Transcript for Andrew Hardie's presentation at CL2021 (July 13 2021).

Extensibility as a focus for corpus analysis software - The CQPweb plugin framework

https://youtu.be/JerPDB_gYJE

Hello, and thanks for tuning in to my talk on the CQPweb plugin framework – which is at the focus of my current work to make EXTENSIBILITY a major feature.

CQPweb is the graphical interface for Corpus Workbench (CWB). What is CWB? A very well-known system for indexing and querying large corpora with multiple layers of annotation, under development since 1990s, first at IMS Stuttgart, later as an open project.

CQPweb was originally designed for me to use in my own teaching at Lancaster University back in 2008. It replicated the interface of the well known BNCweb system, but could be used for work with any corpus, not just the BNC.

But people outside Lancaster were gradually granted access to our server and it became widely used. And when CQPweb was adopted into the CWB project, from then on it was open source. Meaning anybody can install it – and many people have, so there are many servers out there on the web.

In principle, since it's open source, anybody can extend or modify CQPweb – to support user needs that the core system doesn't cover. In practice, to do that, you not only need fairly strong programming skills – you also need a pretty detailed understanding of CQPweb's data model, program flow and internal function libraries. That's a big ask.

So what is possible in theory is unsurprisingly rarely done in practice.

But what I can do, as the main developer, is make CQPweb extensible – that is, create a framework where others can add extra gizmos to CQPweb without having to know about the entire system architecture.

The main CQPweb program can never support the needs of every user. But an extensibility framework can let others support their own needs.

What are the kinds of user needs that I'm talking about?

The number 1 need is that users want to insert their own corpora into CQPweb. People have wanted this since approximately 5 minutes after CQPweb arrived on the scene.

But users also want to be able to make interfaces to corpus construction/annotation tools – or pipelines.

They want more options for formatting of textual downloads.

They want a broader range of analysis tools for concordances and corpora, especially statistical analyses and novel data visualisation.

I am trying to meet these needs via extensibility -- By adding a PLUGIN framework to CQPweb.

What’s a plugin? It’s a self-contained code object which performs a narrowly defined task within a bigger system.

There have always been many places in CQPweb where the program performs a modular job – one that doesn’t make reference to the rest of the system. This includes data download as text files, corpus installation, annotating textual data, and many more. Any of these modular jobs could be replaced *as a unit* by another unit that does an equivalent job.

CQPweb now allows people to write plugins to do particular modular jobs, and slot them in to the system.

Different named types of plugins are defined to address different user needs. ***Downloader*** plugins generate a text file download from the results of any query. They connect up to the “Download…” option in the concordance action menu.

On the “download” page, CQPweb’s default tool is a complex set of options at the bottom. But, there have always been shortcuts – buttons at the top for frequent-used-options. Now, those buttons are plugins! It’s fully configurable what buttons for specific download formats are added or removed at the top here.

Behind the scenes, the plugins are managed by the system administrator, who has an interface showing what plugins are available, and controlling which users and corpora they are activated for.

Annotator and Corpus Installer plugins support user corpora. An Annotator plugin runs on plain text to tag it. A Corpus Installer manages the whole upload, tag, and configure process.

Here we can see the plugin info for my Installer and Annotator plugins. Unlike Downloaders, they need a bit of extra setup data to be configured to the particular server. This setup information allows the actual interface presented to the user to bypass a whole lot of complexity.

The user-corpus install page looks like this.

Every available Installer plugin appears in the top part of the interface.

You select an installer to use by clicking on it. Then, you select files from your upload area to use. Push the button. And that is more or less it.

Once the Installer finishes work, the new corpus appears in your user corpus list.

Click on the name, and you enter the standard CQPweb interface for your corpus.

CQPweb needs to be able to talk to the plugins. So each type of plugin has certain rules about the information it receives and what it produces.

Here I've laid out the info received and produced for all three of the plugin types mentioned so far. A Corpus Installer for instance, from CQPweb receives a list of input files, which contain the raw text of the corpus to be created. It produces new, annotated versions of the files, and tells CQPweb where they are. It also answers queries from CQPweb about corpus configuration.

Each plugin is written as a PHP file – PHP is the language CQPweb is written in.

The file contains a single PHP Class which has the actual plugin program in it.

Put your plugin in the right folder on the computer that runs CQPweb – a folder called "plugins", which already has many built in plugins in it.

And now it is part of CQPweb.

Some technical detail: Each type of plugin is specified as an object *interface* – a definition of the methods that plugins of that type must provide. CQPweb loads plugin classes as required, and utilises the interface methods to execute the plugin's capabilities at the appropriate time.

Do plugins HAVE to be written in PHP? Right now, yes.

BUT you can write a really small plugin that does nothing except pass off the main work to some external program, e.g. Python, R, or Perl, and then collect the results. This allowing re-use of existing tools of any kind.

Certain CQPweb functions are designated available for plugin authors. These enable plugins to "talk to" the rest of the system. So plugins can alternatively use the "Face" tools CQPweb provides: interfaces to R and Python.

CQPweb itself uses the RFace for some of the maths involved in calculating Log Ratio confidence intervals. The code on screen illustrates CQPweb passing a statistical command to R. Plugins can use the RFace for whatever they like!

The PyFace, which allows commands to be fed into a Python process in the same way, is still work in progress. But the faces allow you to write MOST of the plugin in either R or Python, whichever your language of choice is.

Then the actual plugin PHP class can just request an Rface or Pyface; pass in data and commands; retrieve the results; and return them to CQPweb.

Either way, writing plugins is much easier than attempting to dig into the CQPweb main code!

In summary: This plugin framework is, I think, a promising example of a paradigm for enhancing extensibility and flexibility of corpus analysis software. This is the future of CQPweb!

That’s my presentation; thank you very much for your attention.