The Project: MEO

Massimiliano Farinella

MEO stays for "Marx-Engels, Opere" ("Marx-Engels, Works") and it's the project, led by the *Istituto di studi sul capitalismo*[1] in collaboration with *Edizioni Lotta Comunista*[2] to complete, digitize and publish the works by Karl Marx and Friedrich Engels in Italian.
It's a volunteering project working on texts coming from different sources. Why talking about it here? Because ConTEXt is used to preview the texts and produce the PDFs to be printed.

1. The Texts, the People and the Tools

It's a huge material: 50 volumes of 600 and more pages each, with hundreds of footnotes and endnotes, a biographical index of names and bibliographies.
It's a varied material: texts, letters, articles, documents, manuscripts with many kinds of annotations, tables, mathematical formulas (a dozen volumes contain a significant amount of them), two kinds of footnotes (the authors' ones distinct from the editorial ones), juvenile poems and even sketches of theatre plots, quotes of text and poems, even in footnotes, parallel texts, sketches by the authors.
Also the sources are variable: more on this in the section "The Sources".
The main goal is to have them published as printed volumes. A second goal is to have a database of electronic texts that remain available and usable in the long term.
There are many volunteers willing to help, but nearly no one is trained to use a DTP software; almost everyone knows one of the versions of Microsoft Word or, less frequently, Open/Libre Office.
Those were the boundary conditions when we started developing a solution three years ago.
The solutions changed over time and are still changing, to adapt to the challenges that such a huge project poses.

1.1 MEOEDIT

We are using (and still developing) a software — MEOEDIT — to store, edit and typeset the texts.
It uses many open source softwares: Debian, Ubuntu, Docker, ConTEXt, CKEditor, Node.js, Mysql, PHP, Pandoc, Gitit.
It's a web-based, rich text editor of (X)HTML texts, stored in a Mysql database and converted to PDF with ConTEXt. That is its current shape, but it has not been that way from the start.
For example, before discovering ConTEXt — it happened by chance — we were experimenting with *PrinceXML*[3], an HTML/XML+CSS to PDF converter.

2. An XHTML to PDF Workflow of Texts Edited in a Browser

The format in which the texts are edited and stored is (X)HTML, so the typesetting follows an XML to PDF workflow.
(X)HTML was finally chosen to maintain, as much as possible, a "structured text" approach, separating structure from style and rendering.
Of course HTML is not the best choice for such an approach.
XML is better, but you need to define a grammar or choose an existing one like XML-TEI.
We wanted an easy, visual editor for users willing to help but frightened by tags and angular parentheses. It is difficult to find free, good, visual XML editors. *XMLMind XML Editor*[4] is one of the best, but it's not free nor open source. It is much easier to find HTML ones.
Finally we chose CKEditor: it's visual but it has a source code view that lets you inspect the "real" text: tags, plain text and no hidden elements.

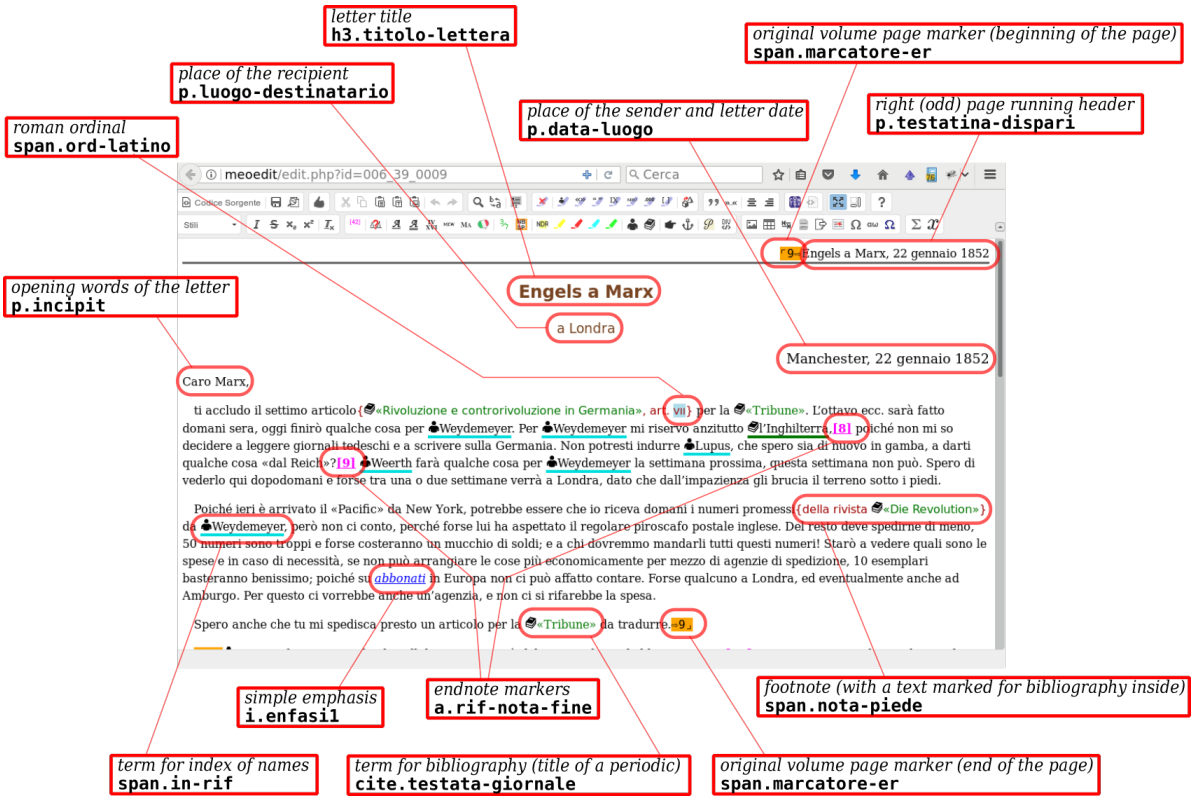

Figure 1: A letter in the editor and styles in `tag.class` notation.

Another important feature of CKEditor is that it lets you specify the tags allowed to mark your text; unspecified tags are flattened to plain text; that's something similar to the DTD for XML documents. That way your texts stay quite “clean” even when you copy and paste from different sources.
So the trade-off is to use HTML as if it were XML:

- use existing, consistent HTML elements where possible: <p> for paragraphs, <h1> ... <h6> for titles, and so on
- simulate missing elements with existing HTML elements: i.e. <a> for footnote or endnote markers
- use the `class` attribute to diversify the standard HTML elements or simulate new ones
- prohibit the `style` attribute: no anonymous, local specifications of styles; CKEditor's controls to choose font, size and color have been disabled and do not appear in the toolbar
- style the editor with an external CSS to make the structural elements look similar to what you would expect: i.e. italic for emphasis, bold for strong emphasis, an icon of a person before a name to be included in the biographical index, and so on
- warn users that the editor is intended as “What You See Is What You Mean” and not WYSIWYG; train them to expect some quirks from CKEditor, since this latter was not conceived primarily for XML, and to call for help from other users more comfortable with the source code view
- for developers, take advantage of jQuery and Cheerio.js, its server side counterpart, to work with text structures (easier than XSLT)

There is nothing new here. Many ideas are borrowed from other software and specifications, so that a newbie with prior knowledge can read the code without finding it so alien.

This grammar — specified through HTML elements and custom classes — is still arbitrary and not standard. But it is conceived to remain easily editable, understandable and convertible into other formats (DOCX, EPUB, XML-TEI).

3. The Sources

For the most part the texts came as printed volumes by the publisher Editori Riuniti, some others were available as PDFs and others are under translation, since they are still unpublished in Italian.
Printed volumes have been scanned and passed through OCR page by page, then uploaded as single page files to the site *meo-correzione.org*[5] — along with the PDF image of the original volume — for a distributed correction of OCR errors.
The downloaded, corrected texts were then converted into HTML files and loaded into MEOEDIT.
Volumes available as PDFs were converted first to DOCX, then to HTML with a custom converter based on Pandoc, then loaded into the program. The same goes for unpublished volumes, supplied as DOCX files.
Once inside MEOEDIT, the texts were given a first formatting and then converted to PDF for preview (initially with PrinceXML, then with ConTEXt); then they were read and checked against the original sources — "validated" — by pairs of people. The corrections were then reported in MEOEDIT.

4. Text Units

Since the editor lives in a web browser, you can't edit a whole volume at once, as you would do with a word processor: the browser would get slow and it's difficult working on a text that is not WYSIWYG nor organized in pages.
We overcame that problem by "slicing" every volume into atomical units of text, and then finding a way to assemble them later. We named them "Unità di testo" (UDT), which means "Text Units":

- they consist in a number of original printed pages ranging from one to some dozens; in rare cases, there are 70 pages long Text Units and they are the hardest to edit

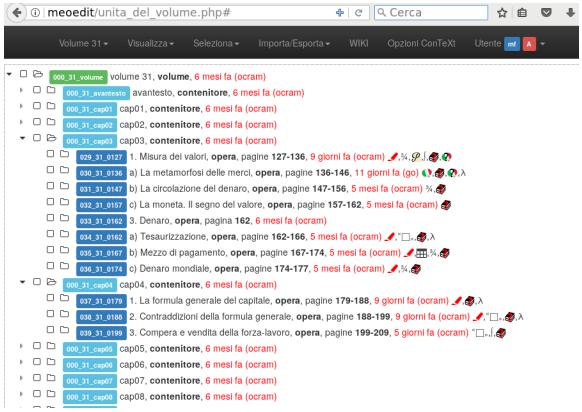

Figure 2: The tree structure of a volume. Container UDTs are in green or light blue

- they try to match the divisions of the original book: a single letter, an article, a chapter...
- in a minority of cases, the text has been split at arbitrary points, just to avoid UDTs of too many pages

4.1 Container Text Units

To assemble the Text Units (UDT) into volumes first we thought of a master file, but then we decided to reuse something we already had: Text Units.
Inside a Text Unit you can insert a reference to another one: think of "includes" in source code or `xi:include` for XML. They are coded as HTML links, `<a>` elements with a particular class.
When the software assembling Text Units finds a reference to another UDT, it replaces the reference with the UDT contents (actually the HTML contents of its `<body>` element). For example, this replacement happens when a Text Unit is converted to PDF by ConTEXt.
The Text Units containing references to other ones are called "container Text Units" ("UDT contenitore") and they can be organized to form the tree structure of a volume: a volume UDT, containing references to the chapter UDTs, containing references to smaller units.
The container Text Units are designed so that they start at a new page and the next one begins at a new page too.
This way you can get the ConTEXt preview of a Text Unit — representing a part of a book — and be sure that its typesetting won't influence the

text flow and page breaking of other sections. It’s also faster than previewing a whole volume, after a local change.

5. Docker

The software is installed on a server. At an earlier stage it was a Debian installation with an Apache server. In case of failure, we had to reinstall from scratch and remember all the needed libraries. In case of new software dependencies, we had to make sure they did not conflict with existing ones. Then we shaped MEOEDIT to work inside *docker*[6] containers, and we solved these problems:

- the need for an easy way to install from scratch, because of failure recovery or simply to have a machine for MEOEDIT development
- software libraries dependencies and conflicts: you can run different GNU/Linux distributions in distinct containers
- the possibility of building different services with different tools and languages

MEOEDIT is modularized into 7 Docker containers:

- mysql server
- apache server + the site (PHP + CKEditor)
- a wiki
- a service based on *Express.js*[7] that acts as a proxy between the site and the mysql database, aggregating data and returning them as JSON objects
- a redis cache (a RAM-resident key-value cache) used by the previous service, to speed up the responses
- a service that accepts XHTML and typesets it with ConTEXt, returning a PDF and some progress information as well
- a service to export the texts into other formats using Pandoc

Every container is built following a recipe written in a `Dockerfile`, a text file describing the operations needed to rebuild the (image of the) container. This enforces discipline, because every time you introduce a new dependency you need to specify it in the `Dockerfile`, with its configuration. When

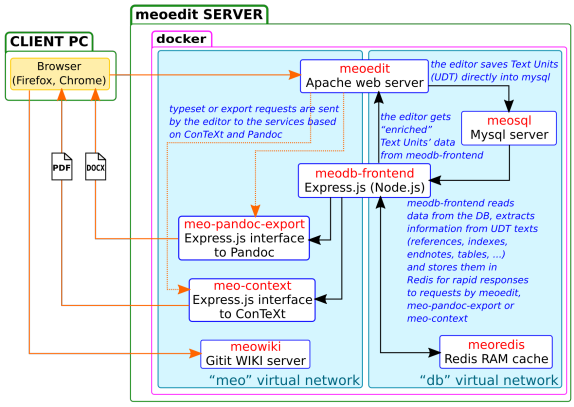

Figure 3: The container structure of MEOEDIT

the server must be replaced, you’ll have an automatic recipe to build an identical one.
Every container is a sort of isolated machine, with its own distribution of GNU/Linux, its versions of libraries and so on. For example, the context container is built on a Debian Jessie distribution, with the latest version of ConTEXt and the version of Node.js coming from its site instead of the older one from the official Debian repository; the wiki is built on Ubuntu Xenial and has *gitit*[8] installed.
The containers are configured and managed with *Docker Compose*[9], which takes the configuration from a single text file — `docker-compose.yml` — and pulls up the containers; if the container images do not exist yet — think of a new installation — the config file contains information for docker-compose to build them.

6. `meo-context`, the Container Containing ConTEXt

ConTEXt is run inside its own container, whose name is `meo-context`. In earlier versions of MEOEDIT, it ran on the same machine of the web server and was called by PHP code running inside Apache. If anything went wrong, the server could be blocked too. Now it is only `meo-context` that goes down, leaving the web server untouched. You loose the PDF preview until `meo-context` gets restarted or rebuilt, but you can continue editing the texts.

7. Further Developments

7.1 XHTML and ConTEXt

The MEO workflow is XML to PDF, where the XHTML should only contain information about the structure of the documents.
But there's no completely automated typesetting, and you always need to do manual adjustments.
They are easier and more natural in ConTEXt documents, but how to do them when your sources are XHTML? How to code them inside XHTML? How to inject ConTEXt macros inside XHTML? How to clean up your XHTML from those artifacts, that are intended only for a particular typesetting?
Hans once told me he uses processing instructions — <?...?> tags — in XML; in MEOEDIT we are using a combination of custom classes, attributes and CSS properties in the `style` attribute of an XHTML element.
That can be the subject of presentations to come at the next meetings.

7.2 ConTEXt and Docker

Using ConTEXt inside a Docker container can be also good to "scale up" ConTEXt: not the parallelization of a single job, but multiple distinct jobs that run in parallel using all the cores of a CPU on one or multiple PCs. There's room for further development and discussion.

Notes

[1] www.isc-studyofcapitalism.org
[2] www.edizionilottacomunista.com
[3] www.princexml.com
[4] www.xmlmind.com/xmleditor
[5] meo-correzione.org
[6] www.docker.com
[7] expressjs.com
[8] github.com/jgm/gitit
[9] docs.docker.com/compose