An Editor for Pandoc Types

Massimiliano Farinella

An editor for Pandoc abstract syntax tree (AST) lets you visually modify any document readable by Pandoc and export it into any output format it writes. Configurations make the editor customizable to different workflows.

1. Project MEO in maintenance mode

The software for the MEO[1] project is going to maintenance mode, because all of its 50 volumes have been published.

The software I'm introducing here – a visual editor for Pandoc's internal document model – is part of an effort to generalize and reuse some of the tools and the expertise accumulated for MEO.

Here's a recap of the features of the software for the MEO project:

- a software to edit and typeset rich texts, running on a single server, organized in a bunch of services living inside Docker containers;
- the texts are XHTML encoded and edited with a browser connecting to the server;
- the documents are stored in a MySQL database;
- typesetting is done mostly with ConTEXt (43 volumes), but also with InDesign (7).

2. Long-term conservation of documents

Since the texts amounted to about 30,000 printed pages, their manageability – once digitized – was the main aspect that guided the choice of the documents' format.

The texts were very varied, so something like Markdown was not enough.

XML is a good choice for long-term conservation of texts that have a fair amount of complexity. But saying 'XML' is not saying anything, because you must choose a XML grammar: some existing ones, like XML-TEI, are overkill; many are missing certain elements you'd really need; most of them lack good, free tools or require skills that you can't find easily. A home-tailored grammar would fit your needs, but *only you* would know it and carry all the burden of the tools and documentation.

[1] articles.contextgarden.net/journal/2017/98-102.pdf

We chose XHTML: it's HTML, and HTML is everywhere, with some good, free, visual and customizable tools available. But it's also XML, so you can use XML-related tools, e.g. typesetting it directly with ConTEXt. It has elements for structured texts, though not page-oriented. It lets you specialize elements like <p> or adding a class; that way they can work as surrogates for custom elements of a tailored XML grammar.

3. Converting documents: Pandoc

Since formats and tools change over time, the conservation of documents involves their ability to adapt to new tools: it means that you can convert them into a new format without losing much of the information they carry.

Pandoc has become a de facto standard in the conversion of texts between different formats. It's a collection of *readers*, one for every input format it knows about, that convert input formats into Pandoc's AST (abstract syntax tree, its document model), and *writers*, which do the opposite, i.e. transform the AST into the output format.

Both transformations often cause some loss of information. Pandoc's goal is to preserve as much as possible of the document's structure and contents, not necessarily its graphical layout.

We used it for volumes to be typeset with InDesign, because the texts needed to be converted from HTML to DOCX before getting imported in InDesign.

Interestingly for ConTEXt people, Pandoc now incorporates a Lua engine that lets you customize its conversions writing Lua code.

4. Editing documents in the browser: Prosemirror

Prosemirror is the best kit around to build a visual editor of rich text in a browser.

In the MEO project, we used a HTML editor (CKEditor), but we did not need all the HTML tags. So we spent some time getting around the quirks and the unwanted HTML that the editor introduced now and then.

We had to edit the HTML source directly to solve the trickiest problems. Fortunately, the editor provided an alternative view to directly edit the HTML source with the help of syntax highlighting, though it is error prone and not for everyone.

Prosemirror uses HTML only to render your document in the browser and generates descriptions of the changes to be done to the document accordingly to the user's keystrokes and mouse actions.

When such changes are applied, the modified document will be rendered again – say "transformed into HTML" – in the browser. So the data structure of your document can be of any kind, not necessarily HTML, and you have full control of the textual elements that can or can't enter your document.

5. Back to MEO and its legacy

I was asked if the MEO software could be used for the production of other books (someone did it against my admonitions, though with some success).

The first answer to that question was “no”, because the software is tailored to solve a very particular problem: dozens of books with similar features, the same layout, organized in nearly isolated volumes identifiable by their number, from 1 to 50. This software broadly economizes on regular patterns to reduce the complexity of a more general model.

Then I started experimenting with the coding of new editors for different kinds of documents. They were all based on Prosemirror, and for each one I set about providing functions to transform their document model into Pandoc AST, so that I could transform them to any of the formats Pandoc supports.

Eventually, I asked myself: why not writing – and maintaining – only one editor for Pandoc AST instead of many similar custom editors? And, as a consequence, how to provide customizations of the editor interface for any different document model?

6. An editor for Pandoc types

So here it is: a visual editor for Pandoc AST that can be adapted to different workflows through configuration files.

It can import or output every format supported by Pandoc. If that is not enough, you can write custom format readers and writers[2], filters[3], templates[4], and reference docs[5] to better fit your model, add them to its configuration and have them directly available from the program interface.

It’s not exactly an editor for everyone, because you need to know a little of the internal document model of Pandoc, but it’s not that hard. It may be a bit harder understanding how the Pandoc AST is mapped onto your document model, unless you are the one who writes the custom readers, writers, etc.

Maybe you won’t get an editor perfectly fit for your model, but it pretends to be like that as much as possible, through the customizations available in the configuration files – essentially a main JSON file surrounded by CSS (for GUI customization) and Lua files, since Pandoc facilitates the writing of custom readers, writers, and filters in Lua.

Adding funtionalities to custom workflows will mostly consist of external Lua scripts, while the main editor should stay the same.

[2] pandoc.org/MANUAL.html#custom-readers-and-writers
[3] pandoc.org/filters.html
[4] pandoc.org/MANUAL.html#templates
[5] pandoc.org/MANUAL.html#option--reference-doc

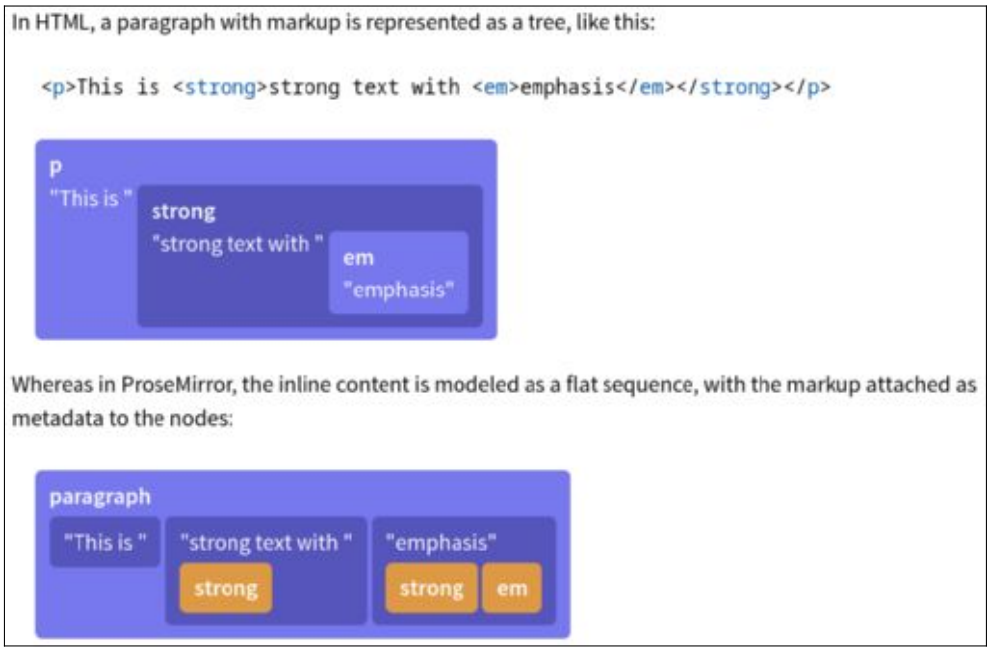

Fig. 1. HTML and Prosemirror document models

7. Pandoc vs. Prosemirror

Pandoc and Prosemirror have a different document model. At block level, they are similar:

- both have a root node whose children are blocks (`Blocks` in Pandoc, block `Nodes` in Prosemirror);
- both have blocks containing blocks, like a blockquote, that usually contains only paragraphs;
- at paragraph level, they differ: while Pandoc's `Inlines` can contain other `Inlines` in a tree-like fashion of arbitrary depth, just like HTML does – e.g. you may have a `Strong` emphasis inside an `Emph` emphasis, all inside a `Quoted` quotation, and so on –, the text nodes of a Prosemirror paragraph can't contain other nodes: the world is flat inside Prosemirror paragraphs;
- Prosemirror has `Marks` instead: they are like labels attached to portions of a paragraph, e.g. you may have a text that has both a strong (bold) and a simple (italic) emphasis, and so having two `Marks`.

Figure 1 is a snippet of the Prosemirror library guide, that shows the structure of the same HTML code in the HTML *document object model* (conceptually equivalent of Pandoc's one) and in a Prosemirror structure (`Marks` are the orange ones).

In Prosemirror, you don't have a predefined document model; instead you define a *schema* specifying, for each node, which nodes it may contain, and which `Marks` you can apply to it.

The `schema` for Pandoc documents follows the hierarchy of Pandoc types:

- there's a root `Node`, `Pandoc`;
- such `Node` can contain an optional `Meta` object and a sequence of block `Node`s;
- the block `Node`s, that contain `Inline`s in Pandoc, will contain text `Node`s, labelled with `Mark`s matching Pandoc `Inline`s.

This is approximately the Prosemirror `schema` for a document of Pandoc.

This is only the big picture, because it is actually more complex, in particular the translation between Pandoc `Inline`s and Prosemirror `Mark`s.

So, essentially, the editor:

- transforms Pandoc `Block`s and `Inline`s (and `Meta`) from their JSON representation into Prosemirror `Node`s and `Mark`s,
- lets the user manipulate the document, changing `Node`s and `Mark`s,
- saves the modified document transforming Prosemirror `Node`s and `Mark`s back into Pandoc `Block`s and `Inline`s (and `Meta`).

8. Conventions and extra features

The editor has interfaces for Pandoc conventions on custom styles[6], but it also adds some other conventions to support the inclusion of external documents, indices – registers in ConTEXt – and more than one kind of notes, since Pandoc has only footnotes.

Conventions don't require changes in Pandoc types specification[7]. They are a way to support new features simply using the flexibility of some of its `Block`s or `Inline`s, in particular those carrying the `Attr` data structure[8]).

Here are some examples:

- a custom character (inline) style is a `Span` with a `custom-style` attribute in its `Attr`; this convention is natively supported by Pandoc;
- a `Div` with a `custom-style` attribute in its `Attr` sets the style of the paragraph it includes; this is also natively supported;
- when a `Div` with a `note-type` attribute in its `Attr` is the only child of a `Note`, it sets the note's type (e.g. *endnote*, *margin note*); this is supported by the editor, but not by Pandoc, so you need custom writers or filters to use it;
- a `Div` with an `include-src` attribute in its `Attr` specifies the inclusion of an external document; a custom filter will replace the `Div` contents with the `Block`s of the included document.

[6] pandoc.org/MANUAL.html#custom-styles
[7] hackage.haskell.org/package/pandoc-types-1.23/docs/Text-Pandoc-Definition.html
[8] hackage.haskell.org/package/pandoc-types-1.23/docs/Text-Pandoc-Definition.html#t:Attr

9. Technical specs for developers

The editor is based on the Vite+Electron+Vue[9] template by Alex Kozack, though it's not kept in sync with it.

It's written in Typescript[10] and it uses Tiptap[11], Prosemirror[12], Vue[13], Electron[14] and Quasar[15] for its GUI.

The code base structure lets you integrate the editor in an online app, like the MEO one, but also package it as a standalone, Electron-based app.

For now you can deploy the standalone version only, for GNU/Linux (`deb` format) and MS Windows (a single, portable executable).

10. Dependencies

The editor depends on many packages.

All have liberal licences, at least they are free as in "free beer"; some are more easily replaceable than others because they are less widespread in the code base.

But upgrading them can bring in incompatibilities, since versions change at a fast pace in the JavaScript/TypeScript world.

I have no fixes for that. Behind their automatic, self-supporting appearance, software projects are more like living organisms, with real people struggling to keep these automatisms working.

Even worse, I have one more package to maintain: `prosemirror-tables-sections`[16].

Prosemirror's module for tables (`prosemirror-tables`) could not support Pandoc's most recent and richer table model; it was also without an official maintainer.

So I had to upgrade it to support table sections, lacking the skills of its original author, Marijn Haverbeke, who's also the author and maintainer of Prosemirror.

It seems to work fine for now, at least in my editor, but that's undoubtedly another dependency asking its toll.

[9] github.com/cawa-93/vite-electron-builder
[10] www.typescriptlang.org/
[11] tiptap.dev/
[12] prosemirror.net/
[13] vuejs.org/
[14] www.electronjs.org/
[15] quasar.dev/
[16] www.npmjs.com/package/@massifrg/prosemirror-tables-sections