Rafał SZKLARCZYK*

INTEGRACJA GNU PROLOG – PHP ZA POMOCĄ WARSTWY POŚREDNIEJ

Streszczenie

Artykuł dotyczy zagadnienia integracji programów CLP działających w formie samodzielnych aplikacji z popularnymi technologiami WWW[1] [8]. Opisywane rozwiązanie oparto o autorski pomysł stworzenia serwera warstwy pośredniczącej, który umożliwia integrację programów CLP napisanych w środowisku GNU Prolog z aplikacjami WWW zaprogramowanymi w języku PHP. Wspomniana warstwa została zaimplementowana jako serwer napisany w języku ANSI C.

Powyższe pozwala na pracę systemu, w którym funkcje interface'u użytkownika obsługiwane są przez aplikację WWW, zaś część obliczeniowa zbudowana jest w oparciu o narzędzia CLP. Dzięki temu można połączyć zalety aplikacji WWW z możliwościami narzędzi CLP.

Stworzone oprogramowanie umożliwia rozwiązanie problemu timeout'ów serwera WWW podczas wykonywania długich obliczeń programów CLP oraz może stać się bazą do rozwinięcia bardziej ogólnego narzędzia o większych możliwościach.

***Słowa kluczowe**: CLP, GNU Prolog, PHP, ANSI C Sockets*

Wstęp

Kiedy w dzisiejszych czasach mówimy „internet", bardzo często mamy na myśli pajęczynę stron WWW. Aplikacje działające w oparciu o protokół HTTP stały się jednym z najważniejszych narzędzi internetowych. Popularność wspomnianych aplikacji ciągle rośnie[2].

Programy działające w sieci WWW mają wiele zalet, działają w oparciu o model klient/serwer. Instalacja aplikacji odbywa się po stronie serwera. Po stronie klienta z reguły wystarcza zainstalowana przeglądarka internetowa. Opisana architektura upraszcza proces wdrożenia aplikacji i zarządzania aktualizacjami. Czynności związane

*Dr inż. Rafał Szklarczyk, adiunkt w katedrze informatyki w Akademii Techniczno-Humanistycznej w Bielsku-Białej.

[1] R. Szklarczyk, *GNU Prolog-PHP multi-tier integration*, IDAACS'13, Berlin 2013.

[2] http://news.netcraft.com/

z instalacją, wykonuje się w większości wypadków po stronie serwera internetowego. Dodatkową zaletą jest możliwość łatwego zorganizowania centralnego systemu kopii bezpieczeństwa.

Stosowanie aplikacji HTTP rozwiązuje problem wieloplatformowości. Większość współczesnych systemów operacyjnych jest dostarczanych z przeglądarkami internetowymi. Niestety aplikacje WWW mają również wady. Przeglądarki internetowe nie są kompatybilne w stu procentach. Dlatego zachodzi konieczność przeprowadzania testów i dostosowywania do różnych przeglądarek. W praktyce zakłada się kompatybilność aplikacji z najpopularniejszymi przeglądarkami. Czasami kłopoty sprawia np. dostęp do lokalnych zasobów komputera klienta. Twórcy przeglądarek celowo blokują niektóre możliwości celem poprawy bezpieczeństwa i odporności na ataki hackerskie. Brak także obsługi połączeń TCP/IP w postaci interface'u gniazd z poziomu skryptów JavaScript. Powyższe zalety i ograniczenia skłaniają do poszukiwań nowych, bezpiecznych technik realizowania nietypowych zadań przez aplikacje WWW.

Jednym z takich zadań jest integracja narzędzi CLP *(ang. Constraint Logic Programming)* z aplikacjami WWW. W niektórych zastosowaniach narzędzia CLP sprawdzają się dobrze[3]. Ważną cechą narzędzi CLP jest elastyczność, rozumiana jako zdolność do formułowania modeli problemów, do których można łatwo wprowadzać modyfikacje[4]. Dodatkową zaletą narzędzi CLP jest zdolność do rozwiązywania problemów, dla których nie znaleziono jeszcze postaci kanonicznej[5]. Narzędzia CLP posiadają wbudowaną obsługę mechanizmów takich jak: *branch and bound, forward checking and looking ahead*[6]. Są zatem interesującą propozycją narzędzi optymalizacji problemów kombinatorycznych. Zasadne jest poszukiwanie metod integracji narzędzi CLP z aplikacjami WWW.

---

[3] T. Szczygieł, *Solving selected packing problems: traditional approaches versus constraint logic programming*, praca doktorska, Politechnika Śląska, Wydział AEiI, Gliwice 2002; T. Tatoń, *Wybrane problemy aukcji i przetargów kombinatorycznych*, praca doktorska, Politechnika Śląska, Wydział AEiI, 2008.

[4] R. Szklarczyk, *Constraint Logic Programming solution for Capacitated Vehicle Routing Problem*, AI-METH, Gliwice 2007.

[5] R. Szklarczyk, *Zastosowanie programowania w logice z ograniczeniami do problemu marszrutyzacji pojazdów*, praca doktorska, Politechnika Śląska, Wydział AEiI, Gliwice 2009.

[6] A. Niederliński, *Programowanie w logice z ograniczeniami. Łagodne wprowadzenie do platformy ECLiPSe*, Wyd. Prac. Komp. J. Skalmierskiego, Gliwice 2010.

1. Przykładowe techniki integracji

Podejmowano już próby implementacji rozwiązań opartych o protokół HTTP, które używają technologii programowania z ograniczeniami lub programowania w logice z ograniczeniami. W roku 1995 Lee Naish zaprezentował NU-Prolog, który działał jako program CGI[7]. NU-Prolog nie jest narzędziem CLP, nie wspiera ograniczeń.

Można uruchomić inne oprogramowanie jako CGI, w tym także programy CLP. Niestety czasami programy CLP mogą działać nawet w ciągu kilku godzin[8]. W powyższym przypadku uruchomienie programu CLP jako CGI spowodowałoby tak zwany *timeout*, czyli zatrzymanie wykonywania programu na skutek zbyt długiego czasu działania. Powyższy problem może pojawić się w szczególności, gdy potrzebne są obliczenia dla problemów o dużej złożoności.

Narzędzia CLP takie jak CHIP, czy GNU-Prolog umożliwiają dostęp do najlepszego znalezionego rozwiązania na danym etapie optymalizacji. Jeżeli użytkownik jest zadowolony z otrzymanych wyników na danym etapie, może zatrzymać działanie programu i nie czekać na jego całkowite zakończenie. Skorzystanie z opisanej funkcjonalności byłoby bardzo trudne w przypadku uruchomienia programu CLP jako CGI.

Innym sposobem integracji narzędzi CLP z aplikacjami WWW jest próba implementacji narzędzi CLP po stronie klienta. Rozwiązaniami działającymi według powyższej zasady są: JLog dostępny na stronie sourceforge.net[9] oraz [1] – implementacja CLP w języku JavaScript. Obydwa projekty są interesujące i mogą być użyteczne. Niestety czas wykonywania programu CLP w narzędziu zaimplementowanym w JavaScript może być nawet 47 razy dłuższy od czasu wykonywania oryginalnego programu napisanego w Ciao Prolog. W opisanych powyżej rozwiązaniach obliczenia odbywają się po stronie klienta, nie daje to możliwości skorzystania z mocy obliczeniowej serwera.

Mechanizmem, który może być zastosowany do implementacji aplikacji internetowych bazujących na narzędziach CLP, jest integracja z językiem Java. W tym miejscu należy wspomnieć o Jinni Paul'a

[7] L. Naish, *HTML Web Forms Interface to NU-Prolog*, 1995. http://www.cs.mu.oz.au/lee/src/forms/

[8] R. Szklarczyk, *Zastosowanie programowania w logice z ograniczeniami do problemu marszrutyzacji pojazdów*, praca doktorska, Politechnika Śląska, Wydział AEiI, Gliwice 2009.

[9] http://jlogic.sourceforge.net

Tarnau'a[10]. Technik agentowych łączących Javę z narzędziami CLP używa też Wojciech Pieprzyca w swojej pracy doktorskiej[11].

Bardzo popularnym narzędziem do tworzenia aplikacji internetowych jest język PHP. Integrację PHP z narzędziami CLP można osiągnąć w bardzo prosty sposób. Do tego celu można wykorzystać polecenia *exec* lub *system* jak to zostało zrobione w [4]. Niestety opisane rozwiązanie nie eliminuje problemu timeout'ów. Dodatkowo administratorzy serwerów internetowych mogą blokować dostęp do wspomnianych poleceń w celu poprawy bezpieczeństwa.

Bardzo ciekawy interfejs został stworzony dla środowiska ECLiPSe[12]. Wzmiankowane rozwiązanie funkcjonuje w taki sposób, że program napisany w środowisku ECLiPSe działa jako serwer nasłuchujący na wybranym porcie protokołu TCP/IP. Program klient, który jest napisany w języku PHP, łączy się z serwerem poprzez interface gniazd. Do realizacji powyższego celu program klient wykorzystuje bibliotekę obiektową, która została zaprogramowana w PHP. Po nawiązaniu połączenia z serwerem klient wysyła zapytanie i określa cel *(ang. goal)* dla programu CLP. Następnie program jest wykonywany przez serwer, a wyniki jego działania są odsyłane do klienta. Powyższy model zakłada obsługę wykonania programu PHP oraz CLP przez pojedyncze zapytanie pochodzące z protokołu HTTP. Założenie wyeliminowania problemu timout'ów poprzez odseparowanie wątku obsługi programu CLP od obsługi zapytania HTTP musiałoby zostać dopiero zaimplementowane. Opisane rozwiązanie jest dedykowane dla środowiska ECLiPSe. Próba zaadoptowania systemu do pracy z innymi środowiskami wymagałaby oddzielnej pracy.

2. Proponowane rozwiązanie

Proponowane rozwiązanie zakłada wprowadzenie warstwy pośredniej. Wspomnianą warstwę stanowi serwer napisany w języku ANSI C. Za pomocą serwera skrypty napisane w języku PHP mogą sterować wykonywaniem programów CLP. Programy CLP mogą być inicjowane przez skrypty PHP. Działają jednak niezależnie i mogą

[10] P. Tarnau, *Jinni: Intelligent mobile agent programming at the intersection of java and prolog*, PAAM'99, 1999.
[11] W. Pieprzyca, *Techniki agentowe w środowiskach Eclipse i Java wraz z zastosowaniami*, praca doktorska, Politechnika Śląska, Wydział AEiI, 2009.
[12] http://code.google.com/p/php-eclipseclp/

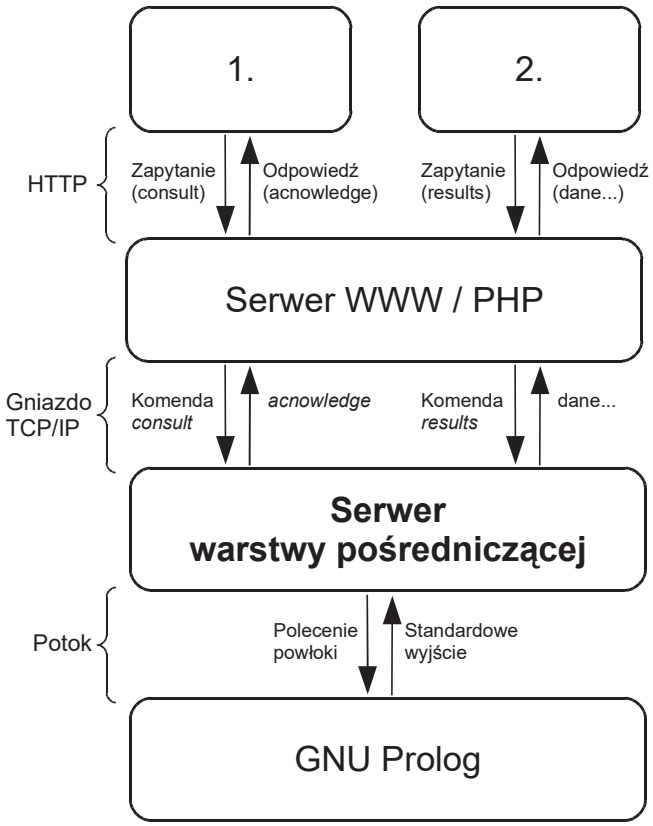
wykonywać się przez dłuższy czas nie powodując jednocześnie wystąpienia timeout'u. W trakcie wykonywania programu CLP jego status może być sprawdzany przez program PHP. Separacja zadań pomiędzy różne warstwy aplikacji umożliwia przeniesienie funkcji GUI do aplikacji internetowej napisanej w PHP, natomiast zadania obliczeniowe mogą być wykonane przez program CLP. Aplikacja WWW przez cały czas działania programu CLP ma dostęp do jego dotychczasowych wyników, w tym do najlepszego jak dotąd znalezionego rozwiązania problemu optymalizacyjnego. Poszukiwania rozwiązania są niezależne od działania aplikacji WWW. Dzięki takiej architekturze zostaje zachowana interaktywność. Program PHP cały czas reaguje na akcje użytkownika. Możliwe jest także przerwanie wykonywania programu CLP. Powyższe jest szczególnie użyteczne, gdy czas wykonywania programu CLP jest dłuższy niż timeout serwera WWW.

Najważniejsze komponenty, które wchodzą w skład systemu, to:

- przeglądarka internetowa;
- serwer WWW z obsługą PHP;
- serwer warstwy pośredniczącej stworzony w ANSI C;
- GNU Prolog.

Diagram działania systemu przedstawiono na rysunku (Rysunek 1).

Rysunek 1. Diagram działania systemu



Źródło: opracowanie własne

Zapytania HTTP są wysyłane przez przeglądarkę do serwera WWW. Następnie serwer uruchamia program PHP, który nawiązuje połączenie z serwerem warstwy pośredniczącej. Połączenie jest realizowane za pomocą interface'u gniazd TCP/IP *(ang. socket connection)*. W dalszej kolejności program PHP przesyła do serwera pośredniczącego komendę. Sposób komunikacji programu PHP z serwerem jest określony przez protokół opisany poniżej. Po otrzymaniu odpowiedniej komendy serwer uruchamia program CLP poprzez odpowiednie polecenie powłoki systemu operacyjnego za pośrednictwem mechanizmu potoków *(ang. pipeline)*.

Obsługa uruchomienia programu CLP jest realizowana w oddzielnym wątku. Dalsze połączenia do serwera są możliwe, gdyż są obsługiwane przez główny wątek serwera.

Po uruchomieniu programu CLP serwer przesyła potwierdzenie z powrotem do programu PHP za pośrednictwem otwartego gniazda TCP/IP.

Protokół komunikacyjny jest bardzo prosty. Dane przesyłane są w formacie tekstowym. Protokół obejmuje obsługę trzech komend:

- consult – do uruchomienia programu CLP; komenda potrzebuje dodatkowego parametru w postaci ścieżki do programu CLP, który ma być uruchomiony;
- results – do pobrania aktualnych wyników wykonywania programu CLP, czyli standardowego wyjścia przechwyconego przez potok;
- stop – do zatrzymania wykonywania programu.

3. Server w ANSI C

Serwer warstwy pośredniczącej oraz GNU Prolog są zainstalowane na tej samej maszynie. Szkielet serwera stanowi główna pętla wykonywana w głównym wątku, która została zaprezentowana w Listing 1.

Listing 1. Główna pętla serwera warstwy pośredniczącej

```
while(keep_working){
      listen(listenfd, LISTENQ);
      printf("Awaiting connection...\n");

      connfd = accept(listenfd, (struct sockaddr *)
NULL, NULL);
      printf("Connected...\n");

      n = read(connfd, buf, BUFSIZE);
      buf[n] = 0; /* end of C string */
      printf("Received: %s\n",buf);

      if(strcmp(buf, "stop")==0){
          keep_working = 0;
          write(connfd, "Stoping server...\n",
strlen("Stoping server...\n"));
          if(thread_running){
              pthread_kill(thread,SIGKILL);
              thread_running = 0;
          }
      }

      if(strcmp(buf, "results")==0){
```

```
            if(thread_running){
                write(connfd,    "Consulting    thread    is
running.\n", strlen("TConsulting thread is running.\n"));
            }
            else{
                write(connfd,  "Consulting  thread  is  not
running.\n",     strlen("TConsulting     thread     is     not
running.\n"));
            }
            for(i=0; i<result_count; i++){
                write(connfd,                    result[i],
strlen(result[i]));
            }
        }

        if(strncmp(buf, "consult",strlen("consult"))==0){
            if(thread_running){
                write(connfd,        "Thread        already
running...\n", strlen("Thread already running...\n"));
            }
            else{
                program_path = &buf[strlen("consult")];
                printf("Consulting: %s\n",program_path);
                write(connfd,            "Consulting...\n",
strlen("Consulting...\n"));
                pthread_create(&thread,              NULL,
thread_function, (void *) program_path);
            }
        }

        write(connfd, "EOF", strlen("EOF"));
        close(connfd);
    }
```

Algorytm pracy serwera można opisać następująco:

I. Oczekiwanie na połączenie.
II. Ustanowienie połączenia i odczyt łańcucha znaków z otwartego gniazda TCP/IP.
III. Jeżeli odczytano prawidłową komendę, wykonanie odpowiedniej procedury obsługi.
IV. Zapisanie odpowiedzi do otwartego gniazda na podstawie wyników procedury obsługi.
V. Zamknięcie gniazda TCP/IP.

Jeżeli przesłano komendę *consult* z odpowiednim parametrem, który zawiera ścieżkę do inicjowanego programu CLP, to uruchamiany jest wątek obsługi pokazany w Listing 2.

Listing 2. Funkcja wątku obsługi

```
void *thread_function(void *argument)
 {
     thread_running = 1;
    char* path;
    char* cmd;

    path = (char *) argument;

    if(strlen(path)>0){
     cmd    =    malloc(strlen("gprolog    --consult-file
")+strlen(path)+strlen(" --query-goal top") + 1);
     strcpy(cmd, "gprolog --consult-file ");
     strcat(cmd, path);
     strcat(cmd, " --query-goal top");
     shell_command(cmd);
    }

     thread_running = 0;
    return NULL;
 }
```

Wątek obsługi otwiera potok (patrz Listing 2 i Listing 3) i uruchamia program CLP za pomocą komendy powłoki systemu operacyjnego.

Listing 3. Komenda powłoki / potok do programu CLP

```
int shell_command(char *cmd){

   FILE *pp;

   if(result_count>0){
     result_count=0;
   }

   pp = popen(cmd, "r");
   if (pp != NULL) {
     while (1) {
       char *line;
       char buf[1000];

       line = fgets(buf, sizeof buf, pp);
       if (line == NULL) break;
```

```
            result_count++;

            result[result_count-1] = malloc(strlen(line) +
1);
            strcpy(result[result_count-1], line);
          }
          pclose(pp);
        }
        return 0;
      }
```

Wyniki odczytywane z potoku są zapisywane przez wątek obsługi do zmiennej globalnej, która jest dostępna z poziomu głównego wątku serwera.

Jeżeli przesłano komendę *results*, wyniki przechowywane w zmiennej globalnej są przesyłane do programu klienta PHP.

4. Klient PHP

Program PHP odpowiada głównie za GUI. Dodatkowo musi obsługiwać protokół komunikacyjny z serwerem warstwy pośredniczącej. Klient PHP może być prostą aplikacją lub częścią wielkiego korporacyjnego systemu informatycznego, który potrzebuje wsparcia ze strony narzędzi CLP np. w celu implementacji harmonogramowania.

Od strony klienta PHP proces komunikacji z serwerem warstwy pośredniczącej można opisać następująco:

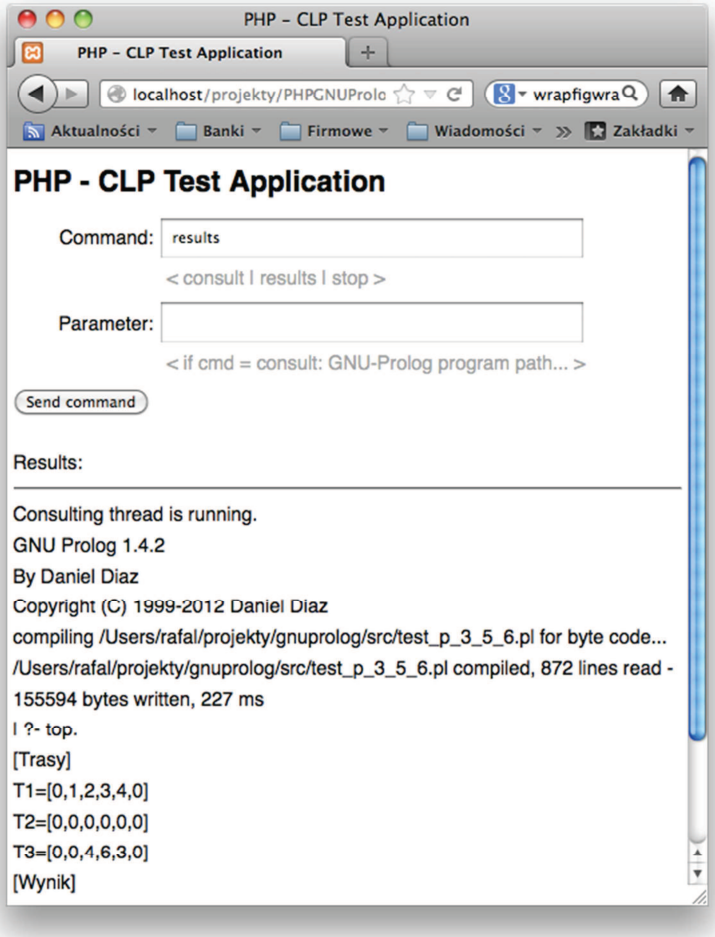
I. Ustanowienie połączenia z serwerem, otwarcie gniazda TCP/IP.
II. Przesłanie komendy (wraz z parametrem) w postaci łańcucha znaków.
III. Odczytanie odpowiedzi serwera w postaci łańcucha znaków.
IV. Zakończenie połączenia.

Na potrzeby zobrazowania działania opisywanego systemu została stworzona testowa aplikacja w PHP. Wspomniana aplikacja składa się z formularza HTML (patrz Rysunek 2), który zawiera dwa pola: *cmd* i *param*. Pole *cmd* jest używane do podania komendy, która ma być przesłana do serwera. Pole *param* jest wykorzystywane do przesłania dodatkowego parametru komendy. We wzmiankowanym polu umieszcza się ścieżkę do programu CLP, która jest potrzebna do wykonania

komendy *consult* zgodnie z opisanym wcześniej protokołem komunikacyjnym.

Wyniki pracy systemu są prezentowane poniżej formularza HTML (patrz Rysunek 2). We wspomnianym obszarze wyświetlane są dane otrzymane z serwera warstwy pośredniczącej, które pochodzą z potoku otwartego do programu CLP.

Rysunek 2. Okno klienta PHP



Źródło: opracowanie własne

Konstrukcja testowej aplikacji jest dosyć prosta. Część odpowiedzialna za komunikację z serwerem i wyświetleniem wyników została zaprezentowana w Listing 4.

Opisany blok kodu programu zostanie wykonany, gdy obecny będzie parametr *cmd* w tablicy PHP *$_GET*. Obecność parametru *cmd* jest z kolei wynikiem przesłania danych z formularza HTML.

Na początku otwierane jest gniazdo protokołu TCP/IP. Następnie komenda, która została przesłana z formularza, przekazywana jest do serwera warstwy pośredniczącej. Jeżeli przekazywaną komendą jest *consult*, dodawany jest parametr (*param*). Bezpośrednio po przesłaniu komendy odczytywana jest odpowiedź serwera, która następnie wyświetlana jest pod formularzem HTML.

Listing 4. Fragment kodu klienta PHP

```
if($_GET['cmd']){
   $address = '127.0.0.1';
   $port = 8000;
   if ((($sock = socket_create(AF_INET, SOCK_STREAM,
SOL_TCP)) === false) {
     echo "socket_create() failed: reason: " .
socket_strerror(socket_last_error()) . "\n";
   }
   $result = socket_connect($sock, $address, $port);
   if ($result === false) {
     echo "socket_connect() failed.\nReason: ($result)
" . socket_strerror(socket_last_error($sock)) . "\n";
   }
   $out = '';
   socket_write($sock,                    $_GET['cmd'],
strlen($_GET['cmd']));
   if($_GET['cmd']=="consult"){
     socket_write($sock,                $_GET['param'],
strlen($_GET['param']));
   }
   while ($out = socket_read($sock, 2048)) {
     echo str_replace("\n","<br>",$out);
   }
   socket_close($sock);
 }
```

Podsumowanie

GNU Prolog został skutecznie zintegrowany z aplikacją PHP. Dla zademonstrowania działania systemu została zaprogramowana i krótko opisana testowa aplikacja w PHP. Stworzono system oddzielający funkcję interface'u użytkownika, za który odpowiada program WWW, od części obliczeniowej, za którą odpowiada program CLP. Powyższe było możliwe dzięki autorskiemu serwerowi warstwy pośredniczącej napisanemu w ANSI C i pozwoliło na rozwiązanie problemu timeout'ów.

Zastosowanie ANSI C powoduje kompilację do formatu bezpośrednio wykonywanego przez maszynę. Taka konstrukcja ma zaletę w postaci optymalizacji wykorzystania mocy obliczeniowej komputera, na którym serwer jest zainstalowany. Niektórzy mniej zaawansowani użytkownicy mogą mieć kłopoty z samodzielną kompilacją i uruchomieniem serwera. W obecnej postaci zaimplementowano jedynie podstawową funkcjonalność serwera. Program CLP musi prezentować wyniki swojej pracy w postaci wyświetlania na standardowym urządzeniu wyjścia. Dzięki czemu wyniki mogą zostać przechwycone przez potok i odesłane do klienta PHP. Serwer obsługuje tylko jedną sesję programu CLP naraz. Oczywistym udoskonaleniem byłaby implementacja możliwości pracy wielu programów CLP naraz.

Przy tworzeniu serwera założono pewne uproszczenia. Na przykład cel wykonywania programu CLP *(ang. goal),* to zawsze predykat o nazwie *top*. Możliwość przesłania dokładnego zapytania do programu CLP mogłaby zostać zaimplementowania. System nie obsługuje komunikacji dwukierunkowej z uruchomionym programem CLP. Po zakończeniu wykonywania programu CLP wątek obsługi się kończy. Nie daje to możliwości wysłania dodatkowego zapytania do załadowanego programu CLP. System zakłada, że program CLP znajduje się już na serwerze. Istnieje możliwość dodania funkcjonalności przesłania programu CLP do serwera warstwy pośredniczącej. Programy CLP mogłyby być dynamicznie generowane przez program klient w PHP.

Występują również inne ograniczenia systemu takie jak maksymalna wielkość wyników programu CLP określona na 5000 linii. W powyższym przypadku można byłoby zastosować dynamiczną alokację pamięci do przechowywania wyników wykonywania programu CLP. Dodatkowo, możliwe jest zaprogramowanie mechanizmu autoryzacji użytkowników i stosownych uprawnień. Wymieniona

funkcjonalność byłaby szczególnie ważna w przypadku próby stworzenia bardziej ogólnego mechanizmu współpracy aplikacji WWW z programami uruchamianymi na serwerze. Perspektywa implementacji opisanego wyżej rozwiązania jest duża. Nie ma konieczności tworzenia klienta w narzędziu PHP. Programy, które są uruchamiane za pośrednictwem serwera nie muszą być napisane w GNU Prologu. Opisany system stanowi zatem podstawę do stworzenia bardziej ogólnego narzędzia o szerokich możliwościach.

Literatura

[1]. Morales J. F., Haemmerle R., Carro M., Hermenegildo M. V., *Lightweight compilation of (C)LP to JavaScript*, Theory and Practice of Logic Programming, vol. 12, no. (4-5), pp. 755–773, 2012.

[2]. Naish L., *HTML Web Forms Interface to NU-Prolog*, 1995. http://www.cs.mu.oz.au/lee/src/forms/

[3]. Niederliński A., Programowanie w logice z ograniczeniami. Łagodne wprowadzenie do platformy ECLiPSe, Wyd. Prac. Komp. J. Skalmierskiego, Gliwice 2010.

[4]. Pieprzyca W., *Techniki agentowe w środowiskach Eclipse i Java wraz z zastosowaniami*, praca doktorska, Politechnika Śląska, Wydział AEiI, 2009.

[5]. Szczygieł T., Solving selected packing problems: traditional approaches versus constraint logic programming, praca doktorska, Politechnika Śląska, Wydział AEiI, Gliwice 2002.

[6]. Szklarczyk R., Constraint Logic Programming solution for Capacitated Vehicle Routing Problem, AI-METH, Gliwice 2007.

[7]. Szklarczyk R., Zastosowanie programowania w logice z ograniczeniami do problemu marszrutyzacji pojazdów, praca doktorska, Politechnika Śląska, Wydział AEiI, Gliwice 2009.

[8]. Szklarczyk R., *GNU Prolog-PHP multi-tier integration*, IDAACS'13, Berlin 2013.

[9]. Tarnau P., Jinni: Intelligent mobile agent programming at the intersection of java and prolog, PAAM'99, 1999.

[10]. Tatoń T., *Wybrane problemy aukcji i przetargów kombinatorycznych*, praca doktorska, Politechnika Śląska, Wydział AEiI, 2008.

Witryny internetowe

[1]. http://code.google.com/p/php-eclipseclp/

[2]. http://jlogic.sourceforge.net

[3]. http://news.netcraft.com/

[4]. *How to Call SWI-Prolog from PHP 5*
http://www.j-paine.org/dobbs/prolog_from_php.html

INTERMEDIATE TIER FOR GNU PROLOG - PHP INTEGRATION

Summary

The paper provides a concept of GNU-Prolog integration with PHP using ANSI C socket server as a middleware [8]. There are a few methods of integration of web programming together with CLP proposed so far. The simplest and the most obvious way is to run CLP program in CGI mode, but it has its disadvantages as well. For some appliance CLP programs can have long execution time that could cause web server timeouts. For that reason a new approach is proposed: to separate user interface (web based) from CLP program execution. That goal is achieved by means of socket server written in ANSI C. The socket server controls the execution of the main program and communicates with the GUI application to send the results. According to what was said above, the logic part is run as GNU-Prolog process and the GUI is built with PHP.

Key words: *CLP, GNU Prolog, PHP, ANSI C Sockets*