Emacs Browser of types
Specifications

Christophe Muller and John S. Baras

Date : 90/06/08
Revision : 0.3
State : Draft
1 Introduction

This document specifies the requirements and functional specifications of a Smalltalk-like Browser of Types (or Classes) for Emacs.

The program will be written in Elisp and considered as a set of majors and minor modes, like gnus (News reader). Like dire or gnus it will help to manage complex and structured information (hierarchy of directories in the case of dire, hierarchy of types in the case of Browser-mode).

It is intended to be used by people programming in an Object-Oriented way, and should increase their productivity (I hope so :-)!)

Then, a configurator mode will be added. It will be close to interactive environments a-la Interface Builder or Smalltalk inspectors [4], etc... It will be able to generate code for a given configuration and then launch a compilation + running.

2 Requirements

The following points might evolve as technical problems arise or tradeoffs have to be found. For the time being, as many feedbacks as possible on these ideas would be greatly appreciated.

1. Browser-mode must be OO-language independent
That is, developers using Eiffel, C++, Objective-C, Smalltalk, or other OO languages should be able to use it with a minimum of work to do to customize it for their own language. To achieve this goal, a small description of a virtual OO language will be provided in the specification section, it should be possible to map this description to most existing OO languages.

The first version will include a sub-mode for SIMULA. It should be relatively straightforward to customize it for other OO languages.

2. Browser-mode must not interfere with other Emacs modes
Especially when editing some text (class description or method), all the languages previously cited already have a "major-mode" in Emacs which knows the language syntax. In the text window, Browser-mode will be a "minor-mode" which will not interfere with the language mode. All other
windows of Browser-mode will be Major-modes (Class window, Method window...).

3. Speed must be acceptable
   
   This is an important issue; if Browser-mode is too slow, an OO developer will not use it. Also it must be decided when the time should be spend mostly (time of parsing files or generating code for example), at the beginning of a session, equally spread during the session, or before each compilation or Make launching.. The decision depends on a lot of things: speed of Emacs, load of work to do, and methods of development of most people. Choices will not be made in this first document.

4. Browser-mode should not interfere with developers' organization
   
   That may be difficult to achieve with a simple tool, but we can precise some points. For example, the hierarchy of directories and where the sources of each classes are located should not be imposed by Browser-mode. It can be fixed by a project manager when several teams are involved, it is also possible that the compiler requires a special organization (I think it is the case with Eiffel).

5. It must be possible at every time to stop using Browser-mode and have a clean project state

   This is actually not difficult to achieve as soon as the previous point is valid and Browser-mode knows how to generate code for a particular language. This will be done for several languages (e.g., C++), and can be expanded by the user itself.

6. Reverse: It should be possible to take an existing OO library and start using Browser-mode
   
   This point here is more difficult as it involves parsing facilities of the language used. First it is technically a more complex work, and second it will requires a lot of work from the user to expand Browser-mode for other languages (future OO languages or new versions of existing ones..). Is there a Elisp-yacc somewhere?

Since the first version of this document, several people (RMS and people from SIMULOG) have suggested than “etags” should be used to parse and retrieve methods from source files. I think it’s a great idea, but from what I know, this tool only recognizes C (for etags), Lisp, Fortran, and LaTeX. So it should work only for CLOS like languages (it recognizes defmethod), what about C++, Objective-C, Eiffel, Qnap2, etc...? Is it possible to write a language specification somewhere and extend etags? (at first glance it seems that everything is hardcoded in etags.c!).
3 Specifications

3.1 Language independence

The last requirement gives a good transition for opening the specification section with a question: What should be the level of knowledge of Browser-mode about languages?

The better the level of knowledge about the language used, the more useful will be the program, but also the more difficult to write in an efficient way, maintain, and adapt to new languages.

At one end of the spectrum, one can consider (like NeXT IB) that a class equals a file and that double clicking on a class name in the browser will open an editor on that file. If you want to find a particular Objective-C method, you can make a search in the editor.

On the other end of the spectrum, the Smalltalk browser [4] comes to mind which basically offers four choices of browsing among:

- several libraries of classes (categories of classes)
- classes
- categories of methods
- methods

In four windows on top of an edit window (see figure 1).

We can note that it is much more useful to be able to browse among the methods of classes rather than just having everything stored in a file!

The "categories" menus, although quite typical of Smalltalk authorize a nice feature: when a category of class is selected, a template of class description appears in the text window, same for the methods.

As one can imagine the behavior of a window is closely related to what happens in the other windows; when a class is selected the class description

\footnote{Note: However, several features like constraints or outlets (i.e., connections between objects) are graphically managed by Interface Builder and stored in a resource file, the point is that the programmer should have to call the editor as few time as possible, at least when developing an interface}
<table>
<thead>
<tr>
<th>Class Categories</th>
<th>Class Names</th>
<th>Methods Categories</th>
<th>Methods Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>Editor</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 1: Smalltalk Browser

appears in the text window and a list of its method categories appears in the following window, etc...

Section ?? will present a principle that authorizes a great flexibility in the coding and maintenance of the different window modes (this principle is OO of course.. :)

Other useful features that will also strongly depend on the language used could be:

- To be able to change graphically signatures of methods,
- To be able to follow pointers to classes used by the current class (import) or using it (export) (if “etags” is used, this should not be a problem),
- To be able to support assertions and class invariants,
- To be able to associate non-code objects to classes (e.g., documentation files, graphical descriptions, data files, etc...).

Only imagination imposes limits, and also the difficulty of implementation. That is why the issue of deciding a “maximum level of language knowledge” is crucial. We cannot bind the mode too closely to a particular language for the work to be done to develop and maintain it would be too important.
For example a very few OO languages know about methods categories for an object, but this is not so important as this structure could appear only at the browser level and be flatten for the compiler. Also a very few languages make a distinction between class methods and instance methods (there is a switch button for that in the Methods Categories window of the Smalltalk Browser), maybe just Smalltalk and Objective-C. Can we decide to use this feature even if a lot of languages will never be able to use it?

A possible answer to this debate is openness and extensibility. We can provide a "shell" of browser and some tools that can be used by the designers of new adaptations to languages. Roughly, a tool will be one Emacs window and a set of functionalities attached to it (e.g., libraries browsing window, class browsing window).

Also, for the two last points, RMS gave an interesting idea: The idea would be to separate a class source into two kind of information: a class definition header and a list of "outlets" which could be methods, but also every list-type features like attributes, pointers on documentation or data files, etc... Then to switch a feature type from one category to another (from class header to outlets), we only need to be able to find this information into the source code (i.e., to parse it with etags or something else). As we can see, the flexibility of parsing is a very important issue too.
These tools/windows could then be:

- Assembled in a new "submode" (e.g., SIMULA, or My-C++-mode)
- Modified (to customize, add or change functionalities)
- Created (to answer new needs, for example, an Eiffel user will create the new “assertion management” window, a C++user will create a “members” window, etc...)

To draw a first draft of the situation, the next section will describe a virtual OO language. Browser-mode shall be used at least with every languages that can be mapped to this description.

### 3.2 Description of a Virtual Object-Oriented Language

The chosen representation will be a mixed of BNF-like rules for non-terminal descriptions and RECORD-like arrays for terminal descriptions. We won’t give any syntax indication, as the purpose is not to design a new language but to structure this information for a graphical interface.

**BNF-like syntax:**

- Non terminal are enclosed between brackets <...>
- ::= means “is”
- No sign in a right formula means “AND”
- | means “OR”
- ( <item> ) represents a list of <item>

**RECORD-like representation:**

An array of fields will be drawn with the following information

<table>
<thead>
<tr>
<th>Name of the Attribute</th>
<th>Type of the Attribute</th>
<th>Default value</th>
</tr>
</thead>
</table>

6
<Class> is

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>STRING</td>
<td></td>
</tr>
<tr>
<td>ancestors</td>
<td>(&lt;Class&gt;)</td>
<td>()</td>
</tr>
<tr>
<td>methods</td>
<td>(&lt;Method&gt;)</td>
<td>()</td>
</tr>
<tr>
<td>description</td>
<td>TEXT</td>
<td></td>
</tr>
</tbody>
</table>

<Method> is

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>STRING</td>
<td></td>
</tr>
<tr>
<td>body</td>
<td>TEXT</td>
<td></td>
</tr>
</tbody>
</table>

Remarks:

As explained in the last paragraph, the “Class description” will be a sort of catch-all feature that describes everything that is not “outlet”. It will be a .H file for C++ for example, or the part composed of attributes, export clause, class invariants for Eiffel.

It is easy to implement and constitute the first draft of a more powerful submode that will be able to parse this information and then permits a graphic (i.e., Emacs-Oriented) management of it.

The minimum information needed on a language to fit this description is relatively straightforward to deduce: keywords and delimiters for header of a class, ancestors specification, and list of methods. Then the code generation will mainly consist of a cat of these different components.
3.3 Example of default assembly

In this section, we will see an example of default assembly of windows proposed by Browser-mode. I have no clues about the questions “Is it the best one?” “What should we do?” and so for. Maybe a configuration with just a window of class and methods selecting could be more appropriate, I put libs selecting because the drawing was made and well, it doesn’t really matter of the language... (you can always group classes together). Any comments or critics are welcomed.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure2.png}
\caption{Emacs default Browser-mode}
\end{figure}

The next section gives a first draft of the conceptual model for such a configuration. Detailed specifications of each part will be provided in the next version of this document.

3.4 Conceptual Model

The following table is a view of Browser-mode’s main functionalities, the objects (from a user point of view) on which they operate, and the list of authorized operations.

\begin{itemize}
  \item It is nice to have a tree-like representation of classes but I don’t know how to handle multiple inheritance properly... Any ideas?
\end{itemize}
<table>
<thead>
<tr>
<th>Functions</th>
<th>Objects</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Libraries Management</td>
<td>Set of Libraries of classes</td>
<td>Add</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Remove</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Order</td>
</tr>
<tr>
<td>Classes Management</td>
<td>Set of Classes descriptions</td>
<td>Add a subclass</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Remove a subclass</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Move a subclass</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Retrieve a description</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Store a description</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Create a template</td>
</tr>
<tr>
<td>Methods Management</td>
<td>Set of Methods</td>
<td>Add a method</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Remove a method</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Retrieve a method body</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Store a method body</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Create a template</td>
</tr>
<tr>
<td>Text Editing</td>
<td>Descriptions of Classes or Body of Methods</td>
<td>The usual operations -</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- permitted by Emacs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Switch to other window</td>
</tr>
</tbody>
</table>

Table 1: Conceptual Model

For the first three tools/windows, existing similar Emacs modes have to be studied to keep a consistency in offered functionalities (and of course in the interface too!). It comes to my mind the feature of virtual delete and confirmation for example (like in Dired).

### 3.5 Example of extension for a language: Eiffel

This paragraph shows a possible extension, it is not the definitive specification for the Eiffel sub-mode, but just an example of one. If the system is flexible enough we can imagine that each inventive programmer might come with fresh new ideas.
If we add typical specifications to our virtual language description, this could give something like:

<Class> is

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>STRING</td>
<td>^n</td>
</tr>
<tr>
<td>ancestors ( &lt;Class&gt; )</td>
<td>()</td>
<td></td>
</tr>
<tr>
<td>attributes ( &lt;Attribute&gt; )</td>
<td>()</td>
<td></td>
</tr>
<tr>
<td>methods ( &lt;Method&gt; )</td>
<td>()</td>
<td></td>
</tr>
<tr>
<td>description</td>
<td>TEXT</td>
<td>^m</td>
</tr>
</tbody>
</table>

Note: description is still a catch-all for a lot of things...
(e.g., I left aside class invariant and export clauses)

<Attribute> ::= <Scalar_attribute> | <Outlet> | ( <Attribute> )

<Scalar_attribute> is

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>STRING</td>
<td>^n</td>
</tr>
<tr>
<td>type</td>
<td>&lt;Scalar_type&gt;</td>
<td>INTEGER</td>
</tr>
<tr>
<td>default_value</td>
<td>STRING</td>
<td>&quot;0&quot; (Case Of type)</td>
</tr>
</tbody>
</table>

<Scalar_type> ::= INTEGER | REAL | BOOLEAN | CHARACTER

An Outlet is a reference (or pointer) on an object. Eiffel doesn’t accept inclusion of objects in attributes.

<Outlet> is

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>STRING</td>
<td>^n</td>
</tr>
<tr>
<td>object_type</td>
<td>&lt;Class&gt;</td>
<td>Nil</td>
</tr>
</tbody>
</table>
<Method> is

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>STRING</td>
<td>&quot;&quot;</td>
</tr>
<tr>
<td>body</td>
<td>TEXT</td>
<td>&quot;&quot;</td>
</tr>
</tbody>
</table>

If we then imagine a modification of Browser-mode for Eiffel, we could have for example five windows on the top menu raw:

Libraries / Classes / Categories of Methods / Methods / Attributes

With the following conceptual model for attribute editing:

<table>
<thead>
<tr>
<th>Functions</th>
<th>Objects</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes Management</td>
<td>Set of Attribute descriptions</td>
<td>Add, Remove, Move, Change type (scalar/outlet), Select state (private/public)</td>
</tr>
</tbody>
</table>

Table 2: Conceptual Model for Attribute editing

The process of creating such new tools will be fully documented.

3.6 Configuration board

This last paragraph is just here to draw an idea of tool that could be very useful for Object-Oriented simulation languages like Qnap2 [1], Sim++, Modsim II, etc...

It will basically replace the Text editing window with a board on which the user can create instances of objects and then edit them with inspectors, connect them together, and generate the code of a main program before launching a compilation.

Such an "Object Editor" will certainly not be used by programmers who prefer a text editor to create their objects, but it might be an interface between the program, especially when the program is a model of a real system, and the external users (customers, managers). They will be able to modify parameters, add objects, and see the results without knowing the language or how to launch
a compilation. Simulation models comes to mind but it can be used each time some people need to modify a program without knowing the software technology.

As for the languages' specialized features (and this one also depends a lot on the language used), it is just a tool/window to add to Browser-mode.

4 Documentation to be provided

- Specification report (this document) (LaTeX)
- Design report (LaTeX)
- User manual (Texinfo)
- Programmer manual (Texinfo)

Which will examine the following points:
- How to extend Browser-mode to new OO languages
- How to modify existing choices (e.g., adding new windows on the top menu raw) for your favorite one.
- To_Do list

5 Conclusion

Detailed specifications will be added after the conceptual model in the next released of this document. I'd like to have comments on the requirements and the architecture described before going further in that direction. Please feel free to send your comments, ideas, etc...to:

muller@phoenix.src.umd.edu

Thanks for listening.
References


