NClosEmacs for Muse: Text-Oriented Knowledge S

jmc

Overview

This document describes text-oriented knowledge services using NClosEmacs. It combines the text orientation of the flexible Emacs editor with NClosEmacs, a tentative implementation of the NClose rules engine in ELisp for embedding expert system functionality in the Emacs environment.

1 Motivation and Introduction

Emacs is the well known extensible, customizable, self-documenting real-time display editor. Among its many uses in editing at large, producing text—whether literary, technical or other—is probably the most obvious one. This is apparent in many ways: the interactive display editor which appeals to end-users for authoring and composing textual documents; the comprehensive library of Emacs Lisp functions related to buffer and text manipulations in its extensive set of libraries; and the variety of externally developed Emacs libraries for authoring and formatting text documents available on Emacs-related community sites (notably EmacsWiki).

With NClosEmacs, expert system based functionality is made readily available in the Emacs environment. Combined with the aforementioned text orientation, it affords what we'll emphatically call *text-oriented knowledge services* to the Emacs author.

In the following sections we illustrate text-oriented knowledge services in the embedded use of NClosEmacs in a popular Emacs-based authoring environment, Muse. While the actual embedding itself is Muse-dependent, the generic process of using NClosEmacs as an embedded service provider is not—the same ideas could be used with '`org-mode`' for instance. This document then also provides guidelines for embedding knowledge-based services into other Emacs-related environments and extensions. (In particular, conjunction of NClosEmacs and Emacs' subprocesses could bear interesting fruits in the provision of Emacs knowledge-based services to external programs.)

2 Text-Oriented Knowledge Services

2.1 Muse

This document relies on Muse, which was written by John Wiegley and is now maintained by Michael Olson. Several versions of the Muse manual are available on-line.

* PDF: http://mwolson.org/static/doc/muse.pdf
* HTML (single file): http://mwolson.org/static/doc/muse.html
* HTML (multiple files): http://mwolson.org/static/doc/muse/

From this documentation, “Emacs Muse (also known as ”Muse“ or ”Emacs-Muse“) is an authoring and publishing environment for Emacs.” It simplifies the process of writing documents and publishing them to various output formats.

Muse consists of two main parts: an enhanced text-mode for authoring documents and navigating within Muse projects, and a set of publishing styles for generating different kinds of output.

What makes Muse distinct from other text-publishing systems is a modular environment, with a rather simple core, in which “styles” are derived from to create new styles. Much of Muse’s overall functionality is optional. For example, you can use the publisher without the major-mode, or the mode without doing any publishing; or if you don’t load the Texinfo or LaTeX modules, those styles won’t be available.

Additionally, Muse is—true to Emacs’ inspiration and spirit—easily extensible both in terms of new styles and new functionality. We’ll rely on the later extensibility feature to embed NClosEmacs into the Muse authoring environment.

2.2 Obtaining Released Versions of Muse

Debian users can get Muse via apt-get. The ‘muse-el’ package is available both at Michael Olson’s APT repository and the official Debian repository. To make use of the former, add the following line to your ‘‘`/etc/apt/sources.list`’’ file and run ‘‘`apt-get install muse`’’.

‘`deb http://mwolson.org/debian/ ./`’

Ubuntu users can also get Muse via apt-get. The ‘muse-el’ package is available both at Michael Olson’s APT repository and the official Ubuntu repository. To make use of the former, add the following line to your ‘‘`/etc/apt/sources.list`’’ file and run ‘‘`apt-get install muse`’’.

‘`deb http://mwolson.org/ubuntu/ ./`’

The reason for making separate Debian and Ubuntu packages is that the Muse manual is under the GFDL, and Debian will not allow it to be distributed in its main repository. Ubuntu, on the other hand, permits this manual to be included with the ‘‘`muse-el`’’ package.

Alternatively, you can download the latest release from ‘http://download.gna.org/muse-el/’
.

2.3 Embedding NClosEmacs in Muse

With the Muse authoring environment, text is entered with generic markup rules and tags to instruct the Muse formatting engine. The Muse engine can produce documents in various styles, ranging from HTML to LaTeX, info and pdf, from the same source. In addition, there are several Muse major modes for Emacs to assist in document authoring and producing. (See Muse Documentation.)

In Muse the extensibility tag '`lisp`' is used to embed executable ELisp code in a document. The code is executed when the document is published, as a side effect of the '`muse-publish`' set of commands.

When Muse encounters '`<lisp>(concat "this " "form")</lisp>`' in the body of the document to be published, it actually replaces the tag content by the result of the Emacs Lisp form evaluation, namely the string "this form" in this example.

The embedding of NClosEmacs in a Muse document is then simply a matter of inserting the high-level '`<lisp/>`' markup calls to the NClosEmacs expert system library at the proper articulation points in the text.

2.4 The Muse Helper

The Emacs Lisp file '`nclosemacs-muse-helper.el`' contains high-level user functions for the integration of NClosEmacs-based knowledge services into a Muse document. An complete example of its use is provided in the '`nclosemacs-muse.txt`' document.

The Helper defines the following functions to be used within the '`<lisp/>`' markup tag in a Muse document:

– **Function: nclose-loadkb kb-file**
This function loads a the knowledge base specified by the filename passed as argument, if not already present in memory. It returns the list of in-memory knowledge bases.

– **Function: nclose-slot-value object &optional property**
This returns either the value of the slot if 'property' is present, or the value of the sign 'object' if no property is specified.

– **Function: nclose-hypo-value hypo**
This function returns the string "true" or "false" according to the status of the passed hypothesis.

Please note that the last two functions may trigger the rules engine if the values are '`UNKNOWN`' yet at publish time.

The session itself is controlled directly through calls to the NClosEmacs API, for instance:

– **Function: agenda-suggest hypo-name**
This function puts the hypothesis 'hypo-name' (a string) on top of the agenda for evaluation.

– **Function: agenda-volunteer sign-name value**
This function is used to volunteer a value for a sign.

– **Function: nclose-knowcess**
This is the usual call to run a session of the rule engine. Interactions happens in the standard message buffer at the bottom of the Emacs window, as in interactive NClosEmacs sessions.

– **Function: nclose-reset-session**
Use this function to reset the whole session. Usually found at the top of the document before the prolog, it may be used within the body of the document for later side-effects.

The Helper needs to be included with the NClosEmacs library, usually near the top of the document with the following markup text:

```
<lisp>
(require 'nclosemacs)
(require 'nclosemacs-muse-helper)
</lisp>
```

which sets everything up.

3 Conclusions

Integration of NClosEmacs into standard or popular Emacs modules has been demonstrated here with the Muse authoring environment. In this instance, the work is made simple as Muse does support extensions to its publishing process through the liberal use of the '`<lisp/>`' tag. Other Emacs modules may have different extension hooks—usually in the form of Emacs Lisp function templates—which can be similarly used to embed NClosEmacs-built knowledge services. (Examples that come to mind are mail, document authoring or workflow, and external computing environment with an Emacs fornt-end.)