Eclipse Proof General

David Aspinall [1]

LFCS, School of Informatics, University of Edinburgh, U.K.

Abstract

This is a description of a plan for new research which has been awarded an Eclipse Innovation Grant for 2004. The objective is to build a proof development environment for interactive theorem provers, using ideas of the *Proof General* project, and in particular, using a protocol for interactive proof dubbed PGIP.

1 Background

The use of machine proof is becoming more widespread, and larger and more complex formalizations are being attempted in numerous interactive theorem proving systems, e.g., HOL [7], Isabelle [8], PVS [9], Coq [6]. Applications of formal proof range from the development of high integrity hardware and software systems to the desire to build a corpus of mechanically verified mathematics.

Some notable examples formal proof construction include: formalizing type safety and a programming logic for Java in Isabelle (30k lines, 1400 lemmas); the Fundamental Theorem of Algebra in Coq (80k lines, almost 3000 lemmas), and the specification and verification of pipelined processors in PVS (examples 10k lines, 500 lemmas). It should be noted that formal proof texts of this size are typically considerably more complex, dense, and interdependent than similarly sized programs, and each of these cases represents several person-years of work.

Yet, as formal proofs become larger, there has been precious little work on how to manage their development, or to apply modern software engineering techniques such as refactoring — which are desperately needed to help maintenance and re-use of proof scripts. The theorem proving community recognizes this as a problem but has yet to seriously tackle the issue. A desirable approach would be to provide dedicated support for writing proofs within a modern IDE such as Eclipse.

[1] Homepage: `http://homepages.inf.ed.ac.uk/da`.

2 Proof General and PGIP

Proof General is a generic interface for interactive proof assistants, built on the Emacs text editor [2,1]. It has proved rather successful in recent years, and is popular with users of several theorem proving systems. Its success is due to its genericity, allowing particularly easy adaption to a variety of provers (primarily, Isabelle, Coq, and LEGO), and its design strategy, which targets experts as well as novice users. Its central feature is an advanced version of *script management*, closely integrated with the file handling of the proof assistant. This provides a good work model for dealing with large-scale proof developments, by treating them similarly to large-scale programming developments insofar as Emacs supports them. Proof General also provides support for high-level operations such as *proof-by-pointing* although these are less emphasised and more difficult to implement within the limited widget set provided by Emacs.

There are some drawbacks to the present Proof General. From the users' point of view, it requires learning Emacs and putting up with its primitive, inconsistent, and at times unintuitive UI. From the developers' point of view, it is rather too closely tied with the Emacs Lisp API which is restricted, somewhat unreliable, often changing, and exists in different versions across different flavours of Emacs. Both of these drawbacks could be addressed by re-engineering the system within Eclipse.

Another engineering disadvantage of the present Proof General arose from its construction following a product-line architecture of successive generalisation. As support for new provers was added, Proof General would need more customization possibilities to allow it to be configured to work with a greater range of input syntaxes, output patterns from tools, etc. This mechanism has meant that little or no specific adjustment of the provers has been required, but it has resulted in a rather overcomplicated instantiation mechanism.

To address this concern, work has begun on a new framework architecture, dubbed *Proof General Kit* [4,3]. This architecture proposes a uniform protocol for communicating with different interactive proof systems, called **PGIP** (for *Proof General Interactive Proof*). It is based on passing XML messages. It has been designed by clarifying and abstracting the existing mechanisms used in Emacs Proof General. We hope that implementors of interactive theorem provers will be persuaded to equip their systems with PGIP support so that they automatically gain interface support from the new Proof General tools.

3 Eclipse Proof General

Proof General in Eclipse would be an appealing environment for managing formal proofs. The objectives of our research are:

- To develop Eclipse plug-ins which allow Eclipse to be used as a front-end in the Proof General Kit framework, to drive the development of proof scripts for back-end theorem provers. This will form the Eclipse Proof General application.
- To investigate the development of new JDT-like tools for proof engineering within Eclipse Proof General.

The first objective is the intial focus of our work. It is planned to build a useful application which would be a viable replacement for the present Emacs Proof General for its several hundred registered industrial and academic users. The new tool should find use in undergraduate courses in formal proof and industrial training courses in ITP systems supported by Proof General.

3.1 System architecture

Typically, proof scripts in are developed by an interactive, goal-directed dialogue with the theorem prover. The user issues a declaration, a definition, or a goal stating a theorem to be proved. If a goal is given, the system enters a mode where the user successively suggests steps in the proof, and the system responds indicating progress, and what remains to be proved. The proof mode is exited when the proof is completed. The proof steps, along with declarations and definitions, are retained in a file as evidence, and so that the proof may be replayed (and re-checked) later.

We want to use Eclipse to facilitate this process. Interaction will be centred on the proof script, by providing a facility to send lines from the text editor to the theorem prover incrementally.[2] To handle responses from the theorem prover, we need to display a range of formats of information, possibly with embedded navigation information (which provides for *proof-by-pointing* like facilities). This prover output is delivered in a custom XML-based markup language called PGML ("Proof General Markup Language"), which includes elements for representing logical symbols.

The architecture of the proposed system is shown in Figure 1. The communication between the Eclipse extensions and the theorem provers takes place in PGIP, which provides a loose coupling for interactive theorem proving systems and their user interfaces. This is appropriate because theorem provers are implemented in a variety of languages, and may be running on remote machines. The PG mediator is a middleware mediator component

[2] This is somewhat reminiscent of JDT's scrapbook pages used to evaluate fragments of Java code; but the expressions being evaluated in this case are not scrap!

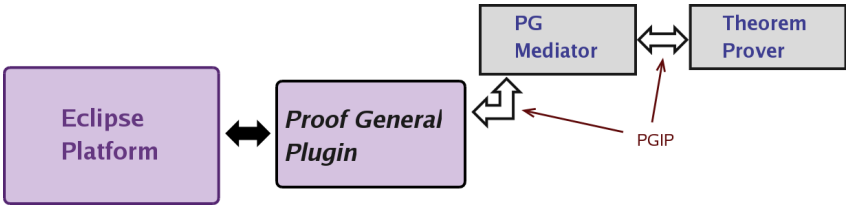

Fig. 1. System architecture

which manages the connection between possibly several interface components and theorem provers. It enforces the protocol for interactive proof, and maintains a persistent state modelling the position in a development. Interaction with the PG mediator is used to define an Eclipse project, with specialised markers stored for proof script files to represent meta-information about their contents (e.g. whether a particular proof has been processed or not).

Work so far includes definitions of PGIP, PGML, and a prototype implementation of the PG mediator in Haskell [5]. There is also an experimental patch for the theorem prover Isabelle/Isar which provides PGIP support. This project will bring support for Eclipse by development of Eclipse plugins. The new components required are:

(i) PGIP and PGML encapsulations

(ii) Proof state, object, and prover message viewers

(iii) Script Management facility provided for text editors

(iv) Syntax-configurable proof-script editors

(v) Additional toolbars and menus for navigation and prover querying, and prover preference setting (configured using PGIP).

Initially, we will focus on supporting the theorem prover Isabelle/Isar and its declarative proof language, but we will hope to extend to other systems soon.

For latest information on the project progress, see:

http://proofgeneral.inf.ed.ac.uk/kit.

References

[1] D. Aspinall, H. Goguen, T. Kleymann, and D. Sequeira. Proof General, 2003. System documentation, see http://proofgeneral.inf.ed.ac.uk/doc.

[2] David Aspinall. Proof General: A generic tool for proof development. In Susanne Graf and Michael Schwartzbach, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science 1785, pages 38–42. Springer, 2000.

[3] David Aspinall. Protocols for interactive e-proof, 2000. Available from http://proofgeneral.inf.ed.ac.uk/doc.

[4] David Aspinall. Proof General Kit. White paper. Available from http://proofgeneral.inf.ed.ac.uk/kit, 2002.

[5] David Aspinall and Christoph Lüth. Proof General meets IsaWin. In *Proc. User Interfaces for Theorem Provers 2003 (UITP'03)*, September 2003. Available from http://www.informatik.uni-bremen.de/uitp03/.

[6] Coq. Home page at Isabelle.

[7] HOL. Home page: http://www.cl.cam.ac.uk/Research/HVG/HOL/HOL.html.

[8] Isabelle. Home page at http://www.cl.cam.ac.uk/Research/HVG/Isabelle/.

[9] PVS. Home page at http://pvs.csl.sri.com/.