Prosper: A Framework for Extending Prolog Applications with a Web Interface

Levente Hunyadi
hunyadi@users.sourceforge.net

Budapest University of Technology and Economics
Department of Computer Science and Information Theory
1117 Budapest, Magyar Tudósok körútja 2., Hungary
Phone: +36 1 463-2585 Fax: +36 1 463-3147

Keywords: web integration; application development framework

Abstract. Clear separation of presentation and code-behind, declarative use of visual control elements and a supportive background framework to automate recurring tasks are fundamental to rapid web application development. This poster presents a framework that facilitates extending Prolog applications with a web front-end. The framework relies on Prolog to the greatest possible extent, supports code re-use, and integrates easily into existing web server solutions.

When developing web applications, separating application logic (what the program does) and presentation (how results are displayed) is of paramount importance. This approach leads to a logic focused closely on the task at hand and a replaceable presentation layer that wraps all web-related issues. Prosper augments regular Prolog modules that encapsulate application logic with a presentation layer that facilitates X(HTML) content generation.

Prosper [1] has a two-layered architecture (Figure 1). The lower layer, *Prolog Web Container*, maintains a direct persistent connection to the web server through the *FastCGI protocol* [2], which caters for flexibility and easy integration with an existing web server. In addition, Prolog Web Container parses HTTP request and generates HTTP response headers and content (similarly to the PiLLoW [3] library), maintains a worker thread pool and assigns jobs to threads. The primary task of the container is to isolate the communication protocol and provide a natural view of request and session data for the programmer as well as balancing incoming request load.

Prolog Server Pages, built on top of the container, defines an XML-based document model. The conventional XML document model is extended with *special elements* belonging to a dedicated namespace each of which realizes a *transformation rule*. A transformation rule can be thought of as a Prolog predicate that defines a mapping between a source and a target XML subtree. For instance, a simple iteration rule repeats its content a specified number of times. The exact behavior of the transformation is specified by means of XML attributes. Attribute values may bind to Prolog predicates or may use the so-called *expression language*. Expression language can be seen as an extension to the `is/2` predicate to include basic atom manipulation, request context variables and user-defined Prolog functions. Prolog Server Pages also offer once-assignable local variables valid inside the server page document to propagate computed values.

Prosper includes a predefined set of special elements implementing the most common transformation rules such as conditionals and iteration constructs. However, the set of transformation rules is not restricted. Relying on the *extension infrastructure*, the user may create new modules that contain hook predicates registered for steps associated with reply generation. Modules correspond to XML namespaces and exported hook predicate names to element names in server page documents. Special elements and their implementor hook predicates are declared in a *configuration file*.

The document model allows declarative, XML-based definition of visual appearance without the use of a foreign language interface (as opposed to e.g. PrologBeans [6]). Complementing the visual part of Prolog Server Pages, *logic modules* give real power to the architecture. Logic modules (which are conventional Prolog modules) provide the code-behind that incorporates application logic and are not interleaved with the presentation layer (unlike [4] and [5]). Server pages can reference code-behind in a variety of ways: assign server page variables based on application logic, test for the satisfiability of predicates and formulate conditions using the return value of functions, thereby affecting visual layout.

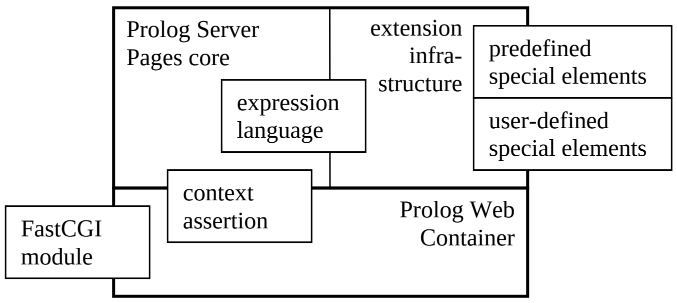

Fig. 1. The architecture of the proposed framework.

References

1. *Project page of Prosper*, `http://sourceforge.net/projects/prospear`, module Prosper in CVS *pserver:anonymous@prospear.cvs.sourceforge.net:/cvsroot/prospear*
2. Mark R. Brown, *FastCGI specification*, Document Version: 1.0, Open Market, Inc., April 29, 1996
3. Daniel Cabeza and Manuel Hermenegildo, *The PiLLoW Web Programming Library*, Reference Manual, The CLIP Group, School of Computer Science, Technical University of Madrid, January 5, 2001, `http://www.clip.dia.fi.upm.es/Software/pillow/pillow.html`
4. Benjamin Johnston, *Prolog Server Pages*, `http://www.benjaminjohnston.com.au/template.prolog?t=psp`
5. Mauro Di Nuzzo, *Prolog Server Pages: A server-side scripting language based on Prolog*, Version 0.2, April 2006 `http://www.prologonlinereference.org/psp.psp`
6. *PrologBeans and PrologBeans.NET for SICStus Prolog*, `http://www.sics.se/sicstus/docs/latest/html/sicstus/PrologBeans.html`