A Web Interface for Isabelle: The Next Generation

Christoph Lüth and Martin Ring *

Deutsches Forschungszentrum für Künstliche Intelligenz
Bremen, Germany

Abstract. We present Clide, a web interface for the interactive theorem prover Isabelle. Clide uses latest web technology and the Isabelle/PIDE framework to implement a web-based interface for asynchronous proof document management that competes with, and in some aspects even surpasses, conventional user interfaces for Isabelle such as Proof General or Isabelle/jEdit.

1 Introduction

Recent advances in web technology, which can succinctly if not quite accurately be summarised under the 'HTML5' headline, let us develop interfaces of near-desktop quality, leveraging the advantages of the web without diminishing the user experience. Web interfaces do not need much resources on the user side, are portable and mobile, and easy to set up and use, as all the user needs is a recent web browser (in particular, there can be no missing fonts or packages). The question arises how far we can exploit this technology for a contemporary interactive theorem prover.

This paper reports on such an attempt: a modern, next-generation web interface for the Isabelle theorem prover. Isabelle is a particularly good candidate for this, because it has an interface technology centered around an asynchronous document model. As demonstrated by the system presented here, Clide, we can answer the motivating question affirmatively, modulo some caveats. Readers are invited to try the public test version of the system at `http://clide.informatik.uni-bremen.de`.

2 Basic System Architecture

The basic system architecture is clear: we need a web server to connect with Isabelle on one side, and with web browsers on the other side. Hence, the questions to address are, firstly, how to connect Isabelle with a web server, and secondly, how to use a browser to edit Isabelle theory files?

* Research supported by BMBF grant 01IW10002 (SHIP).

Isabelle on the Web. Isabelle poses some specific challenges when implementing a web interface, most of which are common to most interactive theorem provers (or at least those of the LCF family). Firstly, Isabelle's syntax is extremely flexible and impossible to parse outside Isabelle. Thus, the interface needs to interact closely with the prover during the syntactic analysis. Moreover, the provided notation is quite rich, and requires mathematical symbols, super- and subscript, and flexible-width fonts to be displayed adequately.

Secondly, Isabelle's document model is asynchronous [1], meaning that at any time changes of the document can be made by the user (editing the text) or by the prover (parsing the text, or annotating it with results of it being processed). Further, the prover may be slow to respond, or may even diverge. Hence, the communication between the web server and the browser needs to be asynchronous too — the browser needs to be able to react to user input at any given time, and simultaneously needs to be able to process document updates from the prover communicated via the web server.

Finally, there is also something of a 'cultural' gap [1], with sequential theorem provers written in higher-order functional languages on one side, and asynchronous web applications written in imperative languages like Java on the other side. Fortunately, the Scala programming language provides the foundation to unify these worlds, and the Isabelle/PIDE framework [2] crosses that chasm to a large extent. It provides access to Isabelle's document model and interaction from a Scala API, encapsulating Isabelle's ML LCF core; all that remains is to connect it to a web server, and thence a browser. In our application, we use Scala together with the Akka library and Play, a fast, reliable and fully concurrent state-of-the-art framework for web development — and because of Isabelle/PIDE, it seamlessly integrates with Isabelle.

Editing Theory Files. HTTP does not allow *server pushes* where the server initiates messages to the browser, which is essential for an asynchronous model as needed here. There are workarounds such as AJAX and Comet, but for an application like this, where up to a couple of thousand small messages need to be sent per minute, the resulting message overhead is prohibitively expensive. The solution here are *WebSockets*, as introduced in HTML5. They allow for a full-duplex TCP connection with just one single byte of overhead, and are supported by all major browsers in their recent incarnations.

Secondly, Javascript (JS) is still the only viable choice for cross-browser client-side logic. Its significant weaknesses can be ameliorated by libraries such as *BackboneJS* for an MVC architecture on the client, *RequireJS* for modularisation and dependency management, and *CoffeeScript*, a language that compiles to JS but exhibits a clean syntax and includes helpful features like classes.

3 Implementation Considerations

The single most important design constraint was the asynchronous communication between server and browser; Fig. 1 shows the system architecture.

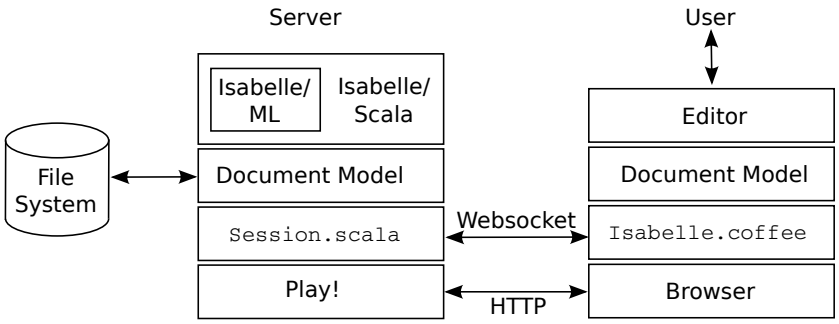

Fig. 1. The Clide system architecture

System Architecture. The asynchronous document model is implemented by two modules, `Session.scala` and `isabelle.coffee` in Scala and JS, which run on the server and in the browser respectively, and synchronise their document models. The two modules communicate via WebSockets, using a self-developed thin communication layer which maps the relevant Scala types to JS and back, using Scala's dynamic types and JSON serialisation; this way, we can call JS functions from Scala nearly as if they were native, and vice versa.

Interface design. The visual design of the interface is influenced by the Microsoft Design Language [3]. It eschews superfluous graphics in favour of typography (Fig. 2), reducing the interface to the basics such that it does not distract from the center of attention, the proof script. The prover states can be shown inline in the proof script (useful with smaller proof states, or when trying to understand a proof script), or in a dedicated window (useful for large proof states).

The Editor. The interface itself is implemented in JS, using jQuery and other libraries. Its key component is the editor. It needs to be able to display mathematical symbols, Greek letters and preferably everything Unicode; perform on-the-fly replacements (as symbols are entered as control sequences); use flexible-width fonts; allow super- and subscripts; and allow tooltips and hyperlinks for text spans. No available JS editor component provided all of these requirements, so we decided to extend the CodeMirror editor to suit our needs. For mathematical fonts, we use the publicly available MathJax fonts. This results in an editing environment allowing seamless editing of mathematical notation on the web.

4 Conclusions

Clide provides a much richer user experience than previous web interfaces such as ProofWeb [4], which is unsurprising because of the recent advances in web technology mentioned above. Comparing it with the two main other Isabelle interfaces (which are representative for other interactive theorem provers), Proof General [5] and Isabelle/jEdit [2], we find that Clide has a better rendering of

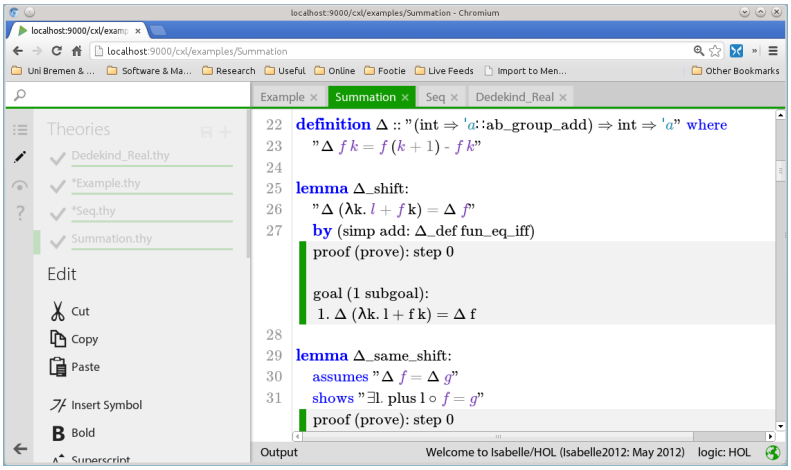

Fig. 2. The Clide interface, with the editor component on the right.

mathematical notation. It equals them in terms of responsiveness, and is easier to set up and use. However, as this is still a research prototype the user and project management is rudimentary, and the data storage could be improved by integrating a source code management system or cloud storage on the server. Moreover, we see a great potential in *collaborative* proving with more than one user editing the same theory file at the same time.

In answer to the motivating question, however, we can offer the following: web technology is ready for theorem proving, but still needs to settle down (we had to use lots of different libraries, and expect none of them to be too stable); and Isabelle/Scala is a practically useful foundation to this end, but took some effort to get acquainted with. (We gratefully acknowledge the support of Makarius Wenzel here.) Readers are invited to validate this assessment on their own. A public test version of the system is online, so why not give it a spin?

References

1. Wenzel, M.: Isabelle as document-oriented proof assistant. In Davenport, J.H. *et al.*, eds.: Intelligent Computer Mathematics — 10th International Conference, MKM 2011. Proceedings. LNCS 6824, Springer (2011) 244–259
2. Wenzel, M.: Isabelle/jEdit - a prover IDE within the PIDE framework. In Jeuring, J. *et al.*, eds.: Intelligent Computer Mathematics — 11th International Conference, MKM 2012. Proceedings. LNCS 7362, Springer (2012) 468–471
3. Microsoft: Ux guidelines for windows store apps (November 2012) `http://msdn.microsoft.com/en-us/library/windows/apps/hh465424.aspx`.
4. Kaliszyk, C., Raamsdonk, F.V., Wiedijk, F., Hendriks, M., Vrijer, R.D.: Deduction using the ProofWeb system. `http://prover.cs.ru.nl/`.
5. Aspinall, D.: Proof General: A generic tool for proof development. In Graf, S., Schwartzbach, M., eds.: Tools and Algorithms for the Construction and Analysis of Systems. LNCS 1785, Springer (2000) 38–42