Paral-ITP Front-End Technologies
interim report

Makarius Wenzel
Univ. Paris-Sud, Laboratoire LRI, UMR8623, Orsay, F-91405, France
CNRS, Orsay, F-91405, France

March 20, 2013

Abstract

This is an overview of front-end technologies for pervasive parallel ITP systems, after the first stage of the project Paral-ITP (ANR-11-INSE-001). It corresponds to milestone M1.

Contents

1 Introduction

Front-end technologies connect conventional editors, IDEs, Web frameworks etc. with the document model of our PIDE framework. Assuming that the prover already provides native support for editing proof documents (in its ML world), and that the results of continous processing of this formal contents are accumulated (in the Scala layer outside the prover), front-ends merely need to retrieve existing information and expose it to the user.

In particular, front-ends do not have to understand formal text. Instead of imitating tasks by the prover, they concentrate on visualization and editing support. Since the connecting infrastructure is based on Scala [4], we can leverage existing frameworks on the Java VM for text editors (such as jEdit[1]) or Web services (such as Play[2]).

There are many possibilities for front-ends. The main focus within the project is on the following aspects.

1. **User Interaction:** Connecting provers and people, by means of a Prover IDE, based on conventional "rich client" platforms, or light-weight Web frameworks as alternative option.

2. **Asynchronous Agents:** Integrating add-on tools that analyze partially processed theories and proofs, and recommend edits to the user (based on automated proof tools, counter-examples, template engines etc.).

So far, our standard Prover IDE is Isabelle/jEdit, which was already available as *experimental* version in Isabelle2011 before the start of the project [6, 10]. jEdit is a sophisticated editor framework that can be augmented by genuine IDE features. It is more light-weight than Netbeans or Eclipse. Building on our document-model API, such Prover IDE applications become feasible to implement and maintain (unlike past failures to connect provers to IDEs directly). Future projects on formal-method tool-integration will also be able to re-use our work.

2 Activities and achievements

2.1 Official Isabelle releases

These are the two main Isabelle releases within the project for far, with full Prover IDE integration; a third release is anticipated before the end of the project.

- Isabelle2012 (May 2012)
  http://isabelle.in.tum.de/website-Isabelle2012
- Isabelle2013 (February 2013)
  http://isabelle.in.tum.de/website-Isabelle2013

[1] http://www.jedit.org
[2] http://www.playframework.com/

Starting with Isabelle2012, the approach of *rich client platform* is taken more seriously than in the past. Isabelle/PIDE is now routinely available on all three platform families: Linux, Mac OS X, and Windows. Such technical side-conditions affect the way how interactive theorem proving is received in a wider user audience: being able to download and run the fully integrated prover with add-on tools and IDE front-end opens up applications of theorem proving to a larger audience. There is already a notable change of the profile of typical subscribers to the Isabelle mailing list, for example.

After some investment of network bandwidth and disk space by the user, the Isabelle Prover IDE delivers a full-scale environment for interactive theorem proving. In the past, problems of compilation and installation of major proof assistants was often seen as motivation to move towards *Web clients* with some remote prover service in the background (e.g. [2]). This accidental aspect is no longer relevant for current Isabelle releases, but Web clients remain interesting for different reasons. For example, prover access via small mobile devices might become relevant for educational purposes [10].

2.2 Practical feedback via Isabelle tutorials

The Prover IDE is meant for production use of our ITP systems. Occasional tutorials help to assess the degree of usability achieved so far, by looking closely how the participants cope with the system. This was done informally and without systematic evaluation, since the project is not primarily about Human-Computer-Interaction in its own right.

- Isabelle tutorial at Orsay (LRI) 14/15-May-2012
- Isabelle tutorial at Orleans (LIFO) 24/25-Oct-2012

The two Isabelle tutorials in 2012 were using the Isabelle2012 release, with quite different groups of participants.

In May 2012 we have had a minority of Coq experts (from the project consortium) and a majority of people with general interest in formal methods and tools, but with little or no experience in theorem proving. It was interesting to see how this mixed group was exposed to the inherent challenges of formal theory and proof development. At least there were little technical problems to run the system and work with the editor environment. After the two days of the course, many participants have got a taste for interactive theorem proving and wanted to see more.

In October 2012 there was a majority of experienced Coq users who wanted to learn about formalization in Isabelle/HOL and structured natural deduction in Isabelle/Isar. At the same time they had a chance to see life beyond

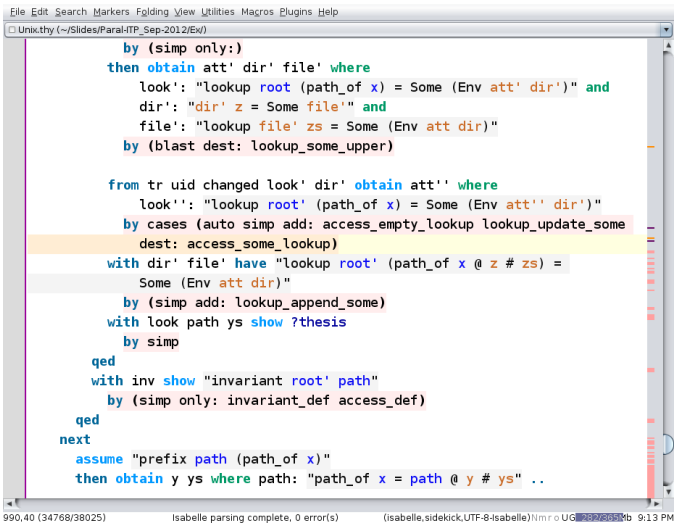
Proof General / Emacs and TTY-style proof script editing. There were interesting discussions about the new style of working in Isabelle/jEdit, and various fine-points to be seen for further improvements. An important lesson learned here is that experienced Coq users are not any different from experienced Isabelle users: there is a need of re-education to go beyond the customs of Emacs-based proof scripting from almost 15 years ago [1].

2.3 Conceptual advances of the Prover IDE

The Isabelle/jEdit Prover IDE has emerged slowly since the beginnings of Isabelle/Scala in 2007. The start of the project coincides with first release that was marked officially as "stable": Isabelle2011-1 (October 2011), see also [7]. Substantial conceptual and technical advances have happened during the past 12–18 months.

What is particularly important is the interplay of the *parallel evaluation* of the prover back-end (with continuous checking of partial proof documents) and the *asynchronous interaction* by the user within the editor front-end. This problem had already become appararent in early versions of parallel Isabelle [5, 3]: classic TTY-based interaction and its canonical Emacs front-end (Proof General) no longer fit to an increasingly parallel proof engine — where exceptions and interrupts interfere with parallel evaluation tasks in real-time.

An overview of the current state (end of 2012) of parallel evaluation versus asynchronous interaction in the Isabelle Prover IDE is given in [8]. The subsequent screenshot taken from that paper shows how the front-end visualizes the various aspects of *command status* information during the ongoing evaluation process, while the user continues editing.



In particular, a proof element that is forked, but still unfinished in the next round of the editing protocol, is interrupted in an *unstable* situation. Its transaction has to be reset and restarted afterwards. Without that explicit mechanism in the proof document-model, the prover would have to be more defensive and join forked proofs at the boundary of single commands, which would reduce the potential for parallelism unduely.

Here is an overview of further important lessons learned in the course of refining the Prover IDE, according to the state of spring 2013.

- For usability *negative syntax analysis* (failure) is more important than *positive syntax analysis* (success). Errors should not lead to exceptional termination of syntax processing, but are better internalized into the result (with markup including error messages etc.). The result is visualized eventually in the front-end.

- *Editor perspective* (visible text range) needs to be taken into account for partial parsing to achieve efficiency and reactivity. The incremental proof processing happens within the context of the front-end state, which includes open windows and cursor position. The user is more interested in quick response about the small area where he is presently working, and less in the status of big libraries loaded in the background.

- *Change of perspective* — scrolling, opening or closing of text views — counts as regular edit in the document model. It tells the prover about the small part of a potentially large theory collection that is currently of interest. A notably violation of this principle of locality is the *overview column* in Isabelle/jEdit, which visualizes the status of the whole theory node and consequently leads to some performance and reactivity issues after spontaneous updates, when it requires more than 20 ms to redraw.

- *Change of module dependencies* (theory headers) affects outer syntax dynamically. This poses an incremental bootstrap problem for processing of theory sources, which is addressed in Isabelle2012 by determining command keywords in Isabelle/Scala outside the prover. The Isabelle header syntax has been reformed accordingly to make it possible to reconstruct some prover command keywords externally, without semantic processing.

- Robust implementation of front-end technology is *encumbered by stateful models* of text editing that are still seen in Java/Swing and jEdit. Threads and locks over mutable objects and arrays are not ideal for efficient parallel + interactive processing, and need to be worked-around in many situations.

The slight mismatch of physical GUI components (with mutable state) and our mathematical document-model (with timeless and stateless update operations) is a price to pay for re-use of existing editor frameworks on the Java VM platform. The PIDE aspect of this project is not about re-implementing full-scale editor and IDE components, so we need to make best efforts to accomodate existing components.

Note that the assumption that our pure ML world already provide adequate editor frameworks was found unrealistic, leading the the "two-legged" approach of Isabelle/ML (back-end) versus Isabelle/Scala (front-end) in the first-place. This is in contrast to the classic "one-legged" approach of CoqIde in OCaml.

2.4 Experimental CoqPIDE front-end

Since Coq prover architecture was still lagging behind for full PIDE connectivety in 2012, some simple experiments were conducted to see how the protocol layers of Isabelle/ML/Scala can be converted to Coq/OCaml/Scala.

In the past, OCaml was quite capable to integrate mainstream C libraries, such as GTK for GUI components, although that is now outdated. OCaml/GTK was used to implement CoqIde within Coq itself, but GTK does not yet provide a full IDE, not even an able text editor.

The Java platform in general, and jEdit in particular, are not free from technical problems, but Isabelle/jEdit shows how these raw industrial materials can be adopted for our provers.

The main results of the CoqPIDE experiment are as follows:

Universality. The requirements for the prover to implement the PIDE document model are easily met by porting existing SML implementations to OCaml. Note that PIDE defines only rather general principles of document editing, leaving most of the details to the prover.

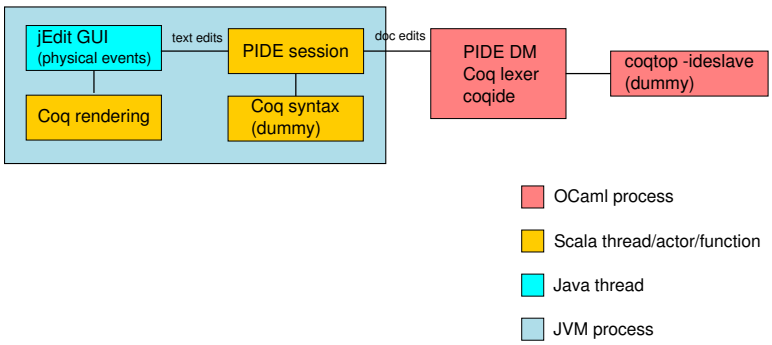
Clarity. The minimal PIDE protocol implementation for OCaml/Coq helps to explain how PIDE actually works, without the additional layers of sophistication and performance tuning that have accumulated in Isabelle already.

Frugality. A meaningful application of PIDE to a different prover merely requires 40 kB of sources in OCaml (26 kB) and Scala (14 kB). Approx. 50% is for the implementation of PIDE datatypes and protocol operations, the other 50% Coq-specific "payload". For more serious semantic processing on the prover side, the payload will grow beyond this initial PIDE configuration.

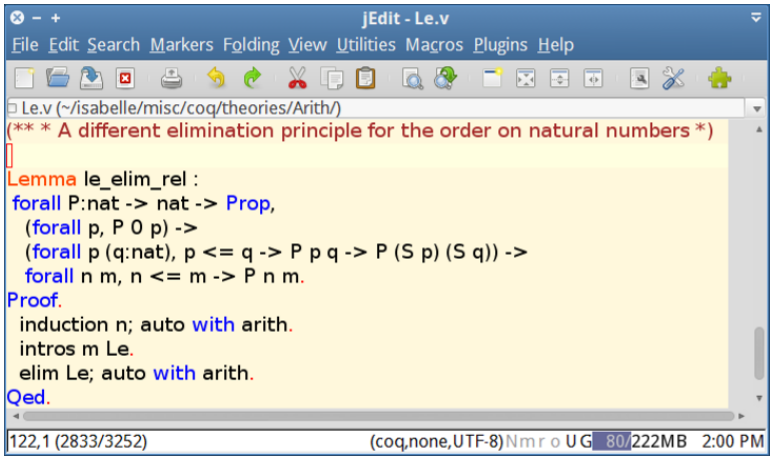
The PIDE document model maintains sources (produced by the editor) and resulting formal content (produced by the prover). Its programming interface consists of statically-typed Scala operations. `Document.update` applies source edits to turn one version non-destructively into another. `Document.remove_versions` indicates obsolete versions to allow garbage collection eventually.

Document update is declarative: it specifies where to insert or remove source text, but its operational consequences are determined by the prover. Each document version is associated with an implicit *execution* in ML to work out these formal details, and report results back to the Scala side. PIDE provides operational hints to improve performance, like `Document.discontinue_execution` and `Document.cancel_execution` in certain situations of its editing pipeline. Cancellation should cause some physical interrupt within the prover, which is important for long-running proof checking, but CoqPIDE ignores this for now.

More details of the PIDE protocol implementation for Coq in OCaml are outlined in [9, §3]. Some additional modifications in Isabelle/jEdit accomodate the Coq/OCaml back-end are described in [9, §4] as sketched below:



Taking all this together leads to *CoqPIDE*, which is a Prover IDE for Coq source files and lexical analysis produced by the prover, with asynchronous feedback according to the regular PIDE document model. In the subsequent screenshot, the GTK look-and-feel of CoqIde is imitated, even with the original default for fonts and colors.



Further semantic document content, beyond mere lexical analysis is now a matter to improve Coq accordingly, by providing more substantial proof document processing within the conventional OCaml process.

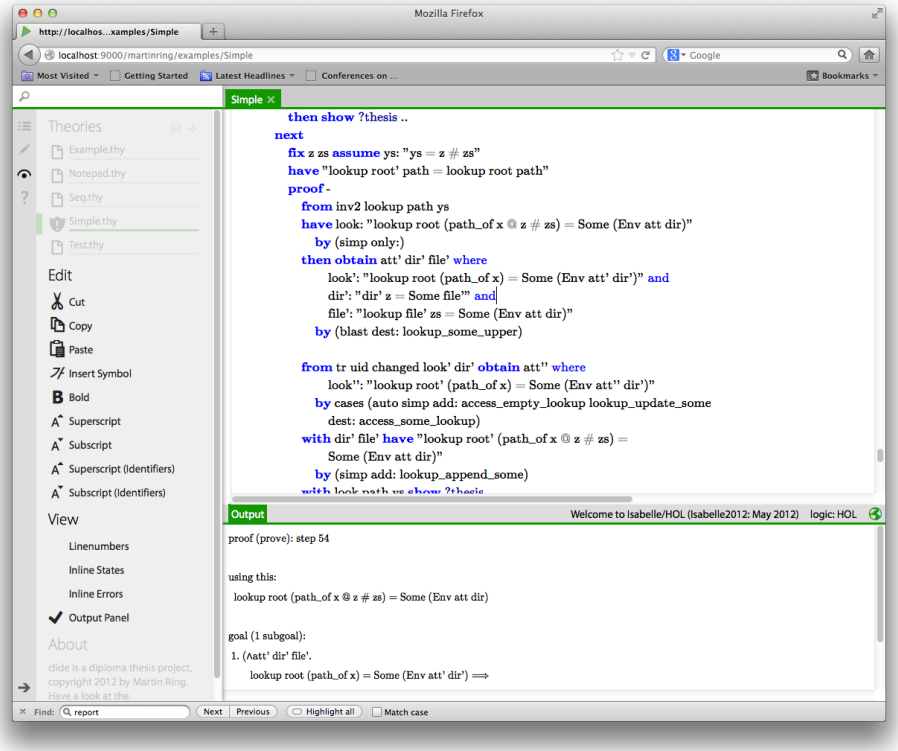
3 External contributions

Although the main implementation of the Prover IDE concepts happen to work best with Isabelle as back-end and jEdit as front-end, the framework is intended to be open on both sides. Coq as alternative back-end has been explored above. Further front-ends have been discussed with several colleagues within the project and from outside.

For example, people behind the British *AI4M* project[3] ("Using AI to aid automation of proof search in Formal Methods") have modified Isabelle2012 Scala and PIDE components to work with Eclipse, for their own purposes of tool integration around the prover. This has happened independently of our own ongoing work on prover front-end technlogies, and we are presently discussing further requirements and refinements to help such projects in the future, without demanding ad-hoc forking of the source code.

Another very interesting project to make a Web client/server application of Isabelle2012 has emerged in Bremen, by C. Lüth and his master student M. Ring, who were working in loose email contact over several months. Using Isabelle/Scala, the Play[4] framework, and some Coffescript[5] for fine-tuning of the Web client, has resulted in a very promising "cloud" version of the Prover IDE, which is called *CLIDE*. The subsequent screenshot shows Firefox running this Web client (with the server on the same machine):

[3] http://www.ai4fm.org/
[4] http://www.playframework.com/
[5] http://coffeescript.org/



Further information is available from `https://github.com/martinring/clide` — some publication is in preparation.

Consequently, our present project does not have to duplicate efforts to explore alternative PIDE implementations for the Web, as was sketched in the original proposal. It still remains to be seen how the greater perspective of "Cloud-based Prover IDEs" can be extrapolated towards further work.

References

[1] D. Aspinall. Proof General: A generic tool for proof development. In S. Graf and M. Schwartzbach, editors, *European Joint Conferences on Theory and Practice of Software (ETAPS)*, volume 1785 of *LNCS*. Springer, 2000.

[2] C. Kaliszyk. Web interfaces for proof assistants. In S. Autexier and C. Benzmüller, editors, *User Interfaces for Theorem Provers (UITP 2006)*, volume 174(2) of *ENTCS*. Elsevier, 2007.

[3] D. Matthews and M. Wenzel. Efficient parallel programming in Poly/ML and Isabelle/ML. In *ACM SIGPLAN Workshop on Declarative Aspects of Multicore Programming (DAMP 2010)*, 2010.

[4] M. Odersky et al. An overview of the Scala programming language. Technical Report IC/2004/64, EPF Lausanne, 2004.

[5] M. Wenzel. Parallel proof checking in Isabelle/Isar. In G. Dos Reis and L. Théry, editors, *ACM SIGSAM Workshop on Programming Languages for Mechanized Mathematics Systems (PLMMS 2009)*. ACM Digital Library, 2009.

[6] M. Wenzel. Asynchronous proof processing with Isabelle/Scala and Isabelle/jEdit. In C. Sacerdoti Coen and D. Aspinall, editors, *User Interfaces for Theorem Provers (UITP 2010)*, ENTCS, July 2010. FLOC 2010 Satellite Workshop.

[7] M. Wenzel. Isabelle/jEdit — a Prover IDE within the PIDE framework. In J. Jeuring et al., editors, *Conference on Intelligent Computer Mathematics (CICM 2012)*, volume 7362 of *LNAI*. Springer, 2012.

[8] M. Wenzel. READ-EVAL-PRINT in parallel and asynchronous proof-checking. In *Workshop on User Interfaces for Theorem Provers (UITP 2012)*, July 2012. http://www.informatik.uni-bremen.de/uitp12/papers/paper-02.pdf.

[9] M. Wenzel. PIDE as front-end technology for Coq. Submitted, February 2013.

[10] M. Wenzel and B. Wolff. Isabelle/PIDE as platform for educational tools. In *Workshop on Computer Theorem Proving Components for Educational Software*, volume 79 of *EPTCS*, 2012.