Getting started with

Prolog Server Pages

A SERVER SIDE SCRIPTING LANGUAGE BASED ON PROLOG

Version 0.4.1

Mauro DiNuzzo

Contents

1 Introduction

1.1 What this software is about

Prolog Server Pages (PSP) is a server-side scripting language based on Prolog. The present brief tutorial describes how to install and use PSP, which in the current implementation is released for usage under Windows 10/8/7/Vista/XP only. PSP is added to the handler mappings of the Microsoft Internet Information Services (IIS) webserver as a Common Gateway Interface (CGI) module. Support for operating systems other than Windows and webservers other than IIS as well as utilization of FastCGI instead of CGI are all planned enhancements for future releases. Nonetheless, the software can be recompiled using the provided source code at needs. Support can be asked through the project website (http://www.prologwebservices.com/).

PSP is developed on top of SWI Prolog (http://www.swi-prolog.org/) and the PSP handler is incorporated into the SWI Prolog HTTP server infrastructure. Therefore, there is native support for all standard HTTP library predicates (for details, please see the SWI Prolog HTTP server package documentation at http://www.swi-prolog.org/pldoc/package/http.html).

1.2 Features

The list of features include:

- Easy Prolog scripting in HTML pages
- Full compliance with SWI Prolog HTTP server infrastructure
- JSON Remote Procedure Call (RPC)-based web services

PSP version 0.4.1 is developed under SWI Prolog version 7.3 and comes with an installer executable for Windows. PSP complies with the requirements of the SWI Prolog HTTP library with respect to handler definition (i.e. the `http_dispatch` library), according to the directive:

```
:- http_handler(root(.), psp_handler, [prefix]).
```

As standard HTTP library does not support CGI-based server, minor changes were necessary to maintain functionality (e.g., for session management), which however are completely transparent to the end-user. This means that no extra predicates are defined and users can refer to the existing documentation sources.

1.3 Notes about the implementation

The current implementation of PSP is stand-alone, i.e. it does not rely on the SWI Prolog HTTP webserver running in the background. The choice of using CGI is motivated by the fact that internet service providers commonly provides Microsoft IIS, Apache and/or Nginx as web servers. Providing an easy-to-install product will possibly help the Prolog language to gain popularity as a server-side language even among programmers that normally do not use Prolog extensively. PSP version 0.4 introduces a new programming scheme (see below) that makes use of HTML tag attributes to insert Prolog code in HTML pages.

SWI Prolog is free-software and comes with a relatively large amount of libraries providing interfaces to popular applications such as ODBC, SQL, Semantic Web, SGML/XML and RDF. Webmasters can also benefit of several language-specific features like, for example, Constraint Logic Programming (CLP). It should be realized, however, that the best way to support spreading of Prolog language is through applications, notably web-based ones. This software is a first step towards such goal. Useful applications include, but are not limited to, content-management systems and educational web resources.

2 Start using PSP

2.1 Download and installation

The PSP installer executable containing the software and documentation is available at http://www.prologwebservices.com/ or alternatively at http://sourceforge.net/projects/prolog-ws/files. Bug-fixes and changes from previous version are described in the `changelog.txt` file that comes with the distribution.

To install PSP, just run the executable with administration privileges, being sure to have IIS installed on your machine. During installation, the sources are compiled to ensure maximum compatibility with the local server machine. Application data folders (e.g., for logs and temporary files) will be created in the user `%APPDATA%` directory. A default initialization file will also be created in this directory.

2.2 Configuration

The configuration file is a Prolog script named `config.pl`. This file is loaded just before the execution of each PSP page. The administrator has full control over all standard Prolog configuration predicates (not described here). However, it is strongly suggested not to add/remove clauses to/from the `user:message_hook/3` predicate. Control of message reporting can be done by setting the `http_reporting` prolog flag, which accepts a list of atoms indicating the types of message to be reported. For example, the following code will activate reporting of warning and error messages:

```
:- set_prolog_flag(http_reporting, [warning, error]).
```

The available message types are `banner`, `debug(Topic)`, `error`, `help`, `information`, `informational`, `silent`, `trace` and `warning` (as defined by the built-in Prolog predicate `print_message/2`). The `http_reporting` flag is initially set to the empty list, so nothing will be reported (note that the message term will be always present in the log file). However, initially the `config.pl` sets the flag to `[warning, error]` by default.

3 Prolog Server Pages

3.1 Using Prolog in HTML pages

Much like other server-side languages the Prolog code in initial PSP implementation was enclosed in special HTML tags, namely `<?` or `<?psp` and `?>`. However, intermixing Prolog and HTML is not as simple as for other programming languages like PHP or ASP. In the original PSP proposal (see http://arxiv.org/ftp/cs/papers/0603/0603101.pdf) each bracketed block of Prolog code (i.e. a chunk) was assumed to be interpreted in assert-mode. Such mechanism implies independence of chunks and the annoying consequence that variables are not persistent across chunks, which often forces the user to duplicate the code and/or use the `assert` and `retract` predicate families. Combination of HTML and Prolog scripting was subsequently achieved by making PSP functioning always in query-mode (see http://arxiv.org/pdf/0711.0917.pdf) with the introduction of the tags `<?,` and `,?>`. The latter approach was then generalized by removing the requirement about the operator that follow/precede each open/closed PSP chunk. Although the resulting code turned out to be apparently harder to read, it nonetheless preserved the capability of the language to switch between assert- and query-mode. Alternative approaches include, but are not limited to, the DCG-based `html_write` library (see http://www.swi-prolog.org/pldoc/man?section=htmlwrite). Nevertheless, these approaches makes the underlying HTML hardly recognizable, while the idea behind PSP is that each script is primarily a HTML document not a Prolog program.

3.2 The PSP syntax

In order to reconcile Prolog code readability with HTML, the present version of PSP implements a new programming scheme in order to incorporate Prolog into HTML documents. The approach is reminiscent of Prolog Well-formed Pages (PWP), which is implemented in the SWI Prolog HTTP library (see http://www.swi-prolog.org/pldoc/man?section=pwp). Contrary to PWP, the current implementation provides a unique attribute named `psp:prolog` that is valid for all HTML tags [1]. The underlying execution model is that the prolog goal contained in the `psp:prolog` attribute enwraps the HTML code contained within the relevant tag and backtracks successively.

In addition to readability, another advantage is the "incentive" to put long pieces of Prolog code into separate files, which also facilitates code re-use. Let us assume, for example, that a standard Prolog file named `database.pl` contains the following code:

```
% database.pl
:- dynamic p/1, q/1.

p(a).
p(b).
p(c).
p(d).
q(c).
```

Now consider the following PSP page:

```
<!DOCTYPE html>
<html xmlns:psp="http://www.prologwebservices.org/psp">
  <body psp:prolog="ensure_loaded('database.pl')">
    <div psp:prolog="p(X)">
      Now X is <b psp:prolog="write(X)"></b><br>
      <span style="text-decoration: underline;" psp:prolog="q(X)">
        Do something when X=c here!<br>
      </span>
    </div>
  </body>
</html>
```

The resulting code is more compact, elegant and easy to manage than chunk-based code. The generated output will be:

```
Now X is a
Now X is b
Now X is c
Do something when X=c here!
Now X is d
```

[1] Notice that before sending output, PSP removes the `psp` qualification as well as the Prolog code from attributes (in the case of `psp:prolog` the entire attribute is removed). Therefore, the documents always remains HTML-valid.

The above example also illustrated the fact that standard HTML tag attributes are untouched by the PSP parser. One limitation of the present syntax scheme is that sometimes it is necessary to use seemingly redundant HTML tags to incorporate the required prolog code, as we did above using `span`. However, as a side-effect this requirement helps maintaining appropriate context for the Prolog code. Importantly, variables are local to the HTML tag where Prolog code appears.

3.3 HTML tag attributes

Attributes can be substituted into by qualifying the attribute with the `psp` namespace [2]. For example, filling a drop-down menu requires operating on the `value` attribute of the `option` HTML tag, which is achieved by:

```
<!DOCTYPE html>
<html xmlns:psp="http://www.prologwebservices.org/psp">
  <select name="PrologFlag" psp:prolog="
    current_prolog_flag(Name, Value),
    Selected = dialect
  ">
    <option
      psp:selected="Name = Selected, write('selected')"
      psp:value="write(Name)"
      psp:prolog="write(Value)">
    </option>
  </select>
</html>
```

Note that a `psp`-qualified HTML tag attribute appears in the final HTML page only if the code contained therein succeeds. In the example above, only the `option` element with `value="dialect"` will be accompanied by a `selected="selected"` attribute [3].

3.4 Working with HTTP headers

In order to perform actions before HTTP headers are sent, PSP requires that Prolog code is contained in the `<html>` tag. Below is an example to implement server-side redirection:

```
<!DOCTYPE html>
<html xmlns:psp="http://www.prologwebservices.org/psp"
  psp:prolog="
    http_current_request(Request),
    http_redirect(moved, 'http://www.prologwebservices.com/', Request)
">
</html>
```

Note that having a `psp:prolog` attribute in the `<html>` tag is also the only way to have variables that are global to the entire page. The SWI Prolog HTTP library provides several predicates that can be used to inspect and manipulate HTTP headers. This behavior may change in future versions, which would require the introduction of a different attribute name.

[2] The `psp`-qualified attributes can be interpreted as overloaded versions of the corresponding html-qualified ones. In fact, the former are always server-side processed and returned to the client as the latter.

[3] Currently, the SWI Prolog SGML library appears not to properly process the Boolean HTML attributes (i.e. attributes without a value), so it is recommended to always set a value for a boolean attribute (specifications suggest to use as value the name of the attribute itself).

4 Web Services

4.1 Calling Prolog remotely

PSP provides the required predicates to send queries and receive responses within Prolog across the web, which is provided by the integration between PSP and JSON-based bidirectional communication via HTTP POST. The PSP web services exchange simple messages containing a single parameter storing the plain Prolog term. Users interested in the translation between Prolog and JSON can take a look to SWI Prolog pengines (http://pengines.swi-prolog.org/). The present implementation is based on JSON-RPC 2.0 (http://jsonrpc.org/), which is much simpler and less verbose than XML. As an illustration, a Prolog call is translated to:

```
{
  “jsonrpc” : “2.0”,
  “method” : “prolog”,
  “params” : [“...”],
  “id” : “{Prolog Session ID}”
}
```

where `...` is the string representation of the goal. Similarly, the response will be:

```
{
   “jsonrpc” : “2.0”,
   “result” : [“...”] | null,
   “error” : null | [“code” : -32000, “message” : “{Prolog Error Term}”],
   “id” : “{Prolog Session ID}”
}
```

where `...` is the string representation of the solutions to the goal.

At the present stage, it is required that the client is able to generate a Prolog call and examine a Prolog term as the reply. A request is normally sent using the HTTP POST method on a specific PSP page. The request is handled regardless of the content of the PSP page. However, the latter can be used to define predicates useful to handling of the request (e.g., authentication predicates) that are eventually called by the Prolog goal sent through JSON-RPC.

4.2 Library predicates

When the client machine uses PSP, several predicates can be employed to invoke a web services located on a remote server. These predicates are essentially the HTTP version of the so-called "all-solutions" Prolog predicates.

http_call(+URL:Goal)

Invoke `Goal` as a goal on the web services located at `URL`, which must be a PSP page. Note that making messages traveling back and forth between client and server machines would be highly inefficient. Therefore, `http_call/1` determines all solutions (see `http_findall/3` below) to `Goal` and then backtracks successively.

http_findall(+Template, +URL:Goal, -Bag)

The `http_findall/3` predicate creates a list of all instantiations, as defined in `Template`, which satisfy `Goal` on successive backtracking, and unifies the result with `Bag`. If there are no solutions to `Goal`, then `Bag` is unified with the empty list `[]`. The recipient HTTP address of the call is specified using module qualification. The following example illustrates how to send and receive data using PSP (assuming that the address where the web service resides is `http://localhost/ws.psp`):

```
% Note that URL appears as the leftmost term in module qualification
% Parentheses could have been omitted as (:)/2 is left-associative

my_webservice(Bag) :-
  http_findall(
    X,
    'http://localhost/ws.psp':(lists:member(X, [1,2,2,3])),
    Bag
  ).

% Bag = [1,2,2,3]
```

http_bagof(+Template, +URL:Goal, -Bag)

The `http_bagof/3` predicate unifies `Bag` with all alternatives of `Template`. The construct `+Var^Goal` can be used to neglect binding of `Var` in `Goal`. Like the built-in `bagof/3` Prolog predicate, `http_bagof/3` fails if Goal has no solutions. The behavior of `http_bagof/3` together with the use of the `^` operator is illustrated in the following example:

```
my_webservice(Bag) :-
  http_bagof(
    Y,
    'http://localhost/ws.psp':(X^(member(X, [1,2,2,3]), Y is 2*X)),
    Bag
  ).

% Bag = [2,4,4,6]  (binding of X has been ignored)
```

http_setof(+Template, +URL:Goal, -Bag)

The `http_setof/3` predicate is equivalent to `http_bagof/3`, but `Bag` contains no duplicates and is sorted using the `sort/2` Prolog predicate. Consider the following example:

```
my_webservice(Bag) :-
  http_setof(
    Y,
    'http://localhost/ws.psp':(X^(member(X, [1,2,2,3]), Y is 2*X)),
    Bag
  ).

% Bag = [2,4,6]   (duplicate 4 has been removed)
```

4.3 A simple example of Web Services utilization

Below is an example of one PSP page performing a RPC request (client-side):

```
<!DOCTYPE html>
<html xmlns:psp="http://www.prologwebservices.org/psp">
  <body>
    <div psp:prolog="
      member(P, [permission1, permission2]),
      http_setof(User,
        'http://localhost/ws.psp':(
          Permission^Password^(
            user(User, Password),
            permission(User, Permission)
            )
          ), List
        )
      ">
      Permission # <span psp:prolog="write(Permission)"></span>
      is granted to <b psp:prolog="length(Users, N), write(N)"></b> users:<br>
      <span psp:prolog="select(User, Users, _), write(User)"><br></span>
    </div>
  </body>
</html>
```

An example for the PSP page taking care of the RPC request (server-side) is:

```
<!DOCTYPE html>
<html xmlns:psp="http://www.prologwebservices.org/psp"
      psp:prolog="ensure_loaded('ws_database.pl')">
</html>
```

Notice that any output given during the processing of the RPC request on the remote server is suppressed. Finally, the server database file may contain the definitions for `user/2` and `permission/2` as:

```
% ws_database.pl

user(userid1, pwd1).
user(userid2, pwd2).
user(userid3, pwd3).
permission(userid1, permission1).
permission(userid1, permission2).
permission(userid2, permission2).
permission(userid3, permission1).
```

When the RPC request is sent directly from PSP, as in the examples above, the `Reply` term is unified either with a (possibly empty) list of solutions or the first error term raised during the call. Thus, it depends on the PSP page that is handling the request to properly process the reply. For RPC requests sent from other environments (e.g., from PHP) any error is found in the "message" field of the JSON reply. Currently, the message is identical to the Prolog error term, except that it is a string and hence it does require proper parsing.

Contact

For information, feedback or suggestions please send an email to info@prologwebservices.com.

User requests of general interest can also be posted at comp.lang.prolog or the SWI Prolog forum.