

Essence and motivation of Context-Oriented Programming

Ivo Willemsen
Open Universiteit
Maastricht, The Netherlands
ivo.willemsen@outlook.com

ABSTRACT

The last decade, use cases have emerged that emphasise the need to cater for different behaviour depending on situation and context changes. Examples are: Pervasive systems [8] and highly personalised business applications. Conventional programming languages offer constructs to implement context-dependent behavior, like conditional branches using if/switch statements, but they often result in cluttered code and uses of those constructs seriously damage the modularity of the applications. In the early 2000s, a new programming paradigm emerged, called Context-Oriented Programming which targeted to mitigate the aforementioned problems by incorporating context as part of the programming language, like variables, classes, and functions constitute the constructs of many contemporary languages.

This paper presents an introduction to Context-Oriented Programming, focussing on what Context-Oriented Programming is and explaining the *raison d'être* of its usage. As additional reading, examples of Context-Oriented Programming languages are given and some other aspects of these languages are elaborated on.

Keywords

Context-Oriented Programming, context-aware systems, behavioural variations

1. INTRODUCTION

Every intrinsically complex application exhibits behavior that depends on its context of use. Here, the meaning of context is broad and can range from obvious concepts such as location, time of day, or temperature over more technical properties like connectivity, bandwidth, battery level, or energy consumption to a user's subscriptions, preferences, or personalization in general. Besides these examples of context that are often associated with the domain of ambient computing, the computational context of the program itself, for example its control flow or the sets or versions of libraries used, can be an important source of information for affecting the behavior of parts of the system. Even though context is a central notion in a wide range of application domains, there is no direct support of context-dependent behavior from traditional programming languages and environments. Here, the expression of variations requires developers to repeatedly state conditional dependencies, resulting in scattered and tangled code. This phenomenon, also known as crosscutting concerns, and some of the

In order to include context-dependent behaviour in programs using most modern languages, one option available is to use the Strategy Design Pattern [7] to abstract the context-dependent behaviour into separate classes and decide at runtime-level which context-dependent behaviour (strategy) to use. Even worse would be the usage of conditional statements to find out the context in which a certain program is running, and as a result, not adhering to one of the concepts of Object-Oriented Programming: To avoid conditional statements to determine polymorphic behaviour. Both options are suboptimal as they result in cluttered code that is difficult to reuse and to understand and makes maintenance of the code a very cumbersome activity.

Context-oriented programming can overcome these problems by by dedicated language constructs for crosscutting concerns

The context-oriented programming (COP) [6, 4] approach supports the modularization of crosscutting concerns [11] and their control at runtime. In particular, COP focuses on a specific type of crosscutting concerns,

Context-oriented programming [8] (COP) addresses the development of systems, whose behavior varies depending on their context of use. In most cases, a behavioral variation is not implemented by a single object; instead, it is distributed over a team of collaborating objects. Such distributed functionality is denoted as crosscutting concern [10]. The modularization and composition of crosscutting concerns requires additional language abstractions beyond objectoriented programming. COP introduces layers, an encapsulation mechanism for crosscutting behavioral variations. A layer can be dynamically activated and composed with other layers, allowing fine-grained control of an application's runtime behavior

example battery how it affects different parts of an application, wifi, display

The steady convergence towards systems that are aware and reactive to their execution environment brings new functional and technical challenges that were a non-issue upon the time traditional desktop and server systems dominated the computing platform spectrum. Thanks to the real-time availability of information coming from their physical and logical environment, context-aware systems have the potential to adapt swiftly to changing running conditions and

Citations to articles [1, 3, 2, 4], conference proceedings [3] or books [6, 5] listed in the Bibliography section of your article will occur throughout the text of your article. You should use BibTeX to automatically produce this bibliography; you simply need to insert one of several citation commands with a key of the item cited in the proper location in the `.tex` file [5]. The key is a short reference you invent to uniquely

Table 1: Frequency of Special Characters

Non-English or Math	Frequency	Comments
\emptyset	1 in 1,000	For Swedish names
π	1 in 5	Common in math
\$	4 in 5	Used in business
Ψ_1^2	1 in 40,000	Unexplained usage

identify each work; in this sample document, the key is the first author’s surname and a word from the title. This identifying key is included with each item in the `.bib` file for your article.

The details of the construction of the `.bib` file are beyond the scope of this sample document, but more information can be found in the *Author’s Guide*, and exhaustive details in the *LaTeX User’s Guide*[5].

This article shows only the plainest form of the citation command, using `\cite`. This is what is stipulated in the SIGS style specifications. No other citation format is endorsed.

2.4 Tables

Because tables cannot be split across pages, the best placement for them is typically the top of the page nearest their initial cite. To ensure this proper “floating” placement of tables, use the environment `table` to enclose the table’s contents and the table caption. The contents of the table itself must go in the `tabular` environment, to be aligned properly in rows and columns, with the desired horizontal and vertical rules. Again, detailed instructions on `tabular` material is found in the *LaTeX User’s Guide*.

Immediately following this sentence is the point at which Table 1 is included in the input file; compare the placement of the table here with the table in the printed dvi output of this document.

To set a wider table, which takes up the whole width of the page’s live area, use the environment `table*` to enclose the table’s contents and the table caption. As with a single-column table, this wide table will “float” to a location deemed more desirable. Immediately following this sentence is the point at which Table 2 is included in the input file; again, it is instructive to compare the placement of the table here with the table in the printed dvi output of this document.

2.5 Figures

Like tables, figures cannot be split across pages; the best placement for them is typically the top or the bottom of the page nearest their initial cite. To ensure this proper “floating” placement of figures, use the environment `figure` to enclose the figure and its caption.

As was the case with tables, you may want a figure that spans two columns. To do this, and still to ensure proper “floating” placement of tables, use the environment `figure*` to enclose the figure and its caption.

2.6 Theorem-like Constructs



Figure 1: A sample black and white graphic (.png format).

Other common constructs that may occur in your article are the forms for logical constructs like theorems, axioms, corollaries and proofs. There are two forms, one produced by the command `\newtheorem` and the other by the command `\newdef`; perhaps the clearest and easiest way to distinguish them is to compare the two in the output of this sample document:

This uses the `theorem` environment, created by the `\newtheorem` command:

THEOREM 1. *Let f be continuous on $[a, b]$. If G is an antiderivative for f on $[a, b]$, then*

$$\int_a^b f(t)dt = G(b) - G(a).$$

The other uses the `definition` environment, created by the `\newdef` command:

Definition 1. If z is irrational, then by e^z we mean the unique number which has logarithm z :

$$\log e^z = z$$



Figure 2: A sample black and white graphic (.png format).

Two lists of constructs that use one of these forms is given in the *Author’s Guidelines*.

and don’t forget to end the environment with `figure*`, not `figure`!

There is one other similar construct environment, which is already set up for you; i.e. you must *not* use a `\newdef` command to create it: the `proof` environment. Here is an example of its use:

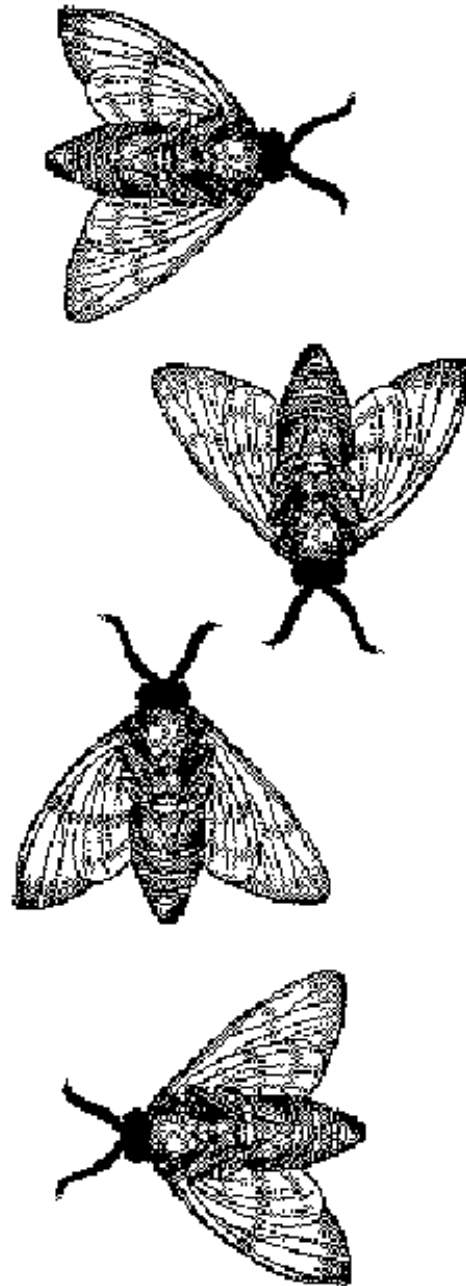


Figure 3: A sample black and white graphic (.png format) that needs to span two columns of text.

Table 2: Some Typical Commands

Command	A Number	Comments
<code>\alignauthor</code>	100	Author alignment
<code>\numberofauthors</code>	200	Author enumeration
<code>\table</code>	300	For tables
<code>\table*</code>	400	For wider tables

PROOF. Suppose on the contrary there exists a real number L such that

$$\lim_{x \rightarrow \infty} \frac{f(x)}{g(x)} = L.$$

Then

$$l = \lim_{x \rightarrow c} f(x) = \lim_{x \rightarrow c} \left[g(x) \cdot \frac{f(x)}{g(x)} \right] = \lim_{x \rightarrow c} g(x) \cdot \lim_{x \rightarrow c} \frac{f(x)}{g(x)} = 0 \cdot L = 0,$$

which contradicts our assumption that $l \neq 0$. \square

Complete rules about using these environments and using the two different creation commands are in the *Author's Guide*; please consult it for more detailed instructions. If you need to use another construct, not listed therein, which you want to have the same formatting as the Theorem or the Definition[6] shown above, use the `\newtheorem` or the `\newdef` command, respectively, to create it.

A Caveat for the T_EX Expert

Because you have just been given permission to use the `\newdef` command to create a new form, you might think you can use T_EX's `\def` to create a new command: *Please refrain from doing this!* Remember that your L^AT_EX source code is primarily intended to create camera-ready copy, but may be converted to other forms – e.g. HTML. If you inadvertently omit some or all of the `\defs` recompilation will be, to say the least, problematic.

3. CONCLUSIONS

Bypassing the need of scattering context-dependent behaviour throughout a program is one of the motivations of Context-Oriented Programming. In most modern programming languages,

This paragraph will end the body of this sample document. Remember that you might still have Acknowledgments or Appendices; brief samples of these follow. There is still the Bibliography to deal with; and we will make a disclaimer about that here: with the exception of the reference to the L^AT_EX book, the citations in this paper are to articles which have nothing to do with the present subject and are used as examples only.

4. ACKNOWLEDGMENTS

This section is optional; it is a location for you to acknowledge grants, funding, editing assistance and what have you. In the present case, for example, the authors would like to thank Gerald Murray of ACM for his help in codifying this *Author's Guide* and the `.cls` and `.tex` files that it describes.

5. REFERENCES

- [1] M. Bowman, S. K. Debray, and L. L. Peterson. Reasoning about naming systems. *ACM Trans. Program. Lang. Syst.*, 15(5):795–825, November 1993.
- [2] J. Braams. Babel, a multilingual style-option system for use with latex's standard document styles. *TUGboat*, 12(2):291–301, June 1991.
- [3] M. Clark. Post congress tristesse. In *TeX90 Conference Proceedings*, pages 84–89. TeX Users Group, March 1991.
- [4] M. Herlihy. A methodology for implementing highly concurrent data objects. *ACM Trans. Program. Lang. Syst.*, 15(5):745–770, November 1993.
- [5] L. Lamport. *LaTeX User's Guide and Document Reference Manual*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1986.
- [6] S. Salas and E. Hille. *Calculus: One and Several Variable*. John Wiley and Sons, New York, 1978.
- [7] Wikipedia. Strategy pattern — wikipedia, the free encyclopedia, 2017.
- [8] Wikipedia. Ubiquitous computing — wikipedia, the free encyclopedia, 2017.