Integracja Prologa z Java Server Pages oraz protokołami Http i Tcp

Łukasz Pękala
PWSZ Tarnów, 2010/2011

Plan prezentacji:

1. Wątki w Prologu
2. Komunikacja Tcp/Ip
3. Prolog jako klient Http
4. Prolog jako serwer Http
5. Prolog i Java Server Pages
6. Krótko o Prolog Server Pages

1. Wątki w Prologu:

Wątki zostały zaprojektowane z myślą o:

- aplikacjach internetowych
- interaktywnych aplikacjach
- integracji z kodem programów napisanych w innym języku np. C

Wątki mają własny stos, współdzielą jedynie:

- predykaty
- rekordy
- flagi

Podstawowe predykaty do tworzenia i obsługi wątków:

thread_create*(:Goal, -Id, +Options)*

Przykłady użycia predykatu:
thread_create(zadanie, ID1, []).
thread_create(do_nothing, ID3, [alias(watek)]).

Więcej o parametrze *+Options*
http://www.swi-prolog.org/pldoc/man?predicate=thread_create%2F3

Predykat *thread_self/1* zwraca identyfikator wątku, w którym został wywołany:

thread_self(ID).

threads.

Przykładowy efekt wywołania predykatu *threads.*

```
----------------------------------------------------------
              Thread  Status
----------------------------------------------------------
                main  running
                   2  running
                   3  true
                   4  running
                   5  running
               watek  running
----------------------------------------------------------
true.
```

Listing: *1watki.pl*

% wersja bez muteksów

```
start :-
      thread_create(write_ID(10), ID1, []),
      write('Thread with ID: '), write(ID1), writeln(' has been started!'),
      thread_create(write_ID(10),ID2,[]),
      write('Thread with ID: '), write(ID2), writeln(' has been started!').

write_ID(0).

write_ID(Times) :-
      thread_self(ID),
      write(' '),
      write(ID), write(': I am running! '),
      NewTimes is Times - 1,
      write_ID(NewTimes).
```

Muteksy:

Tworzenie i usuwanie muteksów:

mutex_create(?MutexId).
mutex_create(moj_muteks).

mutex_destroy(+MutexId).
mutex_destroy(moj_muteks).

Blokowanie i zdejmowanie blokad:

mutex_lock(+MutexId)
Nakłada blokadę. Muteksy mogą być blokowane wielokrotnie przez ten sam wątek!
Inne wątki mogą blokować muteks tylko gdy zostanie on odblokowany tyle samo razy ile został on zablokowany. Jeśli *MutexId* jest atomem i nie ma aktualnie muteksu o podanej nazwie, to muteks zostanie stworzony automatycznie poprzez predykat mutex_create/1

mutex_trylock(+MutexId)
Działa podobnie jak predykat mutex_lock/1, ale jeśli muteks jest przetrzymywany przez inny wątek, predykat natychmiastowo wraca z porażką (fail.).

Zamiast kombinacji lock/unlock:

with_mutex(+MutexId, :Zadanie)
Wykonuje *Zadanie* Muteks zostanie odblokowany niezależnie od tego czy *Zadanie* się powiedzie, zakończy porażką (fail) albo nastąpi wyjątek. Wyjątek wyrzucony przez *Zadanie* jest wyrzucany ponownie po odblokowaniu muteksu.

Listing: 2*watki.pl – Najprostsze zastosowanie muteksów*

```
% wersja z muteksami
start :-
      thread_create(write_ID(10), ID1, []),
      write('Thread with ID: '), write(ID1), writeln(' has been started!'),
      thread_create(write_ID(10),ID2,[]),
      write('Thread with ID: '), write(ID2), writeln(' has been started!').

write_ID(0).

write_ID(Times) :-
      thread_self(ID),
      mutex_lock(display),
      write(' '),
      write(ID), write(': I am running! '),
      mutex_unlock(display),
      NewTimes is Times - 1,
      write_ID(NewTimes).
```

Komunikacja między wątkami:

thread_send_message*(+QueueOrThreadId, +Term).*
thread_send_message(4, say("Costam Costam")).
thread_send_message(watek, say("Hello!")).

thread_get_message*(?Term).*
thread_get_message(say(A)).

2. Prolog i Protokół Tcp/Ip:

- *tcp_socket(-SocketId)*
  inicjalizuje gniazdo i unifikuje jego identyfikator ze zmienną SocketId.
- *tcp_connect(+Socket,+Host:+Port)*
  dokonuje połączenia z hostem na wskazanym porcie przy użyciu uprzednio zainicjowanego gniazda.
- *tcp_open_socket(+SocketId,-InStream,-OutStream)*
  otwiera dwa strumienie (wejścia i wyjścia) SWI-Prolog dla gniazda.
- *tcp_close_socket(+SocketId)* zamyka gniazdo. Ten sam efekt można uzyskać zamykając strumienie gniazda predykatem *close/1*.
- *tcp_bind(+Socket, ?Port)*
  Powiązuje *Socket* z podanym portem na komputerze, na którym działa program.
  Aby zaimplementować obsługę serwera potrzebujemy jeszcze predykatów tcp_listen/2 oraz tcp_accept/3 .
- *tcp_listen(+Socket, +Backlog)*
  Uwaga: Użycie tego predykatu ma sens dopiero po użyciu *tcp_bind* .Użycie tego predykatu powoduje rozpoczęcie nasłuchu przychodzących połączeń. Parametr *Backlog* wskazuje ile oczekujących połączeń jest dozwolonych. Oczekujące połączenia to te które nie zostały jeszcze obsłużone przez tcp_accept/3. Jeżeli liczba *Backlog* zostanie przekroczona klientowi jest zwracana

informacja „Usługa tymczasowo niedostępna”.

- *tcp_accept(+Socket,-Slave,-Peer)*
  Działa tylko po stronie serwera, akceptuje połączenie ze strony klienta, tworzy nowe gniazdo *Slave* dla klienta, którego adres IP jest określony przez zmienną unifikowaną z parametrem *Peer.*

Listing: *1tcp.pl – Dodawanie faktów do bazy wiedzy poprzez protokół Tcp*

```
:- use_module(library(socket)).
:- dynamic(matka/2).

% Przykładowe użycie:
% 'matka(kasia,robert).'

send_msg(X) :-
  create_client(X).

create_client(X) :-
      tcp_socket(Socket),
      tcp_connect(Socket,localhost:33888),
      tcp_open_socket(Socket,Read,Write),
      write(Write, X),nl(Write),
      close(Read),
      close(Write).

% Uruchamia serwer w nowym wątku:
start_server :-
  thread_create(create_server, _, [alias(watek_serwera)]).

create_server :-
  write('Serwer uruchomiony...'), nl,
  tcp_socket(Socket),
      tcp_bind(Socket, 33888),
      tcp_listen(Socket, 5),
  tcp_open_socket(Socket, _, _),
      odbierz(Socket).


odbierz(Socket) :-
  repeat,
    tcp_accept(Socket, Slave, _), % Akceptacja połaczenia ze strony klienta.
    tcp_open_socket(Slave, In, Out),
    read(In, Msg), % Odczyt wiadomości.
    assert(Msg),
  fail.
```

3. Prolog jako klient Http:

SWI-Prolog udostępnia dwie biblioteki umożliwiające obsługę protokołu Http od strony klienta:

- http/http_open - oparta jedynie na metodzie HTTP GET
- http/http_client - bardziej rozbudowana.

3.1 Biblioteka http_open

Biblioteka ta obsługuje jedynie metode GET i jest bardzo ograniczona

Listing: *http1.pl – Najprostszy sposób na pobranie zawartości strony – biblioteka http/http_open:*

```
:- use_module(library(http/http_open)).

view_site :-
     http_open('http://www.google.pl', In, []),
     copy_stream_data(In, user_output),
     close(In).
```

Więcej przykładów:

- **http://www.swi-prolog.org/pldoc/doc_for?object=section(3,'2.1',swi('/doc/packages/http.html'))**
- **https://ai.ia.agh.edu.pl/wiki/pl:prolog:prolog_lab:prolog_lab_http**

3.2 Biblioteka http_client

Obługuje metody GET i POST.
Opis działania wymienionych metod jest dostępny w wikipedii:
http://pl.wikipedia.org/wiki/GET_(metoda)
http://pl.wikipedia.org/wiki/POST_(metoda)

Dostępne predykaty w tej bibliotece:
http_get(*+URL, -Reply, +Options*)
Przeprowadza operację GET na podanym URL-u i zwraca odpowiedź używając predykatu http_read_data/3
Parametr *Options* to lista predykatów, niektóre z nich to:
proxy(*+Host, +Port*)
http_version(*Major-Minor*)
request_header(*Name = Value*)
reply_header(*Header*)
http_post(*+URL, +In, -Reply, +Options*)

http_read_data(*+Header, -Data, +Options*)
http_post_data(*+Data, +Stream, +ExtraHeader*)

4. Prolog jako serwer Http:

http://www.swi-prolog.org/pldoc/doc_for?object=section(2,'3',swi('/doc/packages/http.html'))

Predykat **http_server**(*:Goal, +Options*) - uruchamia serwer, który rozpoczyna wykonywanie *:Goal* .
Na liście podanej jako parametr *+Options* musi znaleźć się predykat **port**(*?Port*).
Inne (wybrane) opcjonalne predykaty z tej listy to:
workers(*+N*) określający maksymalną liczbę wątków uruchamianych w procesie (domyślnie dwa).
timeout(*+SecondsOrInfinite*)- określa limit czasu odpowiedzi. Jeśli nie określimy limitu klient czeka na odpowiedź w „nieskończoność”.

http_handler(*+Path, :Closure, +Options*) – dla wskazanej lokalizacji uruchamia predykat *:Closure*
http_delete_handler(+Path) – unieważnia działanie poprzedniego predykatu dla ścieżki.

reply_html_page(+HEAD, +HTML)- generuje odpowiedź w formie dokumentu HTML.

Listing: *http2_prosty_serwer.pl* – Najprostszy serwer Http Prologa

```
:- use_module(library(http/thread_httpd)).  % serwer
:- use_module(library(http/http_dispatch)). % dispatch
:- use_module(library(http/html_write)).    %


start_server(Port) :-
  http_server(http_dispatch, [port(Port)]).


% Poniższymi poleceniami określamy
% jak mają zostać obsłużone poszczególne adresy przez http_dispatch

:- http_handler('/', root, []).
:- http_handler('/dokumenty/lukasz/', dokumenty_lukasz, []).
:- http_handler('/liczba/', liczba_page, []).

root(_) :-
  reply_html_page([ title(':: Strona Główna ::') ],
                  [ h1('Witaj na mojej stronie'),
                    p( a( href('/dokumenty/lukasz/'), dokumenty)),
                    p( a( href('/liczba/'), 'Twój szczęśliwy numerek'))
                    ]).


dokumenty_lukasz(_) :-
  reply_html_page([ title(':: Dokumenty ::') ],
                  [ h1('Bla bla bla')
                    ]).

liczba_page(_) :-
  X is random(10),
  reply_html_page([ title(':: Twój szczęśliwy numerek ::') ],
                  [ h1('Witaj na szczęśliwej stronie'),
                    'Szczęśliwa liczba dnia to: ', X
                    ]).
```

5. Prolog i Java Server Pages:

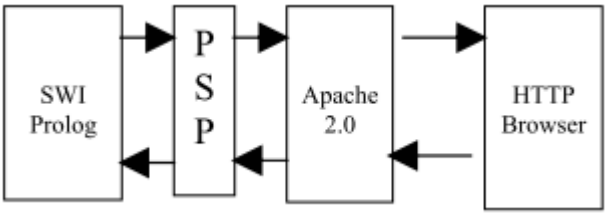
Spis potrzebnych składników:

- SWI-Prolog w wersji 3.1.0 lub nowszy
- Środowisko Javy
- JPL (w nowszych wersjach instalowany razem z SWI-Prolog)
- Przeglądarka internetowa
- Netbeans
- serwer (np. GlassFish)

Uwaga! Jeśli zainstalowaliśmy 64bitową maszynę wirtualną Javy musimy użyć 64bitowej wersji SWI-Prologa. W wersjach 5.10.X występuje błąd uniemożliwiający wykonywanie zapytań do Prologa z poziomu Javy. Problem rozwiązuje zainstalowanie wersji 5.11.X (obecnie beta).

Program będący przykładem integracji technologii Java Server Pages i Prolog znajduje się poza tym plikiem. Patrz folder: **JspPrologTest.**

6. Prolog Server Pages:



Schemat działania technologii PSP

Listing Hello.psp :

```
<html>
<head>
<title>Hello World example</title>
</head>
<body>
<?psp
 msg('Hello, World!').
 ?-msg(X), write(X).
?>
</body>
</html>
```

Kod Prologa zawieramy pomiędzy:

```
<?psp
...
```

?>
Przykład predykatu, który nie zostanie wypisany bezpośrednio:

msg('Hello, World!').

Wypisywane są jedynie zapytania, np.:

?-msg(X), write(X).

PSP sprawia że standardowym wyjściem jest strona www.

Prosper = Prolog Server Pages Extensible Architecture
http://prospear.sourceforge.net

Programming **i**n (Constraint) **L**ogic **L**anguages **o**n the **W**eb
http://clip.dia.fi.upm.es/Software/pillow/

Bibliografia:

1. https://ai.ia.agh.edu.pl/wiki/pl:prolog:prolog_lab:prolog_lab_http
2. http://home.agh.edu.pl/~ligeza/wiki/presentations:prolog
3. http://www.swi-prolog.org/pldoc/refman/
4. http://www.swi-prolog.org/pldoc/package/http.html
5. http://193.226.6.174/roedunet2003/site/conference/papers/SUCIU_A-Prolog_Server_Pages.pdf
6. http://www.swi-prolog.org/packages/jpl/java_api/index.html