

Ocamlwizard: smart tools for IDEs

Proposal for the OCaml Summer Project
Sylvain Conchon (sylvain.conchon@lri.fr)

Abstract

In the attempt to eventually provide a nice IDE for OCaml, we propose to design and implement small yet powerful tools for contextual help and refactoring. These will be batch-oriented tools to be called from existing IDEs (such as Emacs or Eclipse), in the UNIX tradition. We believe that a development performed outside of any IDE will allow the three-month effort to be mostly focused on OCaml-specific tools, without being disturbed by IDE-related considerations.

1 Proposal

Current integrated development environments (IDEs) for OCaml provide basic features such as syntax highlighting, code indentation or compilation support, but few, if none, OCaml-related features. Examples of such features are type-directed completion, pattern-matching macro-generation, code refactoring, etc. For instance, completion is implemented in only one of the three existing Eclipse plugins for OCaml [2, 3, 4], and it is incorrect in presence of sub-modules. The other features are not implemented at all.

We propose to develop tools to bring such features to existing OCaml IDEs (Emacs modes, Eclipse plugins). Instead of extending the existing code of these IDEs, we propose to design independent external tools, to be interfaced with IDEs in a second step. More precisely, we propose to implement batch-oriented tools, in the UNIX tradition. We believe this approach has several advantages:

- the tools are developed in OCaml, instead of using the IDE host language (Emacs Lisp, Java);
- the tools are developed *once* and then easily *reused* in several IDEs with little glue code;
- the three-month development effort of this summer project will be mostly focused on the desired features, not on IDE-related issues.

In particular, the batch-oriented aspect of the tools we propose should allow them to be even used independently of any IDE.

The IDE features we have in mind can be classified in two topics:

- *contextual help*: to help users writing code efficiently and using large libraries, IDEs must provide an assistance which depends on the context. Relevant OCaml-specific contextual helps include

- *path completion*: the user writes the beginning of a path expression (*e.g.* `M.A.cr` or simply `M.`) and the tool completes the path (*e.g.* `M.A.create`) or proposes a list of possible completions when there are several possibilities. Depending on the context, paths can be restricted to types, values, modules or signatures. The notion of path expression to be completed can be extended to include record labels and object methods.
- *pattern-matching*: the user writes the beginning of a pattern-matching construct, `match e`, and the tool generates a skeleton with all constructors, depending on the type of `e`. A similar feature for `function` can be considered when type information is available from the context (signature, user type annotation, etc.).
- *uncaught exceptions*: in many situations, it is highly valuable to know the set of exceptions which can be raised when evaluating an expression. Such an information is provided by `ocamlexc` (a tool developed by X. Leroy and F. Pessaux for an old version of OCaml and recently ported to OCaml 3.10 by N. Barnier). A possible application of this feature is the ability to automatically surround an expression with a `try/with` construct.
- *code refactoring*: in addition to contextual help, IDEs usually provide global operations on source code known as code refactoring. Relevant OCaml-specific operations would include
 - *renaming* files, modules, types or values through all code;
 - *moving* modules, type definitions or values from one file to another;
 - *adding or deleting `open` directives* automatically, within one or several files.

2 Technical Details

There are two technical elements which are required for the success of this project:

- a scoping analysis which gives the definitions and uses of all identifiers;
- the ability to get the type of any sub-expression.

The second element is already provided by the OCaml compiler through its `-dtypes` option. The first element consists in traversing the parsed tree while building tables; this is already performed in OCamlweb [1] and could even be simplified using `Camlp4`.

These two analyzes require an OCaml parsed tree. To be used on incomplete source files, as it is usually the case for contextual help, we also need to be able to get a parsed tree from a syntactically incomplete file. There are at least two ways to do that: either by completing the file to make it syntactically correct, or by extending an existing OCaml parser to make it more robust to syntax errors.

Once these key elements are set up, there are remaining issues, such as name captures or signature mismatches resulting from code refactoring. Design choices have to be made: should the tool fail with an explicit error message (“I cannot rename `f` into `g` due to capture in module `M` line `l`”) or try to make complex code transformations (using cascading

renamings and module wrappers)? Whatever the choice, we will focus first on soundness of the tools.

3 Applicants

The project will take place at Orsay (France) in the ProVal research team of INRIA. This team provides an ideal environment for such a project. It has a long tradition of developing tools in and for OCaml (Coq, the Why platform, CiME, Alt-Ergo, Ocamlgraph, bibtex2html, etc.). Beside the mentor, several members of this team are likely to provide technical assistance to make the project successful, including Jean-Christophe Filiâtre (author of Ocamlweb), Romain Bardou (Ocamlbuild specialist) and Aurélien Oudot (developer of an Eclipse plugin). Moreover, Albert Cohen (author of the Emacs Tuareg mode) works in the same building.

3.1 Students

The project will be realized by two graduate students (first year of Master in Computer Science) from University Paris-Sud 11 (Orsay, France):

- Mohamed Iguernelala (mohamed.iguernelala@yahoo.fr)
- David Baudet (david.baudet@lri.fr)

3.2 Faculty Member

The project will be supervised by Sylvain Conchon (sylvain.conchon@lri.fr), assistant professor at University Paris-Sud 11 (Orsay, France). Sylvain Conchon is teaching functional programming at both undergraduate and graduate levels. The corresponding courses include programming projects in OCaml (compilation, software engineering, games). Sylvain Conchon is conducting research in the area of automated deduction. He is the author of the SMT solver Alt-Ergo (<http://alt-ergo.lri.fr/>) and co-author of Ocamlgraph (<http://ocamlgraph.lri.fr/>). Sylvain Conchon has published in programming and functional programming conferences such as ICFP, ESOP, ML workshop, TFP. Sylvain Conchon is PC member of ML'08.

4 Deliverables

The expected outcome of this project is the following:

- the `ocaml-wizard` tool, distributed under the GNU Library General Public License Version 2;
- its documentation, containing both a tutorial and a reference manual;
- a proof-of-concept integration in both Emacs and Eclipse, as additional features to existing environments.

References

- [1] OCamlweb. <http://www.lri.fr/~filliatr/ocamlweb/>
- [2] Ocaml Development Tools. <http://ocamltd.free.fr/>
- [3] EclipseFP. <http://eclipsefp.sourceforge.net/ocaml/>
- [4] OcaIDE. <http://ocaml.eclipse.ortsa.com:8480/ocaide/>