Experiments with CoqIDE and PIDE

Makarius Wenzel
Univ. Paris-Sud, LRI

July 2012

Project **Paral-ITP**
ANR-11-INSE-001

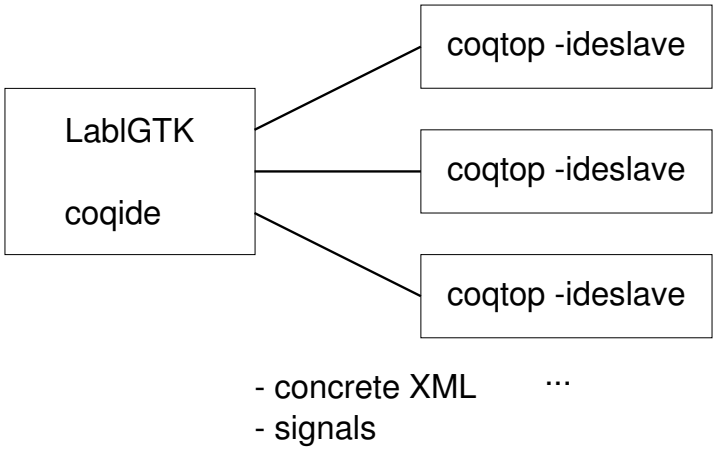
Last TODO (May 2012)

- from bottom: Coq participates in Scala integration layer
  $\rightarrow$ more people getting acquainted with Scala/JVM technology
- from top: more applications, e.g. by early adopters out there
  $\rightarrow$ more student projects
- administrative side-conditions:
  - reliable OSS licensing, to accommodate existing situation of Coq and Isabelle
  - software-technical organization of sources and repositories

Here: only the first point

CoqIDE (Coq v8.4 beta)

- remake of Proof General in OCaml — on-board technology
- GUI based on LablGTK (mainly GText widget)
  (originally by Benjamin Monate, then at LRI)
- interprocess communication `coqide` $\leftrightarrow$ `coqtop -ideslave`
  (by Vincent Gross, then at pi.r2)



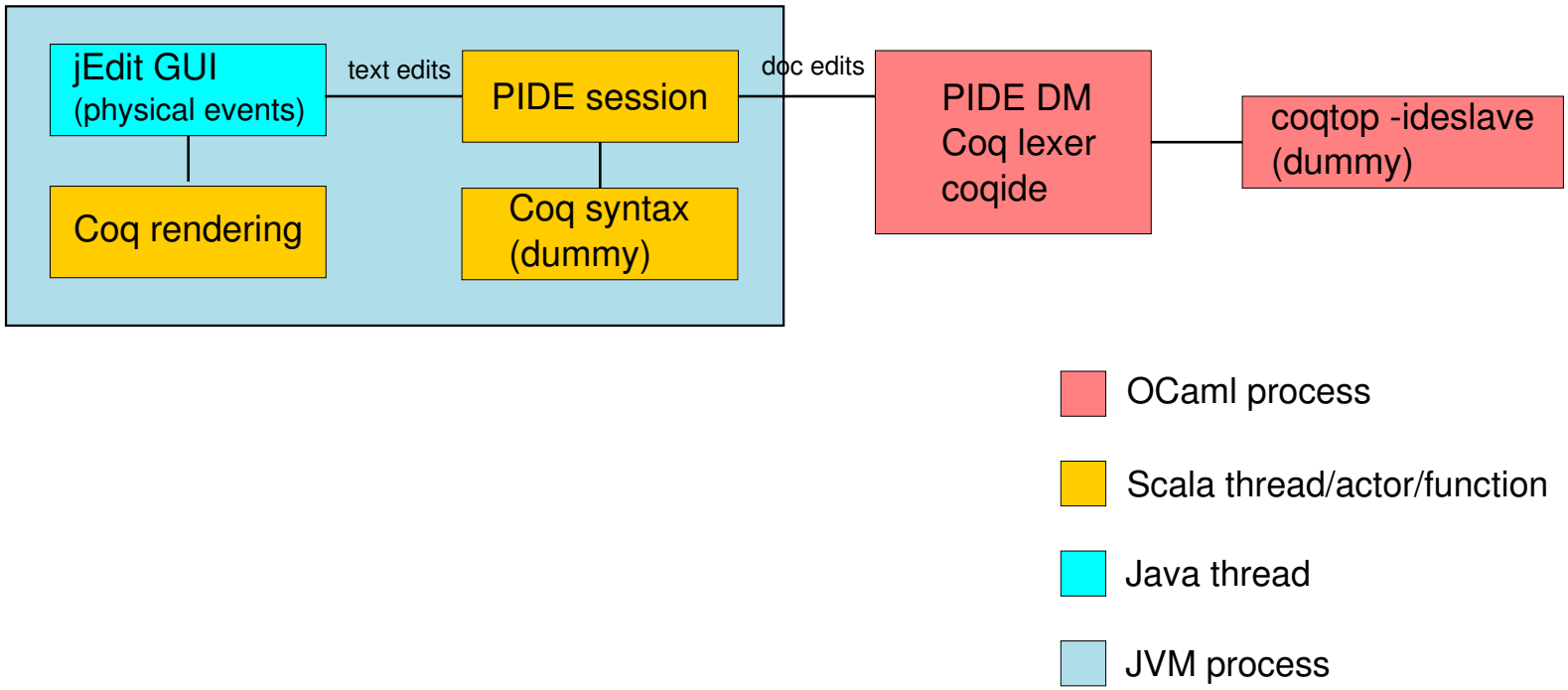
CoqPIDE (June 2012)

- integration of Coq into PIDE, coverage of its main layers
- re-use of CoqIDE architecture to accommodate PIDE
- replacement of Gtk front-end by PIDE document-model operations
  (internal PIDE protocol in OCaml, using some CoqIDE functions)
- $\approx$ 6 days work, including $\approx$ 2 days struggling with Coq makefiles

Sources:

- `https://bitbucket.org/makarius/isabelle-coq`
  (clone as `Isabelle-Coq` e.g. revision 07c7fc8ba7bd)
- `https://bitbucket.org/makarius/coq-clone`
  (clone branch `v8.4` e.g. revision 4b806b6e1757)
- See `coq-clone/README.PIDE`
- See especially coq-clone/ide/pide.ml for main implementation
  (25 kB total; 1.7 kB payload for Coq interpretation of content)

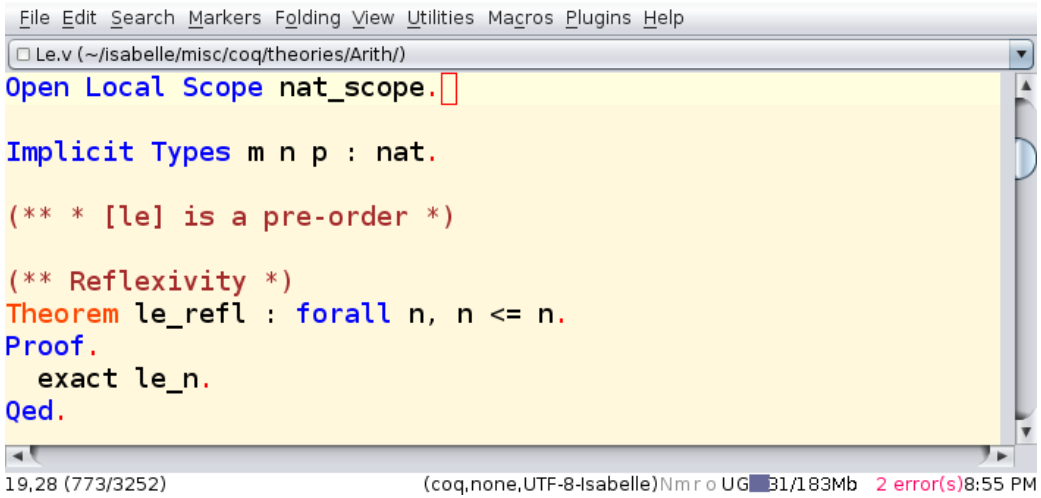
CoqPIDE building blocks



Notable Coq modules encountered so far

- `ide/coqide_main.ml` — main `coqide` process entry
- `ide/coq.ml` — operations to manage `coqtop` slave processes
  - `type Coq.coqtop`
  - `Coq.spawn_coqtop`, `Coq.respawn_coqtop`
  - `Coq.kill_coqtop` — based on kill -9 (!)
  - `Coq.break_coqtop` — based on kill -2
  - Signalling *only* from `coqide` to `coqtop` within OCaml process world; special treatment for Win32
    Note: OCaml `Sys.Break` ≈ Isabelle/ML `Exn.Interrupt` (but more control here)
  - `Coq.interp ...` `Coq.mkcases` main operations to evaluate and manage on Coq toplevel commands
    Note: closed type `Ide_intf.call` versus open-ended table of protocol functions in PIDE; different type-discipline

- `lib/xml_lexer.ml`, `lib/xml_parser.ml` — xml-light derivative
  - used to represent Coq IPC protocol by V. Gross
  - not fully XML compliant (!), bad treatment of whitespace (?)
  - PIDE prefers YXML with XML/ML data encoding (8 kB total)
- `ide/coq_lex.mll` — shadow-lexer for CoqIDE, not Coq
  - `Coq_lex.delimit_sentence`
  - used in CoqIDE to determine command spans
  - used in CoqPIDE to produce token markup (semantic payload of this experiment)



TODO

- actual execution of Coq commands, preferably via value-oriented toplevel (stateless/timeless, with abstract management evaluations, tasks, worker processes)
- Coq messages with symbolic pretty trees and additional markup
- Coq reports with position/markup for more semantic annotations of the source
- management of dependencies between document nodes, preferably via stateless exchange of `.vo` content (exchange values via OCaml serialization instead of writing to physical file-system)
- external parsing of Coq spans in Scala (optional, but useful)
- more serious integration into Coq, maybe as "plugin"