer". From his mouth, I took that as a compliment.

This is not the place for a complete genealogy of all specification formalisms we have collectively worked on, but some context is useful. It all started with ASF [1], an algebraic specification formalism with an implementation based on term rewriting. Next came ASF+SDF [5] that added general context-free grammars and user-defined notation. Its implementation was based on GLR parsing and term rewriting. Eelco used his experience with ASF+SDF and the idea of strategic rewriting [4] to design Stratego [9]. Eelco introduced the idea of scannerless parsing in SDF and later evolved it to SDF3 [2].

In the spirit of literate programming I had been working on a program called ToLaTex that took ASF+SDF specifications and turned them into LaTeX source. I was particularly proud about the use of the equation and table features of LaTeX to typeset our specifications. Figure 1 shows a manually formatted ASF specification of the Boolean datatype using a fixed width font. Figure 2 shows the Booleans in ASF+SDF, typeset using ToLaTex.

```
module Booleans
begin
  exports
    begin
      sorts BOOL
      functions
        true :              -> BOOL
        false:              -> BOOL
        or   : BOOL # BOOL -> BOOL
        and  : BOOL # BOOL -> BOOL
        not  : BOOL        -> BOOL
    end

  equations

  [1]  or(true, true)     = true
  [2]  or(true, false)    = true
  [3]  or(false, true)    = true
  [4]  or(false, false)   = false
  [5]  and(true, true)    = true
  [6]  and(true, false)   = false
  [7]  and(false, true)   = false
  [8]  and(false, false)  = false
  [9]  not(true)          = false
  [10] not(false)         = true

end Booleans
```

Figure 1 Booleans in ASF, manual layout.

While Eelco started working on what would become his actual Master's Thesis, I was further improving ToLaTex. Eelco got interested to use and enhance it for his thesis. Eelco's typical first step was to split the single Lisp file into 50 small ones. I see many people performing such "modularization" and I never understand them: it is certainly more modular but at the same time overview and convenient editing are gone. Although not amused by Eelco's action, I took it from the bright side: from then on Eelco was the maintainer of ToLaTeX. In 1993 Eelco's Master's Thesis appeared [7]. Another 130 page tome dense with equations, but stylistically different from the Eiffel document: a nice, black-and-white cover that showed his former training as an artist (shown in Figure 4a). Inside, all specifications had been typeset using ToLaTex that by this time supported subscripts, superscripts and the complete zoo of LaTeX escapes for special characters and mathematical symbols. In later years Eelco continued to work on various techniques for prettyprinting and source code formatting [10]. Eelco continued the same style in his dissertation [8]: a green cover with a design that can easily be taken for a bull's eye merged with trees (shown in Figure 4b). And indeed Eelco was aiming at trees. The 383 page tome is, amongst others, a major ToLaTeX showcase: every feature has been used to the max.

3 Antithesis: Thou Shalt Not Typeset Specifications

Despite the relative success of his dissertation Eelco was disappointed. He ventilated his frustration in a post on a programming-related mailing list[1]. A copy of that post has been hanging in the hallway of our institute for years. Eelco was a programmer and his complaint

[1] I recall this from memory. Although I tried, I was not able to retrieve it.

module Booleans

imports Layout[1.3]
exports
sorts BOOL
context-free syntax
true → BOOL {**constructor**}
false → BOOL {**constructor**}
BOOL "∧" BOOL → BOOL {**left**}
BOOL "∨" BOOL → BOOL {**left**}
"¬" BOOL → BOOL
"(" BOOL ")" → BOOL {**bracket**}
priorities
"¬" > "∧" > "∨"
variables
$P \rightarrow$ BOOL
equations

[1]	$\text{true} \vee P = \text{true}$	[3]	$\text{true} \wedge P = P$	[5]	$\neg\,\text{true} = \text{false}$
[2]	$\text{false} \vee P = P$	[4]	$\text{false} \wedge P = \text{false}$	[6]	$\neg\,\text{false} = \text{true}$

Figure 2 Booleans in ASF+SDF, typeset with ToLaTex.

was that due to the fancy typesetting everybody thought that what he was writing about is math and not programming. It is undeniably true that Greek letters, mathematical symbols, and subscripts and superscripts add to the mathematical flavour of a document. Let's also discuss the issue of readability using the example shown in Figure 3 taken from [8]. Although this is a relatively readable module one thing stands out. Due to the incredible syntactic flexibility of ASF+SDF nearly everything is user-definable, this includes names and variables. For instance $\overline{Al}$, al_1^* and al_2^* are just names (declared in one of the imported modules) while the notation suggests otherwise. The nice typesetting reinforces this effect. In hindsight I think that Eelco's frustration resulted from a combination of ASF+SDF's syntactic flexibility, his actual use of that flexibility, and the fancy typesetting offered by ToLaTex. Since then Eelco gradually gave up on typesetting specifications and started to represent them in a typewriter font. Even to this date, his frustration can be read in the writing advice on his website[2]:

> *Don't use a body text font (such as Times or cmr) for code. Preferably use a proportional font (e.g. some typewritter [**sic**] font), which gives neat alignment and indentation. (Using math for typesetting code, as is done in literate Haskell for example, is not for me. I overdosed on that in my PhD thesis.)*

4 Synthesis

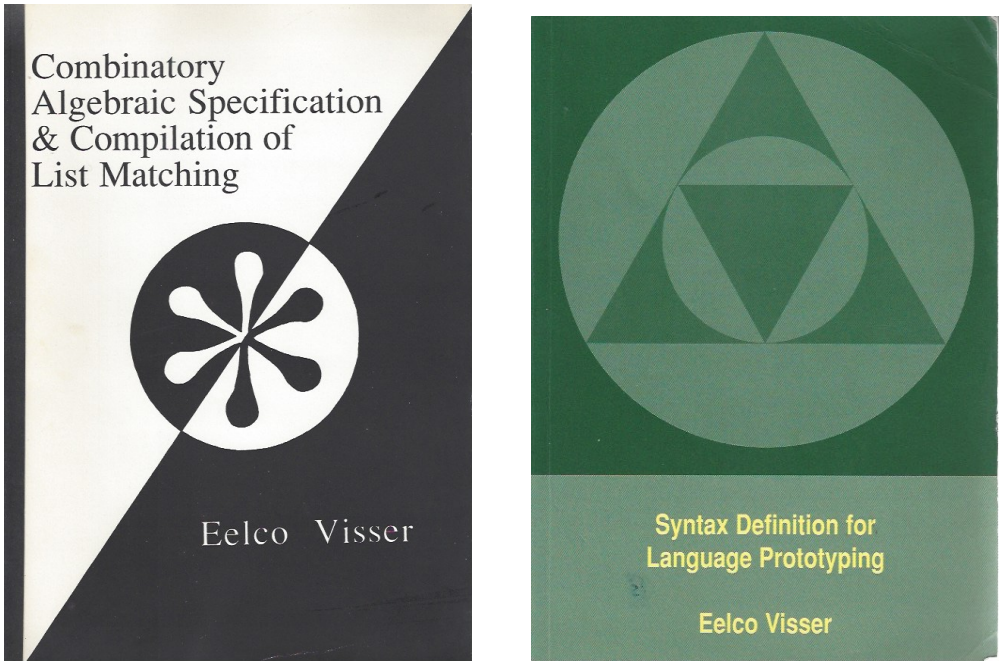
Over the last 5-10 years Eelco's focus has gradually shifted from programming to a more formal view on software development, e.g., safety by construction. Unavoidably, with more formality more complex notation enters the scene. In many of his latest papers we see a

[2] See `https://eelcovisser.org/wiki/teaching/writing/`.

module Alias-Sdf-Projection
imports Alias-Sdf-Syntax[9.2.1]
exports
context-free syntax
Aliases "++" Aliases → Aliases {**right**}
"Al"(Grammar) → Aliases
"$\overline{\text{Al}}$"(Grammar) → Grammar
equations
The function 'Al' gives all alias declarations of a grammar, '$\overline{\text{Al}}$' the grammar without alias declarations.

[1] $al_1^* \mathbin{+\!\!+} al_2^* = al_1^*\ al_2^*$
[2] $\text{Al}(\text{aliases}\ al^*) = al^*$
[3] $\text{Al}(\mathcal{G}_1\ \mathcal{G}_2) = \text{Al}(\mathcal{G}_1) \mathbin{+\!\!+} \text{Al}(\mathcal{G}_2)$
[4] $\text{Al}(\mathcal{G}) =$ **otherwise**
[5] $\overline{\text{Al}}(\text{aliases}\ al^*) = \emptyset$
[6] $\overline{\text{Al}}(\mathcal{G}_1\ \mathcal{G}_2) = \overline{\text{Al}}(\mathcal{G}_1)\ \overline{\text{Al}}(\mathcal{G}_2)$
[7] $\overline{\text{Al}}(\mathcal{G}) = \mathcal{G}$ **otherwise**

Figure 3 A module from Eelco's dissertation.



(a) Master's Thesis. **(b)** Dissertation.

Figure 4 Covers of Eelco's Master's Thesis and Dissertation.

fusion of programming notation and logic notation. A case in point, is the use of syntax highlighting in logical formulae. I see this as an expression of his desire to still keep a link with the programming domain *per se*. He never returned to the literate programming style enabled by ToLaTeX.

5 Epilogue

In the Netherlands we used to have the tradition (these days abandoned at some universities) that a dissertation should be accompanied by a number of "propositions" that summarized the results of the dissertation. The last proposition was supposed to be less serious. I cannot think of a better ending than Eelco's last proposition (translated from Dutch):

> *The use of "hi" when saying goodbye comes from the realization that every ending is also a beginning.*

References

1. J. A. Bergstra, J. Heering, and P. Klint, editors. *Algebraic Specification*. ACM Press/Addison-Wesley, 1989.
2. Luis Eduardo de Souza Amorim and Eelco Visser. Multi-purpose syntax definition with SDF3. In Frank S. de Boer and Antonio Cerone, editors, *Software Engineering and Formal Methods - 18th International Conference, SEFM 2020, Amsterdam, The Netherlands, September 14-18, 2020, Proceedings*, volume 12310 of *LNCS*, pages 1–23. Springer, 2020.
3. P. Klint. *A Study in String Processing Languages*, volume 205 of *Lecture Notes in Computer Science*. Springer-Verlag, 1985. doi:10.1007/3-540-16041-8.
4. Bas Luttik and Eelco Visser. Specification of rewriting strategies. In M. P. A. Sellink, editor, *2nd International Workshop on the Theory and Practice of Algebraic Specifications (ASF+SDF 1997)*, Electronic Workshops in Computing. Springer-Verlag, September 1997.
5. Arie van Deursen, Jan Heering, and Paul Klint, editors. *Language Prototyping. An Algebraic Specification Approach*, volume 5 of *AMAST Series in Computing*. World Scientific, 1997.
6. Eelco Visser. Syntax and static semantics of Eiffel. a case study in algebraic specification techniques, December 1992. URL: https://eelcovisser.org/publications/1992/Visser92.pdf.
7. Eelco Visser. Combinatory algebraic specification & compilation of list matching. Master's thesis, Department of Computer Science, University of Amsterdam, June 1993. URL: https://eelcovisser.org/publications/1993/Visser93.pdf.
8. Eelco Visser. *Syntax Definition for Language Prototyping*. PhD thesis, University of Amsterdam, September 1997. URL: https://eelcovisser.org/publications/1997/Visser97.pdf.
9. Eelco Visser. Stratego: A language for program transformation based on rewriting strategies. In Aart Middeldorp, editor, *Rewriting Techniques and Applications, 12th International Conference, RTA 2001, Utrecht, The Netherlands, May 22-24, 2001, Proceedings*, volume 2051 of *Lecture Notes in Computer Science*, pages 357–362. Springer, 2001.
10. Eelco Visser and Mark G. J. van den Brand. From Box to TEX: An algebraic approach to the generation of documentation tools. Technical Report P9420, Programming Research Group, University of Amsterdam, July 1994. URL: http://ivi.fnwi.uva.nl/tcs/pub/reports/1994/P9420.ps.Z.