The design and implementation of a userfriendly object-oriented markup language

Titouan Vervack
Supervisor: Prof. dr. ir. Marko van Dooren
Counsellor: Dr. ir. Benoit Desouter

***Abstract*—Neio is a markup language that is easy to read, write and learn, and that offers a modern, object-oriented programming model. This model allows for full customisation and control of the document. An object-oriented language was chosen because it can easily represent the structure of a document. The design of Neio was based on the results of a state-of-the-art analysis on document creation. To prototype the language, a compiler was developed and several libraries were implemented to demonstrate its flexibility. As a demonstration of its power and usability, the thesis itself was written in Neio.**

***Index Terms*—Markup, Object-oriented, LaTeX, Markdown**

I. Introduction

What You See Is What You Get (WYSIWYG) editors, such as Word and Pages, are the best-known solution for document creation. They immediately show the final document and help the user by providing a GUI for all of the features. This allows them to be used by anyone, even people without technical expertise.

However, these solutions do not offer a good solution to manipulate documents, outside of the GUI. The features offered in the used programming model are not sufficient to allow for full control of the document. The freedom that comes with a GUI also often leads to inconsistencies in documents.

Markup languages put a greater emphasis on the structure and/or customisation of the document. Markdown [1] for example uses a small fixed syntax [2] for the most common elements used in a document. This makes it very easy to read and write Markdown documents. However, Markdown lacks customisability.

LaTeX [3] on the other hand, offers full customisation using the Turing complete programming model of TeX. This programming model is complex and harder to use than modern programming models and also requires a certain technical proficiency to be used.

As such, we developed Neio. It had to be as easy to read, write and learn as Markdown but as powerful as LaTeX.

II. The design of Neio

We decided to use two syntaxes in Neio. The first is the text mode syntax. It is used to write the actual documents. As such it has to be as simple as possible. In the state-of-the-art analysis, Markdown was found to be very easy to read, write and learn, thus we based text mode on Markdown.

Documents written in text mode are called Neio documents, an example is shown in Listing 1.

Listing 1: A simple Neio document.

```
[Document]

# Chapter 1
This is the first paragraph.
```

The second syntax is called code mode. It is used to provide the objects and methods that are used in a text document. Code mode is very similar to the Java syntax. Documents written in code mode are called code files and look almost the same as Java class files.

An important difference compared to Java, is that the method identifiers can contain symbols. The symbols that can be used are `#`, `-`, `*`, `_`, `$`, `^`, `=`, `|`, `` ` ``. There are also reserved methods for the `newline` and for text.

Java was chosen because it is one of the most popular programming languages [4] of this age. This provides a big advantage because every Java library can be used in Neio. It also creates a large user base that can already read and understand code mode.

Lastly, an important difference with Markdown is that the semantics of text mode are not hard-coded in the language. Instead they can be defined in the code files. For example, for slide shows, # can be redefined to create a new slide instead of a chapter.

A text document is actually a chain of method calls that create an object model of the document. This object model can then be visualised later on.

This chain is called the `call chain`. The call chain for the example in Listing 1, is shown in Listing 2.

It is also possible to enter code mode in text mode by opening a pair of curly braces. This is shown in Listing 4.

Listing 2: The call chain of the example in Listing 1.

```
new Document()
    .newline()
    .newline()
    .#("Chapter 1")
    .newline()
    .text("This is the first paragraph.");
```

III. Programming in Neio

A number of libraries illustrate the programming capabilities of Neio. Listing 3 shows how the `Table` class allows the

writer to create a table using the `|`, `-`, and `newline` methods. Figure 1 shows the rendered version.

Listing 3: A table in text mode.

```
|   Student club   | Rounds | Seconds/Round |
-----------------------------------------
| HILOK           |  1030  |      42       |
| VTK             |  1028  |      42       |
| SK              |   647  |      67       |
| Zeus WPI        |   567  |      76       |
| VBK             |   344  |     126       |
```

Student club	Rounds	Seconds/Round
HILOK	1030	42
VTK	1028	42
SK	647	67
Zeus WPI	567	76
VBK	344	126

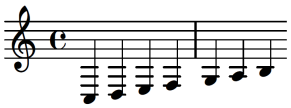
Fig. 1: The rendered version of the example in Listing 3.

The second example (Figure 4) shows how to create sheet music and how to reverse the notes in the `Score`.

Listing 4: A Neio document that creates sheet music.

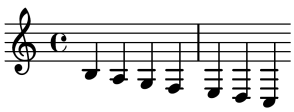
```
[Document]
{
    Score s = new Score().c().d().e().f().g().a().
        b();
    return s;
}
The score is read as {s.print()}.

{
    Score reversed = ss.reverse();
    return reversed;
}
The reversed score is read as {reversed.print()}.
```



The score is read as c, d, e, f, g, a, b.

Fig. 2: The rendered version of the score in Listing 4.



The reversed score is read as b, a, g, f, e, d, c.

Fig. 3: The rendered version of the reversed score in Listing 4.

In Listing 4, we also see that code mode can be entered in a sentence. This allows us to easily execute simple expressions such as as the `s.print()` shown in the example.

IV. Compiler

The process a Neio document goes through from Neio source code to LaTeX code, is shown in Figure 5.

The Neio compiler is created using Chameleon [5], [6] and Jnome [7]. Chameleon is a framework for defining abstract abstract syntax trees (AAST) of software languages. It enables the reuse of generic language constructs and the construction of language-independent development tools. It defines language-independent objects (`Element`, `CrossReference`, ...) as well as paradigm-specific ones (`Type`, `Statement`, `Expression`, ... for Object-Oriented Programming).

Jnome is a Chameleon module for Java 7 that provides language specific objects. These libraries allowed us to reuse most language semantics, and to obtain IDE support with only a few lines of code. The architecture is shown in Figure 4.

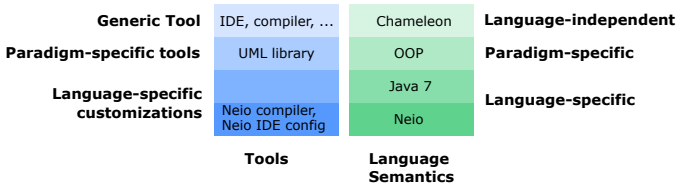


Fig. 4: The Chameleon architecture.

The compiler also copies resources, such as images and configuration files to the output folder.

The final transformation to LaTeX, is done by the Neio standard library. To generate the LaTeX code, the `toTex` method is called on the root of the object model which recursively calls it on all the other objects. This method can also call other programs. To produce the scores in Figure 2 and 3, it calls LilyPond [8], which generates a PDF that is then imported in LaTeX.

References

[1] "Markdown homepage," https://daringfireball.net/projects/markdown/, accessed: 07-05-2016.

[2] "Markdown syntax," https://daringfireball.net/projects/markdown/syntax, accessed: 08-05-2016.

[3] "LaTeX introduction," https://latex-project.org/intro.html, accessed: 20-05-2016.

[4] N. Diakopoulos and S. Cass, "The top programming languages 2015 according to IEEE spectrum," http://spectrum.ieee.org/static/interactive-the-top-programming-languages-2015, accessed: 30-05-2016.

[5] "The Chameleon framework," https://github.com/markovandooren/chameleon, accessed: 18-05-2016.

[6] M. van Dooren, E. Steegmans, and W. Joosen, "An object-oriented framework for aspect-oriented languages," in *Proceedings of the 11th Annual International Conference on Aspect-oriented Software Development*, ser. AOSD '12. New York, NY, USA: ACM, 2012, pp. 215–226. [Online]. Available: http://doi.acm.org/10.1145/2162049.2162075

[7] "The Jnome framework," https://github.com/markovandooren/jnome, accessed: 18-05-2016.

[8] "The LilyPond homepage," http://lilypond.org/, accessed: 30-05-2016.

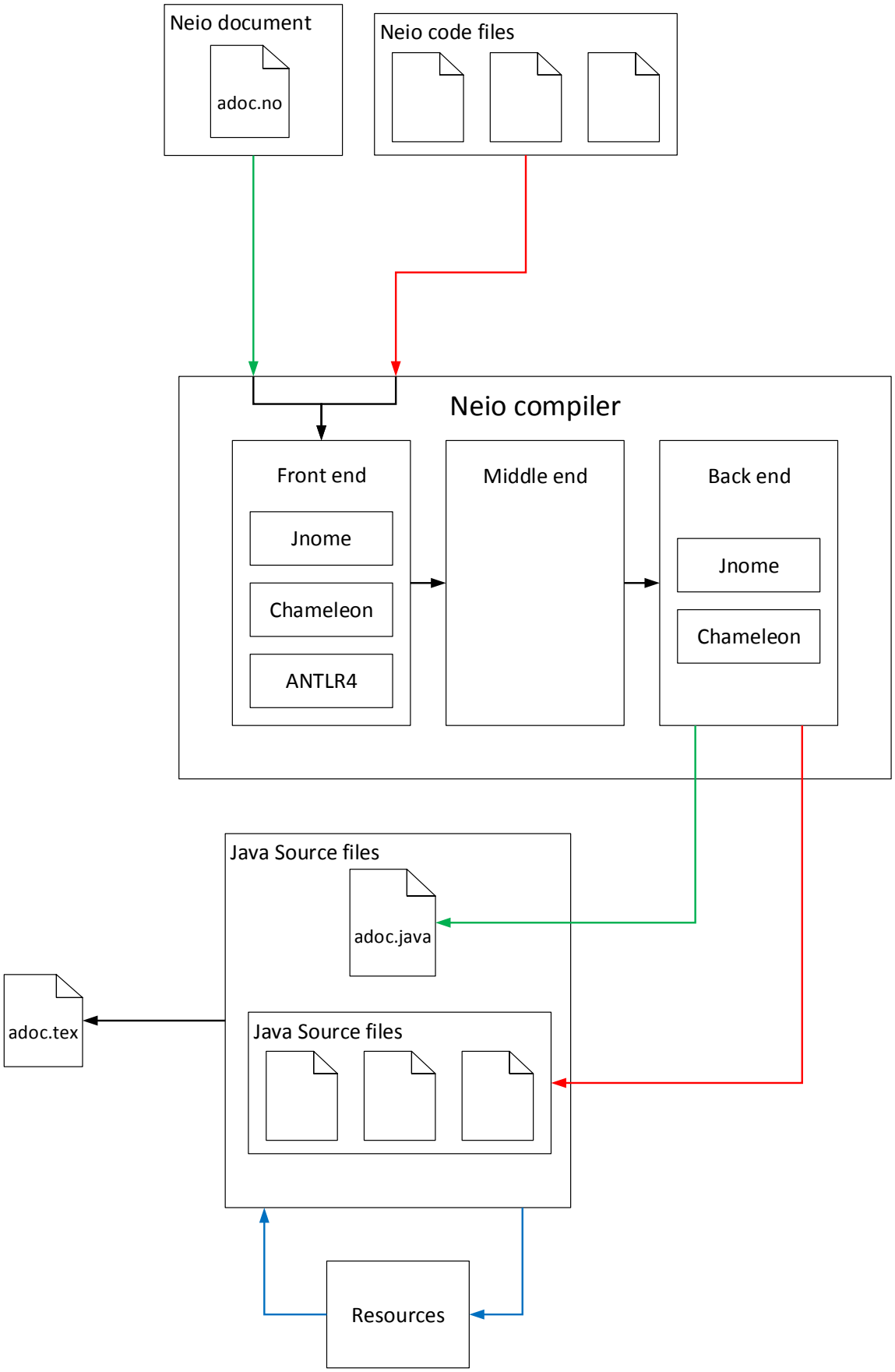


Fig. 5: The process of compiling Neio source code to LaTeX.