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 [10] 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 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'etre* of its usage. As additonal 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

Many applications present behaviour that is determined by the context in which it is being used. Examples of different contexts are: Battery level, GPS location, available connectivity protocols (Wifi/3G/4G), speed of the network, user's preferences, etcetera.

In order to include context-dependent behaviour in applications using most modern programming languages, one option available is to use the Strategy Design Pattern [9] 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 conditonal statements to determine polymorphic behaviour. Both options are suboptimal as they result in cluttered code which is difficult to reuse and to understand and makes maintenance of the code a very cumbersome activity.

For example, battery level of a tablet or a smartphone on which an application (Operating System in this case) runs is a context, which probably impacts many behavioural aspects of that applications. It will not only have affect on the brightness of the screen, but it might also influece the way the Operating System prioritises running threads, and even result in the preventive hibernation of the system.

So, behavioural variation is not implemented by a sole object, rather it is spread over a group of cooperating objects. It is called a crosscutting concern [4] and this is a functionality that is dispersed over several cooperating objects. Plain old Object-Oriented Programming languages don't have constructs that allow for modularization and composition of crosscutting concerns, they lack first-class constructs. A first-class construct [6] is a construct which is an element of a language, like a class or a method in Java.

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 applicationâĂŹs runtime behavior

to cross cutting

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

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

Context-oriented programming [8] (COP) addresses the development of systems, whose behavior varies depending on their context of use. 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 match user needs and expectations more tightly than traditional systems.

Context-aware systems adapt their execution behaviour dynamically to particular situations based on contextual information discovered from their surrounding execution environment, such as weather conditions, localisation information, remaining battery power, user actions, system state, and so on

case definition to be used throughout the paper what is a context ? what are first-class values

The *proceedings* are the records of a conference.

ACM seeks to give these conference by-products a uniform, high-quality appearance. To do this, ACM has some rigid requirements for the format of the proceedings documents: there is a specified format (balanced double columns), a specified set of fonts (Arial or Helvetica and Times Roman) in certain specified sizes (for instance, 9 point for body copy), a specified live area (18 × 23.5 cm [7" × 9.25"]) centered on the page, specified size of margins (1.9 cm [0.75"]) top, (2.54 cm [1"]) bottom and (1.9 cm [.75"]) left and right; specified column width (8.45 cm [3.33"]) and gutter size (.83 cm [.33"]).

The good news is, with only a handful of manual settings, the LaTeX document class file handles all of this for you.

The remainder of this document is concerned with showing, in the context of an "actual" document, the LaTeX commands specifically available for denoting the structure of a proceedings paper, rather than with giving rigorous descriptions or explanations of such commands.

2. THE *BODY* OF THE PAPER

Typically, the body of a paper is organized into a hierarchical structure, with numbered or unnumbered headings for sections, subsections, sub-subsections, and even smaller sections. The command `\section` that precedes this paragraph is part of such a hierarchy. LaTeX handles the numbering and placement of these headings for you, when you use the appropriate heading commands around the titles of the headings. If you want a sub-subsection or smaller part to be unnumbered in your output, simply append an asterisk to the command name. Examples of both numbered and unnumbered headings will appear throughout the balance of this sample document.

Because the entire article is contained in the **document** environment, you can indicate the start of a new paragraph with a blank line in your input file; that is why this sentence forms a separate paragraph.

2.1 Type Changes and *Special* Characters

We have already seen several typeface changes in this sample. You can indicate italicized words or phrases in your text with the command `\textit`; emboldening with the command `\textbf` and typewriter-style (for instance, for computer code) with `\texttt`. But remember, you do not have to indicate typestyle changes when such changes are part of the *structural* elements of your article; for instance, the heading of this subsection will be in a sans serif typeface, but that is handled by the document class file. Take care with the use of the curly braces in typeface changes; they mark the beginning and end of the text that is to be in the different typeface.

You can use whatever symbols, accented characters, or non-English characters you need anywhere in your document; you can find a complete list of what is available in the *LaTeX User's Guide*[7].

2.2 Math Equations

You may want to display math equations in three distinct styles: inline, numbered or non-numbered display. Each of fksld;f k;lds kf;ldsk f;ldsk f;ldsk f;ldsk f;ldskf;ldskf ;ldskf ;lsk fd the three are discussed in the next sections.

2.2.1 Inline (In-text) Equations

A formula that appears in the running text is called an inline or in-text formula. It is produced by the **math** environment, which can be invoked with the usual `\begin. . .\end` construction or with the short form `$. . .$`. You can use any of the symbols and structures, from α to ω, available in LaTeX[7]; this section will simply show a few examples of in-text equations in context. Notice how this equation: $\lim_{n\to\infty} x = 0$, set here in in-line math style, looks slightly different when set in display style. (See next section).

2.2.2 Display Equations

A numbered display equation – one set off by vertical space from the text and centered horizontally – is produced by the **equation** environment. An unnumbered display equation is produced by the **displaymath** environment.

Again, in either environment, you can use any of the symbols and structures available in LaTeX; this section will just give a couple of examples of display equations in context. First, consider the equation, shown as an inline equation above:

$$\lim_{n\to\infty} x = 0 \tag{1}$$

Notice how it is formatted somewhat differently in the **displaymath** environment. Now, we'll enter an unnumbered equation:

$$\sum_{i=0}^{\infty} x + 1$$

and follow it with another numbered equation:

$$\sum_{i=0}^{\infty} x_i = \int_0^{\pi+2} f \tag{2}$$

just to demonstrate LaTeX's able handling of numbering.

2.3 Citations

Citations to articles [1, 3, 2, 5], conference proceedings [3] or books [8, 7] listed in the Bibliography section of your article

Table 1: Frequency of Special Characters

Non-English or Math	Frequency	Comments
Ø	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

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 [7]. The key is a short reference you invent to uniquely 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*[7].

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

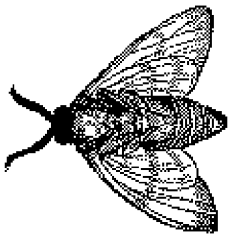

Figure 1: A sample black and white graphic (.png format).

"floating" placement of tables, use the environment **figure*** to enclose the figure and its caption.

2.6 Theorem-like Constructs

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!

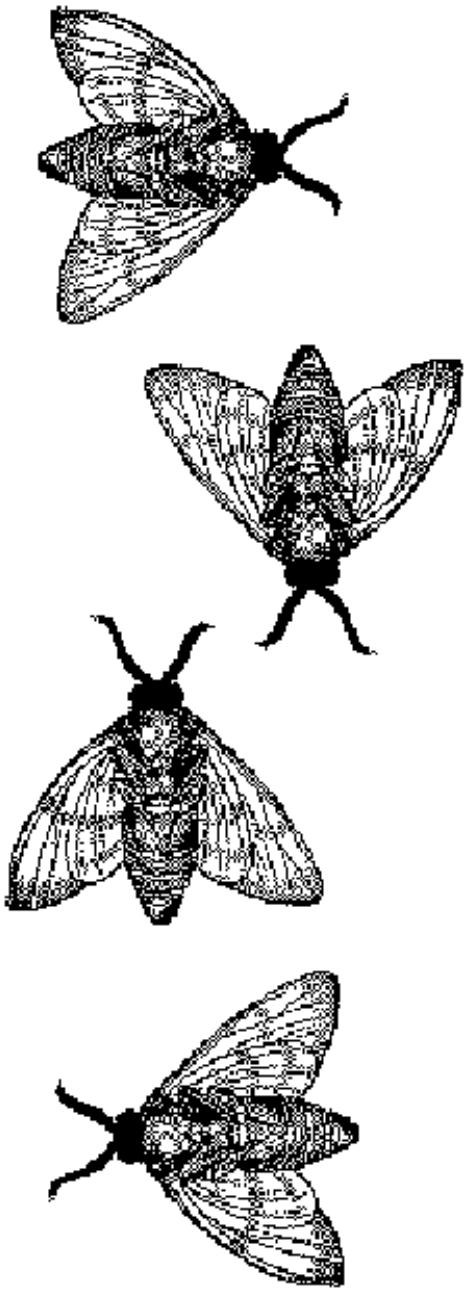

Figure 3: A sample black and white graphic (.png format) that needs to span two columns of text.

Table 2: Some Typical Commands

Command	A Number	Comments
`\alignauthor`	100	Author alignment
`\numberofauthors`	200	Author enumeration
`\table`	300	For tables
`\table*`	400	For wider tables

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 a example of its use:

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

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

Then

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

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

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[8] shown above, use the `\newtheorem` or the `\newdef` command, respectively, to create it.

A *Caveat* for the TEX Expert

Because you have just been given permission to use the `\newdef` command to create a new form, you might think you can use TEX's `\def` to create a new command: *Please refrain from doing this!* Remember that your LATEX 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 `\def`s 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 LATEX 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] A. M. Gregor Kiczales, John Lamping. Aspect-oriented programming. In *11th European Conference on object-Oriented Programming, volume 1241*, pages 220–242. g, 1997.

[5] M. Herlihy. A methodology for implementing highly concurrent data objects. *ACM Trans. Program. Lang. Syst.*, 15(5):745–770, November 1993.

[6] R. Keays and A. Rakotonirainy. Context-oriented programming. In *Proceedings of the 3rd ACM International Workshop on Data Engineering for Wireless and Mobile Access*, MobiDe '03, pages 9–16, New York, NY, USA, 2003. ACM.

[7] L. Lamport. *LaTeX User's Guide and Document Reference Manual*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1986.

[8] S. Salas and E. Hille. *Calculus: One and Several Variable*. John Wiley and Sons, New York, 1978.

[9] Wikipedia. Strategy pattern — wikipedia, the free encyclopedia, 2017.

[10] Wikipedia. Ubiquitous computing — wikipedia, the free encyclopedia, 2017.