BinProlog 11.x Professional Edition Internet Programming Guide

Paul Tarau

BinNet Corp.
WWW: http://www.binnetcorp.com

October 17, 2006

1 Introduction

BinProlog has evolved during the last years towards a sophisticated Logic Programming based Internet application building tool. This document covers, through examples, a number of typical Internet programming patterns.

2 Installing the BinNet Internet Toolkit

The default assumptions hold for a Windows machine, but adapting to Unix is mostly changing some path information.

Just uncompress the file bp_inet.zip, containing the toolkits you have ordered to a directory bp_inet. The resulting directories are as follows:

- **bin**: BinProlog binary executable (i.e. bp.exe)
- **cgi**: CGI scripts
- **library**: shared files, needed for various components
- **client**: HTTP Client Toolkit
- **server**: HTTP Server Toolkit
- **spider**: BinNet Internet Search Engine
- **tests**: Tests - trying out various components

3 The BinNet Internet Client Library

CGI programs are executed on the server upon clicking on special links. They are used from froms for electronic shopping to online conference organizing.

3.1 Programming CGIs: the easy way

To run the CGI scripts securely (assuming http://localhost as the base address) configure your Web server to have bp_inet/bin executable only and bp_inet/cgi readable only.

The directory structure is configured in such a way, that if you add your own new scripts let's say somewhere in bp_inet/myscripts and follow the same include convention as scripts in directory **cgi**, everything will run fine.

BinProlog has some built-in facilities which make CGI-programming easy. The following shows a simple Web page access counter.

```
main:-header,inc(X),show_counter(X).

header:-
  write('220 ok'),nl,
  write('content-type: text/html'),nl,nl.

show_counter(X):-write(counter(X)),write('.'),nl.

inc(X):-
  F='cstate.pro',
```

```
  ( see_or_fail(F)-> see(F),read(counter(X)),seen
  ; X=0
  ),X1 is X+1,
  tell(F),show_counter(X1),told.
```

To install a similar CGI script at your site, put the **bp** executable in directory **cgi-bin**, together with the program and call it from a HTML page as follows:

```
<TITLE>
 BinProlog CGI counter
</TITLE>
<HTML>
<BODY>
Try a BinProlog based
<A HREF=
  "/bp_inet/bin/bp.exe?$/bp_inet/cgi/counter.pro">
CGI Web-counter!        ^^^^^^^^^^^^
</A>                   ABSOLUTE PATH!
</BODY>
</HTML>
```

You can try it out by following the demo link at:

`http://www.binnetcorp.com/BinProlog`

Look for LogiMOO [19, 1], a more advanced BinProlog based Internet application, now also running directly under Netscape, at:

`http://clement.info.umoncton.ca/~tarau/logimoo`

3.1.1 A BinProlog Query Evaluator

A more complex script (see files cgi/query.pro and cgi/query.html) needs to be used to pass information from a HTML file to the Prolog script. BinProlog's cgi_lib.pro library uses POST method for sending information to the server. This means, that after some basic header exchange, the client will read and parse from the standard input field names/field data pairs. This is achieved by components in cgi_lib.pro, the script itself matches the expected parameters and proceeds with its specific tasks.

```
CGI script for querying BinProlog over the net
% Copyright (C) BinNet Corp. 1998

:-['../library/cgi_lib'].
:-['../library/http_tools'].

main:-run_cgi(5,body).

% POST method body
body:-
  (get_cgi_input(Alist)->true;test,fail),
  Alist=
    [ login=Ls,
      passwd=Ps,
      email=Ms,
      home=Hs,
```

```
      query=Qs,
      editor=Es
   ]
->
  ( run_it(Ls,Ps,Ms,Hs,Qs,Es)->true
  ; write('Error in query: '),write_chars(Qs),nl
  )
; ( write('Non matching fields in form'),
    fail
  ; nl
  ).
```

3.1.2 A multi-threaded benchmark

This CGI offers some control over the number of threads to be used as well as the number of iteration and data size.

3.1.3 A Joke Server

This CGI implements a joke server together with an operation to add new jokes. The database is implemented as a Prolog file. Additions occur simply bay concatenation to the end of the file.

3.1.4 A Guestbook script

This simple script reads information provided by visitors of a Web page and appends it in the form of Prolog facts at the end of a file.

3.2 A multi-threaded stock market simulator script

This simple, hand-built form (most other forms are built with HTML generators like Netscape's Composer) calls the script **smarket.pro**. The script was originally written for Jinni - our Java based interpreter - and actually runs under Jinni as well.

It simulaties a stock ticker and two trading agents - performing some typical transactions, triggered by price changes. This is an example of complex, event driven CGI script, with BinProlog's Linda blackboard used for multi-agent synchronization.

```
<html>

<head>
<title>BinProlog CGI form script</title>
</head>

<body>

<form method="post"
 target="answer"
 action="/bp_inet/bin/bp.exe?$/bp_inet/cgi/smarket.pro">

<b>Simulation time in seconds:</b>
   <input name="time" type="text" size="2" value="3">
<b>Variation:</b>
   <input name="variation" type="text" size="4" value="0.09">
```

```
<b>Direction:</b>
   <input name="direction" size="4" type="text" value="0.001">
<p>
Feel free to add your comments related to this program and BinProlog.
</p>
<textarea align="top" name="comment" rows=3 cols=60>
</textarea>
<input align="bottom" type="submit" value="Submit">
</form>

</body>
</html>
```

3.3 The BinNet HTTP Client

The BinNet HTTP Client library supports HTTP 1.x processing directly in Prolog, using BinProlog's portable socket operations. It allows a BinProlog program to get HTML or ASCII text pages from a Web server and "datamine" for links found in the page. It also allows consulting BinProlog files directly from Web pages. This library, easy to extend and reuse is provided in source form. It interoperates with the BinNet HTTP server and the BinNet Programmable Internet Search Engine (spider).

- **reconsult_url(AtomicURL)**: reconsults a Prolog file from a Web server i.e. replaces each predicate with a new definition found there, asserted to the current database
- **consult_url(AtomicURL)**: consults a Prolog file from a Web server i.e. asserts each to the current database
- **http2line(AtomicURL,Line)**: Opens a URL, then gets a stream of lines of chars from the WWW server. It will backtrack over them. At the end, it fails. A query, for

  ```
  AtomicURL='http://www.binnecorp.com/test.txt'
  ```

  pointing to the a file containing the lines:

  ```
  This
  is
  a test.
  ```

  will work as follow:

  ```
  ?-http2line('http://www.binnecorp.com/test.txt',Line).
  ....
  ....  % some header lines
  ....
  Line=[84,104,105,115];

  Line=[105,115];

  Line=[97,32,116,101,115,116,46];
  no
  ```

This acts like if you used

```
?-File=[   "This","is","a test."],member(Line,File).
```

Use name/2 to convert each list of characters returned by Line to an atom. Type help(read), help(write), help(chars) for hints on various operations you can perform on each line.

- **http2line(AtomicURL,Cs)**: Same as http2line, but skips header lines.
- **http2content_line(AtomicURL,Cs)**: extracts the content of html or ascii text page, Prolog code in particular, to a list of charaters, headers excluded.
- **http2link(AtomicURL,Link)**: Extracts links to other URLs, found in a HTML page, given through its URL.

  Try:

  ```
  ?-http2link('http://www.binnetcorp.com',Link).
  ```

- **http2char(AtomicURL,C)**: extracts content of html or ascii text page, Prolog code in particular, one character at at time, header excluded. Backtracks over each character, fails at end of the stream.

  Try:

  ```
  ?-http2char('http://www.binnetcorp.com/test.txt',C),put_code(C),fail;nl.
  ```

- **http2chars(AtomicURL,Cs)**: extracts content of html or ascii text page, Prolog code in particular, to a list of characters.
- **http2clause(AtomicURL,Clause)**: Extract on clause at a time, from a URL pointing to a Prolog file. Backtracks over them, fails at the end.
- **http2db(AtomicURL)**: consults a URL containing Prolog code to current database
- **http2db(AtomicURL,Db)**: consults a URL containing Prolog code to a given database
- **http2db_replace(AtomicURL)**: reconsults a URL containing Prolog code to current database, while replacing existing predicates having the same signature
- **assume_url(AtomicURL)**: assumes with assumei/1, all Prolog clauses at URL

4 The BinNet Internet Search Engine

The search engine is composed of a kernel library (file http_spider.pl) and a set of scripts (http_spider_scripts.pl). The spider remembers links followed. Links can be seen, after executions with `show_links`, together with the depth where they are found.

The main interface to the kernel is provided through the predicate spider/4.

```
spider(K,AtomicRoot,Link,Path):

% Walks through links up to depth K
% starting from AtomicRoot of the form 'http://...'
% Returns each Link and Path (a list of canonical links) used to get there.
```

Keyword search scripts are programmed by extracting list of words, found on each line of the HTML or text file to which the link refers, with `http2words(Link,Ws)`.

Good/bad link classification scripts are programmed using `is_good_link(AtomicURL,YesNoMaybe)` which returns yes/no/maybe depending on valid syntax and actual responsiveness of the URL to which the link points on the Internet.

To scripts limited to links local to the Web site given as RootURL are programmed by overriding internal components of the kernel spider using intuitionistic implication (assumptions), as in:

```
/*
% Internal Spider: moves only inside the domain of TopURL
*/
internal_spider(Depth,TopURL,HarvestedLink):-
  url_filter(internal_only)=>>spider(Depth,TopURL,HarvestedLink).
```

A number of script examples are given in file `http_spider_script.pl`. The features described can be combined independently, and it is also possible to start the spider with a meta-search root, for instance, a Lycos or Yahoo query, which is further explored as if it were an ordinary html file.

Various tests for the spider as well as of its component are available in file `bp_inet/test/http_test.pl`

5 The BinNet Web Server

The server implements a subset of the HTTP 1.x protocol (handling of text/html files together with efficent execution of Prolog server side includes (SSIs) - scripts running much more efficiently that dynamicaly loaded CGI scripts - as they are part of the server. Other requests are redirec by the server to a standard server (assumed on the same machine, at port 80). Our server itself, listens by default on port 8080.

To try out the server, together with a server side include script handling facility, type

```
bp http_query
```

Assuming that the BinProlog Internet tools have been uncompressed in something like `c:\bp_inet`, just open with Netscape the following link:

```
http://localhost:8080/bp_inet/cgi/http_query.html
```

Click on the **submit** button to see the server interacting with the Netscape client, both for fetching a file and running a CGI script-like *server side include - (SSI)* program (based on code in ssi_lib.pl and http_query.pl).

This demo form can be easily adapted to remotely administer the server, with appropriate password protected queries.

To launch the server only, on default port 8080, without the SSI facilities, type in a command window:

```
bp http_server
```

then, type at the prompt:

```
?- http_server.
```

To create distributable bytecode application type:

```
bp

?-fcompile(http_query).
```

Then, run the bytcode application with

```
bp http_query.wam
```

Note that on platforms where multi-threading is available, the server spawns new threads for each client. The script http_query.pl gives an example of a truly *multi-user* Prolog process: users entering different logins in the Web form will be assigned different clause databases as well. Threads/engines spawned to support user requests are recycled after each execution.

6 BinProlog Networking

6.1 Client/server programming in BinProlog: a first look

To try this out, open 3 windows, start BinProlog in each, then type:

```
?-run_server.             % in the first window
?-out(hello(your_name)).  % in the second window
?-all(X,Xs).              % in the third window
```

You will see that they are connected! The following sections will show how this kind of communication is achieved between your computer and, if you are 'wired', other computers over the net.

7 Mobile Code

The latest version of BinProlog supports two simple **move/0** and **return/0** operations which transport computation to the server and back. The client simply waits until computation completes, when bindings for the first solution are propagated back:

```
Window 1: a mobile thread

?-there,move,println(on_server),member(X,[1,2,3]),
        return,println(back).
back
X=1;
no.

Window 2: a server

?-trust.
on_server
```

In case return is absent, computation proceeds to the end of the transported continuation. Note that mobile computation is more expressive and more efficient than remote predicate calls as such. Basically, it *moves once*, and executes on the server *all future computations* of the current AND branch until a return instruction is hit, when it takes the remaining continuation and comes back. This can be seen by comparing real time execution speed for:

```
?-there,for(I,1,1000),run(println(I)),fail.
```

```
?-there,move,for(I,1,1000),println(I),fail.
```

While the first query uses `run/1` each time to send a remote task to the server, the second moves once the full computation to the server where it executes without further requiring network communications. Note that the `move/0, return/0` pair cut nondeterminism for the transported segment of the current continuation. This avoids having to transport state of the choice-point stack as well as implementation complexity of multiple answer returns and tedious distributed backtracking synchronization. Surprisingly, this is not a strong limitation, as the programmer can simply use something like:

```
?-there,move,findall(X,for(I,1,1000),Xs),return,member(X,Xs).
```

to emulate (finite!) nondeterministic remote execution, by collecting all solutions at the remote side and exploring them through (much more efficient) local backtracking after returning.

8 Threads

Windows 95/98/NT versions of BinProlog also support multithreaded execution. Type help(thread), then use info/1 for more information on thread related operations.

In fact, for most programs just using

```
bg(Goal)
```

```
and
```

```
synchronize(Statement)
```

are all you need to get started. Take a look at the StockMarket Web-based demo at

```
http://www.binnetcorp.com
```

for a more interesting example using multiple threads.

9 The Web of BinProlog users

You can get connected to users of BinProlog over the Internet by opening two windows and starting BinProlog in them (this should work on an Internet connected Unix or Windows 95/NT machine).

In the first window type:

```
?-host('your.full.internet.address')=>run_server.
```

In the second window type:

```
?-chat.
TARGET SERVERS ALIVE ON THE NET:
.............................
.............................
^D on Unix or ^Z on PCs to end
>
> TYPE IN YOUR MESSAGES HERE!!!
>
```

Whatever you type in the second window (as well as whatever comes form other users over the net) can be seen in the first one.

Starting `run_server` is a fairly secure operation, the (restricted) server will only accept messages to be displayed and Linda operations.

You can personalize your character (the login name appears on the screen of other users preceding what you say) with:

```
?-login(joe)=>password(evrika)=>chat.
```

If you cannot connect to a master server or you want to keep the chat local to your computer you can start a server with

```
?-run_server.
```

one servant for each user with:

```
?-run_servant.
```

To participate to the local chat do something like:

```
?-login(joe)=>talk.
```

This will show what you say with prompt `joe` on the server and all its 'satellite' servants.

9.1 High-level socket operations

We have tried to hide socket operation as well as possible from the Prolog programmer, usually interested in elegant, high-level programming. Some inspiration from Java's approach to socket programming is gratefully acknowledged.

- new_client(Host,Port,ClientSocket) creates ClientSocket connected to Host (a remote computer's Internet address, by default localhost), Port (a Unix port, by default 9001)
- new_server(Port,ServerSocket) opens on Port (by default 9001) a new ServerSocket
- new_service(ServerSocket, Timeout, ServiceSocket) creates a ServiceSocket answering ServerSocket within Timeout seconds
- close_socket(Socket) closes a server, service or client socket
- sock_read(Socket,CharList) reads a list of characters from a socket. Internally, the length in bytes is sent as 32 bit network-ordered word is read, then the bytes which are converted to a Prolog list of character codes.

- sock_readln(Socket,CharList) reads a list of characters from a socket, allowing the programmer to internalize it with name/2. Note that internalizing is better to be avoided as it fills internal tables which are garbage collected only when restart/0 resets them together with all dynamic data. However, term_chars/2 allows extracting a term from such a list of chars without internalizing it.
- sock_write writes a list of characters to a socket, first its length in bytes then as a sequence of bytes.
- sock_writeln writes a list of characters to a socket, terminated with a newline character. Both sock_read(ln) and sock_write(ln) are intended to be used in 'connectionless' ask/answer style dialog between a client and a server (see the Linda package in file extra.pl built on top of them as an example of use.
- sleep(N) waits N seconds without using CPU resources
- fork(Pid) starts a new completely IO-less background Unix process (child) with stdio, stdout and stderr redirected to /dev/null and returns the child's Pid to the parent and Pid=0 to the child

9.2 Client/server programming: Linda operations

- run_server/0 runs a foreground Linda server
- fork_server/0 runs background Linda server
- fork_server/1 forks Linda server and returns its Pid
- stop_server/0 stops Linda server
- stop_server/1 stops Linda server with given password
- ask_server/2 sends a query and gets an answer from Linda server
- server_info/1 gets language, host and port info from Linda server
- serve_answer/2 is the basic query/answer 'interactor' on Linda server. It is applied with call/N to to an input from a client, to produce an output to be sent back (see file extra.pl for details).
- out/1 puts a term on Linda server
- in/1 waits to remove a term from Linda server
- all/2 gets the list of terms matching arg 1 from Linda server
- rd/1 reads a terms matching arg 1 from Linda server
- cin/1 tries to remove a term from Linda server
- default_host/1 returns default host for Linda server

- default_port/1 returns default port for Linda server. Use something like host('eve.info.umoncton.ca') ⇒ port(8888) ⇒ LindaOperation to 'redirect' a Linda operation to work with the specified host/port. This use of intuitionistic implication ⇒ avoids passing useless parameters to deep calls. Therefore, multiple versions of the same predicate with explicit host/port arguments are not needed for the various Linda operations. Once the host/port are assumed using ⇒, their values are used inside the proof (execution) of LindaOperation. As ⇒ is scoped and backtrackable, no unpleasant side-effects remain after its use, allowing seamless 'multiple-server' dialog.

Some other operations fix the current host/port/remote code file on the server, making reference to them simpler in case we are repeatedly interacting with the same remote server.

- set_host/1 sets IP address or name of server host we want to talk to
- set_port/1 sets port number of the server we want to talk to
- set_code/1 sets the name of the default file/database on remote server, used for instance in remote code fetching (see rload/1).

10 Running remote code

On an Intranet of trusted users and computers, or in different windows of your unconnected PC or workstation you might want to experiment with a server allowing arbitrary Prolog command execution. You can start a local remote predicate call server by typing in a first window:

```
?-run_unrestricted_server.
```

BinProlog's convention is that if the name returned by default_host/1 is different from `localhost` it assumes you want to get connected and interoperate through customized servers with other computers using BinProlog.

To override this default setting you can use either `set_host('my.full.internet.address')` or temporarily override your default host by assuming it as a `host/1` fact with BinProlog's intuitionistic assumption. Such assumptions are scoped and forgotten on backtracking,

With

```
?-server_host('my.full.internet.address')=>run_server.
```

By default such servers are registered on a master server at BinProlog's home, see `default_master_server/1` for its host/port information.

You can try out your server in a client window with

```
?-remote_run((write(hello),nl)).
```

Note that registered servers are in principle accessible to other users and therefore *not* fully secure, unless you use one of the following two methods to implement security.

10.1 A first approach to security: servers with restricted interactors

Here is the code of the chat server:

```
chat_server_interactor(mes(From,Cs),Answer):-show_mes(From,Cs,Answer).
chat_server_interactor(ping(T),R):-term_server_interactor(ping(T),R).
chat_server_interactor(run(_),no).
chat_server_interactor(run(_,_),no).

chat_server:-
  server_interactor(chat_server_interactor)=>>
  run_server.
```

Basically, by overriding the default `server_interactor` with our `chat_server_interactor` we tell to `chat_server` that only commands matching known, secure operations has to be performed on behalf of remote users.

This might look very simple but some of BinProlog's key features (intuitionistic implication and higher-order call/N) are used inside the 'generic' server code (see file extra.pl) to achieve this form of 'configurability'.

Here is some *interactor* code (which might change in the future) for two other 'secure' servers.

The first one (BinProlog's default) is a Linda+chat server (to be started with `run_server/0`). It offers an good combination of security and convenience: only allows modifying its dynamic database remotely through Linda operations and displaying messages. It can be shut down remotely (with password) and checked if alive and although it can be subject to resource attacks by malicious users it can do no harm to your files system or give away information about your computer.

```
term_server_interactor(cin(X),R):-serve_cin(X,R).
term_server_interactor(out(X),R):-serve_out(X,R).
term_server_interactor(cout(X),R):-serve_cout(X,R).
term_server_interactor(rd(X),R):-serve_rd(X,R).
term_server_interactor(all(X),Xs):-serve_facts(X,X,Xs).
term_server_interactor(all(X,G),Xs):-serve_facts(X,G,Xs).
term_server_interactor(id(R),R):-this_id(R).
term_server_interactor(ping(T),T):-ctime(T).
term_server_interactor(stop(W),yes):-assumel(server_done(W)).
term_server_interactor(halt(X),R):-serve_halt(X,R).
term_server_interactor(add_servant(X),R):-serve_add_servant(X,R).
term_server_interactor(mes(From,Cs),Answer):-
  show_mes(From,Cs,Answer),
  forward_to_servants(mes(From,Cs)).
term_server_interactor(proxy(H,P,Q),R):-
  ask_a_server(H,P,Q,A)->R=A;R=no.
term_server_interactor(in(X),R):-serve_cin(X,R). % $$thish should block!!!
```

Note that this interactor even supports proxy forwarding to a give host/port as well as forwarding to passive pseudo-servers (called 'servants', which work by watching for given patterns on real servers and reacting back) and in particular to the Java applets in related package LindaInteractor.tar.gz. Note that the secure execution of forwarded queries are the responsibility of the proxy target.

10.2 A second approach to security: starting an Intranet specific master server

Keeping one's host/port information secret from other users can be achieved by starting a master server local to a secure physical or virtual Intranet.

For users behind a firewall, this might actually be the only way to try out these operations as the default master server might be unreachable.

To start your own master server type in a window something like:

```
?- master_server('my.secure.local.computer',7788)=>run_master_server.
```

Use a fact

```
master_server('my.secure.local.computer',7788).
```

Type in or have your code execute:

```
set_master_server('my.secure.local.computer',7788)
```

in both your client and server programs intended to be managed by your own master server.

A useful application to this is to build a local chat line not seen by users from the outside.

Progressively, commercial quality security will be built in BinProlog, especially if non-sensical export restrictions in US and Canada will be removed on strong cryptography.

To keep the workload of the master server minimal, only when an error is detected by a client, the master_server is asked to refresh its information and possibly remove dead servers from its database.

10.3 Towards a Web of interacting secure mobile agents

The MOO metaphor combined with the idea of 'negotiation' between the agent's intentions and the 'structure of the world', represented as a set of Linda blackboards storing state information on servers connected over the the Internet allows a simple and secure remote execution mechanism through specialized server-side interpreters.

Implementation of arbitrary remote execution is easy in a Linda + Prolog system due to Prolog's metaprogramming abilities. No complex serialization remote procedure/method call packages are needed. In BinProlog (starting with free version 6.25) code fetched lazily over the network is cached in a local database and then dynamically recompiled on the fly if usage statistics indicate that it is not volatile and it is heavily used locally.

As an example, high performance file-transfer (actually known to be faster than well-known `http` as `ftp` protocols) is implemented by orchestrating Prolog based `remote_run` control operations which are used to trigger direct full speed transfers entirely left to optimized C built-ins like `sock2file/2`:

```
fget(RemoteFile,LocalFile):-
   remote_run(RF,fopen(RemoteFile,'rb',RF)),
   fopen(LocalFile,'wb',LF),
   term_chars(to_sock(RF),Cmd),
   new_client(Socket),
     sock_write(Socket,Cmd),
     socket(Socket)=>from_sock(LF,_),
     fclose(LF),
```

```
      sock_read(Socket,_),
    close_socket(Socket),
    remote_run(fclose(RF)).

from_sock(F,yes):-
    assumed(socket(S)),
    sock2file(S,F),!.
```

Once the basic Linda protocol is in place and Prolog terms are sent through sockets, a command filtering server loop simply listens and executes the set of 'allowed' commands.

Despite Prolog's lack of object oriented features we have implemented code reuse with intuitionistic assumptions.

For instance, to iterate over the set of servers forming the receiving end of our 'Web of MOOs', after retrieving the list form a 'master servers' which constantly monitors them making sure that the list reflects login/logout information, we simply override host/1 and port/1 with intuitionistic implication:

```
ask_all_servers(ListOfServers,Question):-
  member(server_id(_,H,P),ListOfServers),
  host(H)=>port(P)=>ask_a_server(Question,_),
  fail.
ask_all_servers(_,_).
```

To specialize a generic server into either a master server or a secure 'chat-only' server which merges messages from BinProlog users world-wide we simply override the filtering step in the generic server's main interpreter loop.

Implementing agents 'roaming over' a set of servers is a simple and efficient high-level operation. First, we get the list of servers from the master server. Then we iterate through the appropriate remote-call negotiation with each site. Our agent's behavior is limited through security and resource limitations of participating interpreters, each having their own command filtering policies.

In particular, on a chat-only server our roaming agent can only display a message. If the local interpreter allows gathering user information than our agent can collect it. If the local interpreter allows question/answering our agent will actually interact with the human user through the local server window.

Such mobile agents are seen as 'connection brokers' between participating independent server/client sites. For instance if two sites are willing to have a private conversation or code exchange they can do so by simply using the server/port/password information our agent can help them to exchange.

Note that full metaprogramming combined with source-level mobile-code have the potential of better security than byte-code level verification as it happens in Java, as meaningful analysis and verification of program properties is possible.

10.4 An overview of some remote operations

The following operations

```
host(Other_machine)=>rload(File).

host(Other_machine)=>code(File)=>TopGoal.

host(Other_machine)=>fetch_remote_operators.
```

allow fetching remote files/operator definitions `rload/1, fetch_remote_operators/0` or on-demand fetching of a predicate at a time from host/code file during execution of TopGoal.

```
host(Other_machine)=>remote_run(RemoteGoal).
```

```
host(Other_machine)=>remote_run(Answer,RemoteGoal).
```

allow remote predicate calls without/with returned answer on the calling site.

```
host(Other_machine)=>rsh(ShellCommand,CharListAnswer).
```

```
host(Other_machine)=>rsh(ShellCommand).
```

allows using BinProlog to remotely execute shell commands on a remote machine and collect/print the answers at the calling site. In particular it allows what Microsoft never could/wanted to build in W95 or NT: remote shell commands. In particular, by starting a BinProlog server on a PC with W95/NT it is possible to do meaningful work on it remotely from another W95/NT PC or Unix box. Note that on machines not supporting pipes rsh is approximated with rexec which redirects execution to a file, gets it back and shows it to the user. Note also that

```
?-run_unrestricted_server.
```

should be used on the remote computer to allow these commands.

11 Interaction with Java Applets.

BinProlog starting from version 5.40 communicates with our recently released Java based *Linda Interactors*[1] special purpose trimmed down pure Prolog engines written in Java which support the same unification based Linda protocol as BinProlog. The natural extension was to allow Java applets to participate to the rest of our 'peer-to-peer' network of BinProlog interactors. As creating a server component within a Java applet is impossible due to Java's (ultra)-conservative security policies we have simply written a receiving-end *servant* close to our example in subsection **??**, which relies on a proxy server on the site where the applet originates from, for seamless integration in our world of peer-to-peer interactors.

Here is the code for a more realistic servant, multiplexed among multiple servers and usable inside a Java applet.

```
run_servant:-
  default_server_interactor(Interactor),
  this_id(ID),
  out(servant_id(ID)), % registers this servant
  repeat,
    in(query(ServerId,Query)), % waits for a query
    (call(Interactor,Query,Reply)->Answer=Reply;Answer=no),
    % sends back an answer, if needed
    ( ServerId=[]->true % no reply sent to anonymous servers
    ; out(answer(ServerId,Answer))
    ),
```

[1] available at http://clement.info.umoncton/BinProlog/LindaInteractor.tar.gz

```
  functor(Query,stop,_),  % stops when required
  !,
  in(servant_id(ID)).
```

Note the presence of an overridable client-side interactor, allowing the generic servant code to be easily reused/specialized. Multiplexing is achieved by having each server's in/1 and out/1 data marked with unique `servant_id/1` records.

To integrate multiple Java applet based clients in our 'Web of Worlds', we use a more complex forwarding server, also available as equivalent Java code, to be run as a BinProlog and/or Java daemon[2] on the same machine as the HTTP server the applet comes from.

```
run_forwarding_server:-
  server_interactor(forwarding_server_interactor)=>>
  run_server.

forwarding_server_interactor(mes(From,Cs),R):-!,R=yes,
   forward_to_servants(mes(From,Cs)).
forwarding_server_interactor(Q,A):-
   term_server_interactor(Q,A).

forward_to_servants(Query):-
   clause(servant_id(_),true),
   assert(query([],Query)),
   fail.
forward_to_servants(_).
```

Note that the forwarding server has the ability to interact with multiple servants, in particular with multiple Java applets. Starting with version 6.25 this forwarding ability is built in default BinProlog servers started with `run_server`. Try out the example applet in the corresponding version of the related Java package Jinn available from

`http://www.cs.unt.edu/~tarau/netjinni/Jinni.html`

12 Communication behind a firewall

The simplest way to communicate behind a firewall or in case of dynamic IP assignement is to use a servant instead of a server. The servant will watch a real server with `in/1` operations to perform commands assigned to it. To attach the servant to a server with full Internet access use the server at BinProlog's home i.e. something like:

```
?-host('cs.unt.edu')=>port(7001)=>run_servant.
```

or your own server on the firewall, if you have an account on it.

13 Generating VRML through BinProlog CGIs

The following is a quick description of a VRML generator based on my ILPS'97 tutorial slides. Try it out its CGI version by following a link from

`http://www.cs.unt.edu/~tarau`

[2]Alternatively, BinProlog can be embedded as a server side include in an Apache server (written in C) while the equivalent Java code is easily embeddable in the Java based Jigsaw HTTP server.

The CGI script dynamcally generate a simple animated VRML 2.0 landscape with some randomly changing colors.

13.1 VRML and Prolog: the mappings

- hierarchical space representation + event propagation for animation = VRML 2.0
- the need for theory: deep isomorphisms between the structure of the *SPACE* and the structure of *TRUTH*
- declarative SPACE representation and TRUTH representation share the same difficulties w.r.t TIME and change
- cross-fertilization?
- as usual: shallow dissimilarities can get in the way: let's show some shallow similarities `:-)`

13.2 A syntactical mapping: VRML "is" Prolog

- VRML: hierarchical decomposition of the space in regions
- Prolog: terms as trees
- concrete syntaxe: closer to Life or Feature Structures than to Prolog
  - all arguments are named, not positional
  - VRML 2.0 ROUTES are structurally similar to DCGs: they are used to *thread* together streams of change

13.3 Syntax mapping of some key idioms:

```
Prolog         VRML
-------------------------
  @{  } ===>   {  }
  @[  ] ===>   [  ]
  f(a,b,c) ==> f a b c
  a is b   ==> a 'IS' b
  a=b      ==> a b
------------------------
```

key idea: let's allow writting unrestricted VRML code in Prolog!

13.4 Pseudo VRML in Prolog: a PROTO

```
proto anAppearance @[
   exposedField('SFColor')=color(1,0,0),
   exposedField('MFString')=texture@[]
]
@{
   'Appearance' @{
       material='Material' @{
```

```
        diffuseColor is color
      },
      texture='ImageTexture' @{
        url is texture
      }
    }
}.
```

13.5 How it looks in VRML?

```
#VRML V2.0 utf8

PROTO anAppearance
[
  exposedField SFColor color 1 0 0 ,
  exposedField MFString texture []
] {
  Appearance { material
    Material { diffuseColor IS color }
    texture ImageTexture {
      url IS texture
    }
  }
}
```

13.6 Some Prolog Macros

```
shape(Geometry):-
  % based on scoped assumptions (=>)
  default_color(Color),
  toVrml(
     aShape @{
       geometry(Geometry@{}),
       Color
     }
  ).

sphere:- shape('Sphere').
cone:- shape('Cone').
cylinder:- shape('Cylinder').
box:- shape('Box').
```

13.7 Using Prolog macros

```
group @{
  children @[
    `transform(
       translation(0,10,4),scale(2,2,2),rotation(0,0,0,0),
       [`sphere, `cone]
     ),
```

```
    `transform(
       translation(5,0,0),scale(1,3,6),rotation(0,1,0,1.5),
       [`box]
     )
  ]
}.
```

13.8 From the Translator: (with Assumption Grammars)

```
toVrml(X):-number(X),!,#X.
toVrml(X):-atomic(X),!,#X.
toVrml(A@B):-!,#indent(=),toVrml(A),toVrml(B).
toVrml(A=B):-!,toVrml(A),toVrml(B).
toVrml(A is B):-!,#indent(=),toVrml(A),#'IS',toVrml(B).
toVrml({X}):-!,#'{',#indent(+),toVrml(X),#indent(-),#'}'.
toVrml(`X):-!,X.
toVrml(X):-is_list(X),!,
  #'[',#indent(+),vrml_list(X),#indent(-),#']'.
toVrml(X):-is_conj(X),!,vrml_conj(X).
toVrml(T):-compound(T),#indent(=),vrml_compound(T).
```

13.9 What's generated?

```
Group {
  children [
    Transform {
      translation 0 10 4 scale 2 2 2 rotation 0 0 0 0
      children [ # aShape is a VRML 2.0 PROTO!
        aShape { geometry Sphere{} color 0.7 0.8 0.8
        } ,
        aShape { geometry Cone{} color 0.6 0.3 0.9
        }
      ]
    } , .....
  ]
}
```

13.10 Some cross-fertilization opportunities:

- trivial but useful: Prolog is a great VRML syntax checker!
- Prolog as a powerful macro language/static optimizer for VRML
- Prolog as an AI *planner* for complex intelligent avatar movement in VRML
- Prolog as a scripting language: Java and Javascript have little structural affinity with VRML which is about TERMS: trees of 3D object groupings!
- completely powerless: Javascript
- unnecessarily complex: Java+EAI

14 Inspecting some BinProlog internals

You can generate a kind of intermediate WAM-assembler by

```
?- compile(asm,[file1,file2,...],'huge_file.asm').
```

A convenient way to see interactively the sequence of program transformations BinProlog is based on is:

```
?- asm.
a-->b,c,d.
^D

DEFINITE:
a(A,B) :-
        b(A,C),
        c(C,D),
        d(D,B).

BINARY:
a(A,B,C) :-
        b(A,D,c(D,E,d(E,B,C))).

WAM-ASSEMBLER:
clause_? a,3
firstarg_? _/0,6
put_structure d/3,var(4-4/11,1/2)
write_variable put,var(5-5/10,1/2)
write_value put,var(2-2/6,2/2)
write_value put,var(3-3/7,2/2)
put_structure c/3,var(3-8/14,1/2)
write_variable put,var(2-9/13,1/2)
write_value put,var(5-5/10,2/2)
write_value put,var(4-4/11,2/2)
execute_? b,3
```

15 LogiMOO

LogiMOO is a BinProlog based virtual world, usable for collaborative work or simply online chat. A simple agent, the Notifier is provided (see file moo.pl in directory LogiMOO).

Still in prototype stage, but working fine.

See the our PAP'96 paper [1] for more info.

Here is an example of session (requires Solaris 2.4 or better) playing with distributed objects (created by "obmaster") and listed by "joe". You'll need some imagination to reconstitute the exact sequence. Better, try it out yourself on a Sparc with Solaris!

```
----------------------------------------------------------------
Window 1: - wizard (see server.pl)

mbp
```

```
?- [server].
.....
?- go.
yes
?- notfier started
wizard says: Hello
?- list.
place(lobby) .
lobby contains wizard .
yes
?- joe says: Hello
?- obmaster says: Hello
?-

-----------------------------------------------------------------
Window 2: - joe: (see joe.pl)

mbp

?- [joe].
...
?- go.
notfier started
joe says: Hello
?- yes
?- obmaster says: Hello
?- list.
place(lobby) .
lobby contains wizard .
lobby contains joe .
lobby contains obmaster .
ob(object,<:,object) .
obs(object,:>,object) .
ob(mammal,<:,object) .
obs(object,:>,mammal) .
ob(mammal,legs,4) .
obs(mammal,>>,legs) .
ob(dog,<:,mammal) .
obs(mammal,:>,dog) .
obs(dog,>>,legs) .
ob(dog,legs,3) .
yes

-----------------------------------------------------------------
Window 3: - obmaster: (see obmaster.pl and included files)

mbp

?- [obmaster].
...
?- go.
notfier started
obmaster says: Hello
```

```
 :-
        object with true.
 :-
        mammal extends object with (legs := 4,true).
 :-
        dog extends mammal with (legs := 3,true).

?- bye.
```

16 Related work

BinProlog related papers can be found following links from

`ftp://clement.info.umoncton.ca/~tarau`

or following links from

`http://www.cs.unt.edu/~tarau`

The reader interested in more information on Internet Programming with BinProlog and its new Java based component is refered to the following papers:
[7, 6, 12, 13, 18, 8, 14, 16] and [15, 3, 17, 4, 20, 1, 2]
Related BinProlog documentation is available at: [11, 9, 10, 5].
Slides from an ILPS'97 tutorials on Internet Programming with BinProlog are available following links from:

`http://www.cs.unt.edu/~tarau`

References

[1] K. De Bosschere, D. Perron, and P. Tarau. LogiMOO: Prolog Technology for Virtual Worlds. In *Proceedings of PAP'96*, pages 51–64, London, Apr. 1996.

[2] K. De Bosschere and P. Tarau. Blackboard-based Extensions in Prolog. *Software — Practice and Experience*, 26(1):49–69, Jan. 1996.

[3] S. Rochefort, V. Dahl, and P. Tarau. Controlling Virtual Worlds through Extensible Natural Language. In *AAAI Symposium on NLP for the WWW*, Stanford University, CA, 1997.

[4] P. Tarau. Logic Programming and Virtual Worlds. In *Proceedings of INAP96*, Tokyo, Nov. 1996. keynote address.

[5] P. Tarau. BinProlog 7.0 Professional Edition: Predicate Cross-Reference Guide . Technical report, BinNet Corp., 1998. Available from http://www.binnetcorp.com/BinProlog.

[6] P. Tarau. Jinni: a Lightweight Java-based Logic Engine for Internet Programming. In K. Sagonas, editor, *Proceedings of JICSLP'98 Implementation of LP languages Workshop*, Manchester, U.K., June 1998. invited talk.

[7] P. Tarau. Towards Inference and Computation Mobility: The Jinni Experiment. In J. Dix and U. Furbach, editors, *Proceedings of JELIA'98, LNAI 1489*, pages 385–390, Dagstuhl, Germany, Oct. 1998. Springer. invited talk.

[8] P. Tarau. Towards Logic Programming Based Coordination in Virtual Worlds. In *Proceedings of HICSS'98, Software Technology: Coordination Languages, Models, Systems*, Big Island of Hawaii, Jan. 1998.

[9] P. Tarau. BinProlog 9.x Professional Edition: Advanced BinProlog Programming and Extensions Guide. Technical report, BinNet Corp., 2002. Available from http://www.binnetcorp.com/BinProlog.

[10] P. Tarau. BinProlog 9.x Professional Edition: BinProlog Interfaces Guide. Technical report, BinNet Corp., 2002. Available from http://www.binnetcorp.com/BinProlog.

[11] P. Tarau. BinProlog 9.x Professional Edition: User Guide. Technical report, BinNet Corp., 2002. Available from http://www.binnetcorp.com/BinProlog.

[12] P. Tarau and V. Dahl. Mobile Threads through First Order Continuations. In *Proceedings of APPAI-GULP-PRODE'98*, Coruna, Spain, July 1998.

[13] P. Tarau, V. Dahl, and K. D. Bosschere. Logic Programming Based Coordination in Virtual Worlds. In W. Conen and G. Neumann, editors, *Coordination Technology for Collaborative Applications - Organizations, Processes, and Agents*. Springer, LNCS 1364, Mar. 1998. ISBN 3-540-64170-X.

[14] P. Tarau, V. Dahl, and K. De Bosschere. A Logic Programming Infrastructure for Remote Execution, Mobile Code and Agents. In *Proceedings of WETICE'97*, pages 106–112, IEEE Computer Society Press, June 1997.

[15] P. Tarau, V. Dahl, and K. De Bosschere. Logic Programming Tools for Remote Execution, Mobile Code and Agents. In *Proceedings of ICLP'97 Workshop on Logic Programming and Multi Agent Systems*, Leuven, Belgium, July 1997.

[16] P. Tarau, V. Dahl, and K. De Bosschere. Remote Execution, Mobile Code and Agents in BinProlog. In *Electronic Proceedings of WWW6 Logic Programming Workshop, http://www.cs.vu.nl/ eliens/WWW6/papers.html*, Santa Clara, California, Mar. 1997.

[17] P. Tarau, V. Dahl, S. Rochefort, and K. De Bosschere. LogiMOO: a Multi-User Virtual World with Agents and Natural Language Programming. In S. Pemberton, editor, *Proceedings of CHI'97*, pages 323–324, Mar. 1997.

[18] P. Tarau, K. De Boschere, V. Dahl, and S. Rochefort. LogiMOO: an Extensible Multi-User Virtual World with Natural Language Control. *Journal of Logic Programming*, 38(3):331–353, Mar. 1999.

[19] P. Tarau and K. De Bosschere. Virtual World Brokerage with BinProlog and Netscape. In P. Tarau, A. Davison, K. De Bosschere, and M. Hermenegildo, editors, *Proceedings of the 1st Workshop on Logic Programming Tools for INTERNET Applications*, JICSLP'96, Bonn, Sept. 1996. http://clement.info.umoncton.ca/ lpnet.

[20] P. Tarau and K. De Bosschere. Virtual World Brokerage with BinProlog and Netscape. In P. Tarau, A. Davison, K. De Bosschere, and M. Hermenegildo, editors, *Proceedings 1st Workshop on Logic Programming Tools for INTERNET Applications*, Bonn, Sept. 1996. http://clement.info.umoncton.ca/ lpnet.