PlayerWebClient – wolne oprogramowanie webowe do sterowania robotami działającymi pod serwerem Player

Maciej Przybylski, Barbara Putz
Instytut Automatyki i Robotyki Politechniki Warszawskiej

W artykule przedstawiono pierwsze wolne oprogramowanie umożliwiające sterowanie z poziomu przeglądarki WWW robotami działającymi pod serwerem Player. Opisana aplikacja PlayerWebClient stworzona została dla potrzeb kształcenia na odległość. Jednak jej możliwości sprawiają, że może również znaleźć zastosowanie w pracy z robotami laboratoryjnymi lub jako element interfejsu człowiek-robot. Architektura aplikacji zrealizowana została na bazie oryginalnie rozwiązanej implementacji koncepcji abstrakcyjnego interfejsu użytkownika, który znacznie przyspiesza i ułatwia tworzenie aplikacji.

Słowa kluczowe: Player, robot mobilny, aplikacja webowa

W rozwoju oprogramowania w ostatnich latach można zanotować dwa trendy. Pierwszy z nich to popularyzacja wolnego oprogramowania (*open source*). Kolejny – to rozwój aplikacji webowych. Robotyka, a w szczególności robotyka mobilna jest dziedziną, w której wykonanie prostych czynności wymaga wykorzystania specjalistycznego oprogramowania, które zwykle nie może być przenoszone między systemami operacyjnymi. Zastosowanie aplikacji webowych zwalnia użytkownika z konieczności instalowania tego typu oprogramowania. Jednocześnie umożliwia pracę wielu użytkownikom na tych samych danych lub na tym samym sprzęcie niezależnie od tego, gdzie się aktualnie znajdują. Jedynym warunkiem jest dostęp do sieci komputerowej i przeglądarki stron WWW. Obecne możliwości przeglądarek wraz ze środowiskiem Java Runtime Environment pozwalają na realizowanie większości typowych funkcjonalności, np. dynamiczną wizualizację danych w postaci tekstowej i graficznej, czy też wykonanie poleceń po stronie serwera. Wykorzystanie tej technologii w robotyce może ułatwić prowadzenie badań a także realizację zdalnego nauczania.

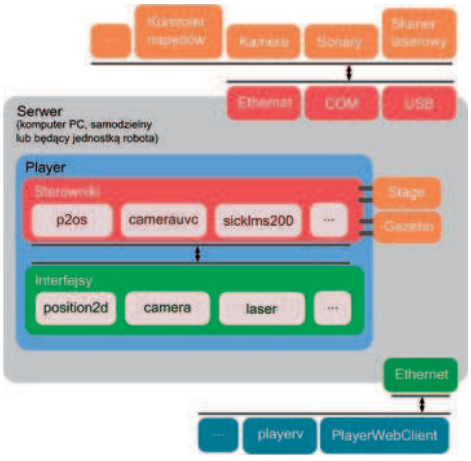
Wprowadzenie – serwer Player

Obecnie dostępnych jest kilka liczących się platform software'owych przeznaczonych dla robotyki [1]. Część z nich ma otwarty kod, jak np. Player [2], OROCOS, CARMEN [4]. Znaczącą część stanowią środowiska komercyjne (Evolution Robotics ERSP, iRobot AWARE, Microsoft Robotics Studio) [1]. Wymienione oprogramowanie różni się możliwościami i przeznaczeniem. Nowym, szybko rozwijającym się oprogramowaniem o dużych możliwościach jest ROS (*Robot Operating System*) [5], który częściowo wykorzystuje rozwiązania z Playera. Wciąż jednak dużą popularnością cieszy się projekt Player stanowiący platformę, w skład której wchodzą serwer robotów `Player` oraz symulatory robotów i otoczenia `Stage` (2D) i `Gazebo` (3D wraz z symulacją oddziaływań fizycznych).

W chwili rozpoczęcia prac nad aplikacją PlayerWebClient żadne środowisko nie oferowało aplikacji webowej umożliwiającej sterowanie, wizualizację danych i programowanie robota. Obecnie na stronie internetowej firmy Willow Garage [6], twórcy systemu ROS, można znaleźć informacje o pracach trwających nad podobnym rozwiązaniem.

Serwer `Player` stanowi oprogramowanie pośredniczące między fizycznymi urządzeniami zainstalowanymi na robocie a programem sterującym typu klient (rys. 1). `Player` działa pod systemem operacyjnym Linux, a od wersji 3.0 może także działać pod kontrolą systemu Windows. Player po pierwsze obsługuje komunikację z fizycznymi urządzeniami, takimi jak sterowniki napędów, enkodery, sonary i inne czujniki, wykorzystując odpowiednie protokoły (RS-232, TCP/IP itp.) i porty (szeregowe, USB itd.). Po drugie jest to serwer, który przez protokół TCP/IP może obsługiwać kilka programów typu klient sterujących robotem i odczytujących dane z jego sensorów przez standaryzowane interfejsy typowych urządzeń m.in.:

- **position2d** – interfejs do sterowania napędami robota i odczytu pozycji obliczonej na podstawie odometrii
- **sonar** – odczyt danych z sonarów, włącznie z informacją o rozlokowaniu sonarów na robocie
- **camera** – udostępnia dane z kamery
- **laser** – interfejs służący do odczytu danych ze skanerów laserowych mierzących odległości od przeszkód znajdujących się wokół robota.



Rys. 1. Schemat serwera Player
Fig. 1. Scheme of the Player server

Wraz z serwerem Player dostarczane są sterowniki do wielu typowych urządzeń i seryjnie produkowanych robotów. Jeżeli wśród standardowych sterowników nie ma odpowiedniego drivera do urządzenia zamontowanego na robocie, możliwe jest samodzielne opracowanie kodu udostępniającego odpowiedni interfejs. Dzięki zestandaryzowanym interfejsom urządzeń aplikacje klienckie umożliwiają kontrolowanie każdego robota i urządzenia niezależnie od konkretnego modelu.

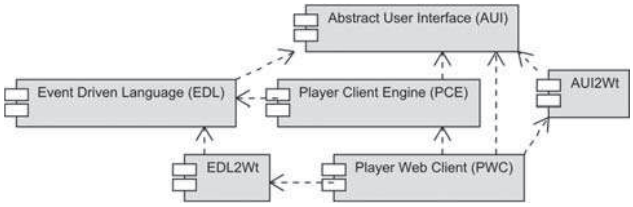
Player do wersji 3.0 działał tylko pod systemem Linux, podobnie jak większość aplikacji typu klient. Aplikacje klienckie tworzone są na bazie biblioteki programistycznej dostarczanej razem z Playerem (interfejsy dla języków C, C++ i Python). W praktyce zbieranie danych i kontrolowanie robota wymaga specjalnej aplikacji klienckiej działającej pod konkretnym systemem operacyjnym. Prezentowana w artykule aplikacja PlayerWebClient jest pierwszą aplikacją typu klient dla serwera Player o interfejsie webowym. Dodatkowo wyposażona w prosty język programowania pomaga zautomatyzować typowe zadania, jak zbieranie danych czy unikanie przeszkód.

Architektura aplikacji PlayerWebClient

Aplikacja PlayerWebClient, zaprojektowana i zrealizowana przez pierwszego z autorów, została w całości utworzona w języku C++. Architektura aplikacji składa się z sześciu podbibliotek:

- Abstract User Interface (AUI) – abstrakcyjny interfejs użytkownika
- Event Driven Language (EDL) – kompilator języka sterowanego zdarzeniami
- Player Client Engine (PCE) – silnik odpowiadający za komunikację z serwerem Player
- Abstract User Interface To Wt (AUI2Wt) – automatyczne tworzenie elementów interfejsu
- Event Driven Language To Wt (EDL2Wt) – automatyczne tworzenie elementów interfejsu związanych z kompilatorem EDL
- Player Web Client (PWC) – właściwa aplikacja z interfejsem webowym stworzonym za pomocą biblioteki Wt.

Relacje między podbibliotekami przedstawione zostały na rys. 2. Każda z podbibliotek ma własną przestrzeń nazw.

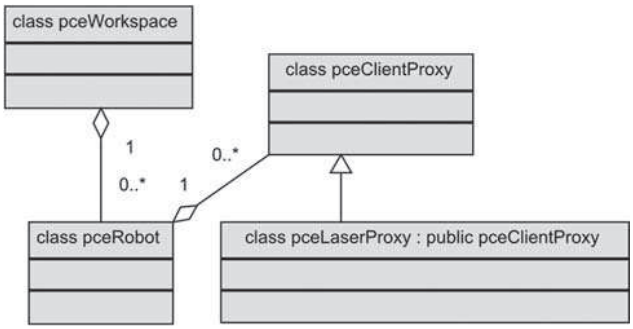


Rys. 2. Diagram zależności między bibliotekami składowymi
Fig. 2. Relation diagram of the sub-libraries

Właściwa komunikacja i obsługa urządzeń podłączonych do serwera Player umieszczona jest w module Player Client Engine (PCE), którego trzonem jest przestrzeń robocza (obiekt klasy pceWorkspace), agregująca dostępne roboty (obiekty klasy pceRobot). Kolejnymi elementami w tej hierarchii są urządzenia robotów (rys. 3).

Nazwy odpowiednich klas dla tych urządzeń tworzone są wg wzorca:

```
pce<nazwa klasy z biblioteki libplayerc++>
```



Rys. 3. Fragment diagramu klas w bibliotece PCE
Fig. 3. Fragment of the class diagram of the classes in the PCE library

Obecna wersja PCE obsługuje takie interfejsy urządzeń serwera Player, jak:

- `aio` – wejścia/wyjścia analogowe
- `dio` – wejścia/wyjścia cyfrowe
- `camera` – kamera
- `laser` – skaner laserowy
- `sonar` – sonary
- `position2d` – kontroler pozycji
- `simulation` – kontroler symulatora.

W obiekcie pceWorkspace znajduje się główna procedura, która wymusza jednoczesne odświeżanie danych we wszystkich obiektach komunikujących się z robotami, a następnie jeżeli uruchomiony jest program użytkownika, wyzwala sprawdzanie zdarzeń i wykonanie odpowiednich poleceń.

Warstwę widoku użytkownika stanowi moduł PWC (Player Web Client), od którego pochodzi nazwa całej aplikacji. Graficzna wizualizacja danych pochodzących z różnych urządzeń wymagała stworzenia osobnych komponentów dla wybranych urządzeń. W obecnej wersji PWC przygotowano wizualizację graficzną dla interfejsów: laser, sonar, position2d, camera. Natomiast interfejs użytkownika stworzony został z wykorzystaniem biblioteki Wt [7] i składa się w dużej mierze z komponentów znajdujących się w modułach AUI2Wt i EDL2Wt.

Język sterowany zdarzeniami

Na potrzeby realizacji projektu został utworzony prosty język sterowany zdarzeniami, którego kompilator znajduje się w bibliotece EDL. O programowaniu sterowanym zdarzeniami najczęściej mówi się w kontekście aplikacji typu serwer, gdzie z otoczenia napływają w sposób niedeterministyczny kolejne zapytania. Z programowaniem sterowanym zdarzeniami można się również spotkać tworząc aplikacje graficzne, gdzie programowane są odpowiednie funkcje dla typowych zdarzeń, takich jak wciśnięcie przycisku lub kliknięcie myszy. Podobny charakter ma proces, jakim jest poruszanie się robota w zmiennym otoczeniu. Przykładem takiego podejścia jest język programowania robotów URBI [8], który stał się inspiracją do opracowania dla potrzeb omawianej aplikacji podobnego, choć prostszego języka programowania.

Podstawowe składniki utworzonego języka to obiekty bazujące na klasie auiObject z biblioteki AUI opisane atrybutami – zmienne, zdarzenia i polecenia. Język nie umożliwia tworzenia obiektów, natomiast pozwala deklarować zmienne oraz definiować własne zdarzenia i polecenia. Programowanie ograni-

cza się głównie do zdefiniowania zdarzeń i określenia poleceń, które zostaną wykonane przy wystąpieniu danego zdarzenia.

Kompilator opracowanego języka współpracuje z dowolną strukturą obiektów pod warunkiem, że obiekty te są instancjami klasy, która dziedziczy po klasie auiObject. Dlatego jego przydatność wykracza poza samo sterowanie robotów. Może również posłużyć np. do programowania interfejsu. Jednak w konkretnym zastosowaniu w kontekście robotyki język otrzymał nazwę **EDeRLezi** utworzoną z akronimu od słów *Event Driven Robotic Language*, czyli język programowania robotów sterowany zdarzeniami.

Definicja zdarzenia w języku EdeRLezi poprzedzona jest instrukcją *event*, po czym wprowadzana jest nazwa zdarzenia, a następnie w nawiasie umieszczane jest wyrażenie, które musi zwracać wartość logiczną *bool*. Wewnątrz wyrażenia można wykorzystać zmienne, polecenia i podstawowe operacje arytmetyczne. Język umożliwia tworzenie własnych poleceń, włącznie z definiowaniem typów argumentów, które dane polecenie przyjmuje. Polecenia użytkownika nie są jednak typowymi funkcjami, ponieważ nie zwracają żadnych wartości.

Najważniejszymi instrukcjami języka są instrukcje *when* (gdy) i *whenever* (za każdym razem, gdy). Po każdej z tych instrukcji w nawiasie umieszczana jest nazwa zdarzenia, a następnie nazwa polecenia, które ma być wykonane. Instrukcja *when* oznacza, że tylko za pierwszym razem, gdy wystąpi zdarzenie, zostaną wykonane przypisane mu polecenia. Natomiast instrukcja *whenever* wykona odpowiednie polecenia za każdym wystąpieniem zdarzenia. Przykładowy program pokazany jest na rys. 4.

Deklaracja zmiennych przypomina deklarację stosowaną w języku C++. Obsługiwane typy zmiennych to: int, double, char, bool oraz string.

```
int a;
when (.run) a=0;

inc(int b)
{
  a=a+b;
}

whenever (.run) .inc(3);

when (.run) .pioneer.position2d0.setSpeed(0.3,0.0);

.pioneer{
  event .toclose(.laser0.minright<0.5
                    || .laser0.minleft<0.5);

  whenever (.toclose) .position2d0.stop();
}
```

Rys. 4. Przykładowy kod w języku EDeRLezi

Fig. 4. Example source code in the EDeRLezi programming language

Decyzja o stworzeniu własnego języka programowania robotów wynikała z faktu, że język URBI był rozwiązaniem zamkniętym i nie dawał pełnego wsparcia dla robotów działających pod Playerem. Ponieważ kod URBI został udostępniony wszystkim zainteresowanym w maju 2010 r., w dalszych pracach zostanie podjęta próba zintegrowania aplikacji PlayerWebClient również z językiem URBI. Jednocześnie opracowany własny język EDeRLezi w dalszym ciągu będzie stanowić część aplikacji, umożliwiając programowanie interfejsu i automatyzację prostych czynności związanych np. z akwizycją danych.

Abstrakcyjny Interfejs Użytkownika

Abstrakcyjny Interfejs Użytkownika (AUI), zrealizowany w nowy, oryginalny sposób, pełni dwie ważne role. Po pierwsze ma na celu przyspieszenie tworzenia interfejsu i ułatwienie dodawania kolejnych funkcjonalności. Po drugie obiekty AUI są bazowymi składnikami języka programowania sterowanego zdarzeniami.

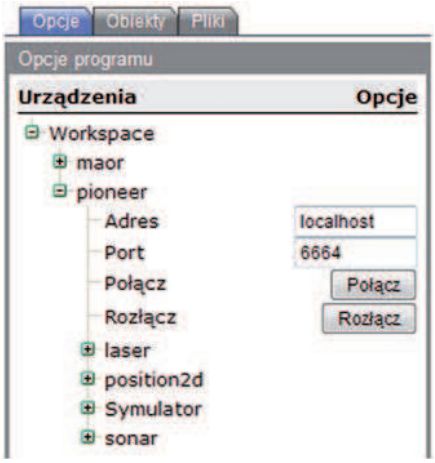
W bibliotece AUI zdefiniowana została klasa bazowa auiObject, po której dziedziczą zarówno obiekty w silniku komunikacji z serwerem Player (PCE), jak i obiekty końcowej aplikacji PWC. Każdy obiekt klasy auiObject przechowuje w mapach indeksowanych unikalnymi nazwami listy zmiennych, poleceń, zdarzeń, właściwości i podobiektów. Zmienne, polecenia i zdarzenia są związane z przystosowaniem modelu do języka sterowanego zdarzeniami. W bibliotece AUI znajduje się również wirtualna maszyna kontrolująca wywołanie odpowiednich reakcji na zdarzenia. Właściwości natomiast mają posłużyć do automatycznego tworzenia elementów interfejsu użytkownika.

Zdefiniowanie właściwości nie oznacza fizycznego stworzenia kontrolki w interfejsie użytkownika, a jedynie ustala, w jaki sposób dana właściwość powinna być wyświetlana. Informacje podawane w trakcie tworzenia właściwości to m.in.: nazwa, nazwa wyświetlana, opis, rodzaj widgetu (pole edycyjne, pole wyboru, lista itp.), wartość początkowa, walidator oraz typowe atrybuty, takie jak widoczność, dostępność, czy zablokowanie możliwości modyfikacji. Ponieważ właściwości przechowywane są w mapach, możliwe jest automatyczne stworzenie odpowiednich kontrolek na podstawie takiej charakterystyki.

Wszystkie informacje związane z daną właściwością znajdują się w jednym miejscu, a dokładniej w klasie, w której została ona zdefiniowana. Dodatkowo charakterystyka właściwości nie jest zakodowana na stałe i może być zmieniana w trakcie działania programu, na przykład na podstawie plików konfiguracyjnych. AUI pozwala na bardzo szybkie dodawanie, nawet w trakcie działania programu, nowych właściwości (zmiennych, funkcji), uwalniając programistę od konieczności ręcznego dodawania kontrolek GUI dla każdej z nich.

GUI aplikacji PlayerWebClient stworzony został za pomocą biblioteki Wt [7], która pozwala na tworzenie aplikacji webowych włącznie ze skompilowaniem własnego serwera WWW lub jako skryptu FastCGI działającego na przykład z serwerem Apache. Interfejs użytkownika tworzony jest półautomatycznie, co oznacza, że w sposób automatyczny tworzone są odpowiednie kontrolki GUI dla własności i innych atrybutów obiektów (rys. 5). Natomiast ich rozmieszczenie w oknie musi zostać zaprogramowane za pomocą dostępnych mechanizmów użytego GUI. Za automatyczne stworzenie kontrolek odpowiadają obiekty biblioteki AUI2Wt. W bibliotece tej znajduje się kilkanaście gotowych widgetów (kontrolek GUI), które pozwalają m.in. automatycznie stworzyć drzewo obiektów wraz z ich właściwościami, lub drzewo obiektów z możliwością podglądu zmiennych, poleceń i zdarzeń. To wystarcza, żeby automatycznie stworzyć interfejs umożliwiający dostęp do większości opcji programu, wymagając ewentualnie dopracowania układu komponentów. Zmienne oraz właściwości wyposażone są w mechanizm sygnałów, z których korzystają widgety AUI2Wt. Oznacza to, że jakakolwiek zmiana wartości zmiennej powoduje odświeżenie stanu odpowiedniej kontrolki. Analogicznie zmiana stanu kontrolki powoduje zmianę wartości zmiennej.

Biblioteka AUI2Wt pozwala na stworzenie interfejsu webowego, jednak stworzenie analogicznej biblioteki np. dla bibliotek Qt czy Ncurses sprawi, że ta sama aplikacja może zostać uruchomiona z typowym interfejsem graficznym lub interfejsem konsolowym. Zaznaczyć należy, że sam pomysł abstrakcyjnego interfejsu użytkownika nie jest pomysłem autorskim; znanym przykładem jego realizacji są choćby języki opisu interfejsów użytkownika [9]. Kolejnym przykładem takiego rozwiązania może być biblioteka AUIT [10] dla języka Java, czy też cała komercyjna technologia Genero [11]. Jednak rozwiązania te nie zakładają możliwości częściowego zautomatyzowania tworzenia interfejsu użytkownika.



Rys. 5. Opcje programu wygenerowane automatycznie na podstawie struktury obiektów i ich właściwości

Fig. 5. Program options which are automatically generated basing on the objects structure and the objects properties

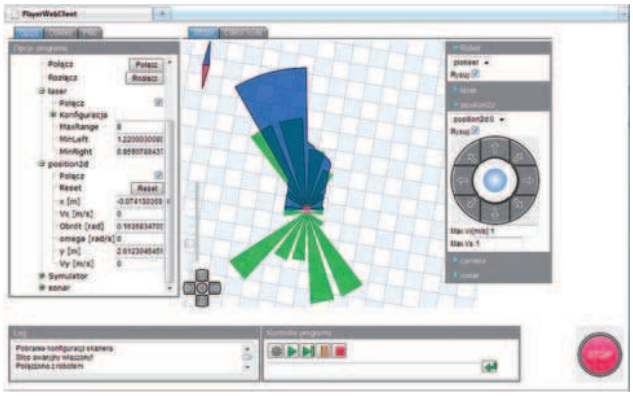
Możliwości aplikacji PlayerWebClient

Aplikacja PlayerWebClient powstała jako element zdalnego laboratorium robotyki tworzonego w ramach Programu Rozwojowego Politechniki Warszawskiej i projektu *e-Lab* prowadzonego przez Ośrodek Kształcenia Na Odległość PW. Głównym przeznaczeniem aplikacji jest e-kształcenie. Jednak już w trakcie tworzenia aplikacji założono dodatkowy cel, jakim było stworzenie interfejsu webowego pozwalającego na łatwe sterowanie oraz akwizycję danych z sensorów robotów działających pod systemem Player. Aplikacja PlayerWebClient wraz z opracowanym językiem programowania może znaleźć zastosowanie w ośrodkach badawczych jako narzędzie ułatwiające pracę z robotami, zbieranie danych lub może stać się bazą dla interfejsów człowiek-robot.

Podstawowa funkcjonalność aplikacji to możliwość ręcznego sterowania robotem. Kontrolowanie robota odbywa się za pomocą grupy przycisków zadających prędkość w jednym z ośmiu kierunków. Możliwe jest również płynne zadawanie prędkości za pomocą kulki, od wychylenia której zależy kierunek i wartość prędkości.

Z możliwością ręcznego sterowania robotem wiąże się konieczność wizualizacji danych z sensorów. W obecnej wersji aplikacji dane wizualizowane są za pomocą grafiki 2D, na której robot przedstawiony jest w widoku z góry. Na obrazie pokazane są aktualne odczyty ze skanera laserowego, sonarów oraz tło zależne od położenia robota (rys. 6). Aplikacja pozwala także na podgląd obrazu z kamery, jeżeli robot lub stanowisko jest w nią wyposażony. Obraz z kamery wyświetlany jest w osobnym oknie.

Aplikacja PlayerWebClient stanowi proste środowisko programistyczne. Ze względu na zastosowanie aplikacji w kształceniu na odległość najważniejszą funkcją jest możliwość tworzenia



Rys. 6. Widok aplikacji PlayerWebClient

Fig. 6. View of the PlayerWebClient application

programu sterującego robotem. Program pisany jest w edytorze umieszczonym na jednej z zakładek aplikacji, natomiast kompilowany i wykonywany jest po stronie serwera. Użytkownik ma podgląd na strukturę obiektów wraz z ich zmiennymi, zdarzeniami i poleceniami, włącznie z wartościami zmiennych i wyrażeń definiujących zdarzenia. W podglądzie tym oprócz efektów kompilacji widoczne są efekty bieżącego działania programu, co ułatwia jego debugowanie.

Użytkownik odpowiednimi przyciskami uruchamia i zatrzymuje program, a także może chwilowo zatrzymać jego działanie lub przejść w tryb wykonania programu krok po kroku. W trybie krokowym kolejny krok oznacza sprawdzenie wszystkich zdarzeń i wywołanie odpowiednich poleceń. Jednakże ze względu na specyfikę języka sterowanego zdarzeniami, wykonywanie krok po kroku może być przydatne tylko w niektórych przypadkach.

Kolejną funkcją programu, która może być przydatna w pracy badawczej, jest zbieranie danych. Dla każdego urządzenia tworzony jest plik tekstowy, w którym dane poprzedzane są stemplem czasowym. Obrazy zapisywane są w plikach .jpg. Zapis danych do plików wywoływany jest z poziomu programu napisanego przez użytkownika. Dzięki takiemu rozwiązaniu użytkownik sam decyduje, jakie dane i w jakim momencie (zależnym od zdarzeń) są zapisywane. W zakładce *pliki* znajduje się lista plików do ściągnięcia z serwera.

W zależności od konfiguracji aplikacji wszyscy zdalni użytkownicy widzą tę samą grupę robotów z określonymi adresami IP i portami lub mają możliwość połączenia z robotem o dowolnym IP i porcie. Również w zależności od konfiguracji aplikacja pilnuje, aby tylko jeden użytkownik mógł w jednej chwili korzystać z jednego robota.

Podsumowanie

Działanie aplikacji PlayerWebClient zostało przetestowane zarówno z symulatorami Stage i Gazebo, dostarczanymi razem z platformą Player, jak i robotami edukacyjnymi klasy miniSumo MAOR, do których został napisany driver dla serwera Player. Aplikacja działa w popularnych przeglądarkach internetowych Firefox, Chrome i Internet Explorer 8.

Prace nad PlayerWebClientem będą kontynuowane, a najbliższe planowane zadania to m.in.:

- wprowadzenie wielojęzyczności abstrakcyjnego interfejsu użytkownika
- dodanie wizualizacji 3D
- dodanie obsługi kolejnych urządzeń
- opracowanie mechanizmu wtyczek.

Planowany jest także rozwój podbibliotek niezwiązanych ściśle z programowaniem robotów (AUI, AUI2Wt, EDL, EDL2Wt) jako zestaw narzędzi do szybkiego tworzenia małych i średnich aplikacji. Wszystkie stworzone biblioteki oparte są na wolnym oprogramowaniu i będą udostępnione na licencji GPL.

Bibliografia

1. Somby M.: *Robotics software platforms review.* Automation.com; [www.automation.com/resources-tools/articles-white-papers/robotics/robotics-software-platforms-review]
2. Gerkey B.P., Vaughan R.T., Howard A.: *The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems*. Proceedings of the International Conference on Advanced Robotics (ICAR 2003), Coimbra, Portugalia 2003.
3. Player – [http://playerstage.sourceforge.net]
4. CARMEN – *Carnegie Mellon Robot Navigation Toolkit* - [http://carmen.sourceforge.net]
5. Quigley M., Conley K., Gerkey B., Faust J., Foote T., Leibs J., Wheeler R., Ng A.Y.: *ROS: an open-source Robot Operating System*. ICRA Workshop on Open Source Software, Kobe, Japonia 2009.
6. *What's Coming Up: Web Interface*. [www.willowgarage.com/blog/2010/03/24/whats-coming-web-interface]
7. *Wt* – [www.webtoolkit.eu/wt]
8. Baillie J.C.: *URBI*: *Towards a Universal Robotic Low-Level Programming Language*. Intelligent Robots and Systems, 2005 (IROS 2005). Edmonton, Kanada 2005.
9. Trewin S., Zimmermann G., Vanderheiden G.: *Abstract User Interface Representations: How Well do they Support Universal Access?* Proceedings of the 2003 conference on Universal usability. Vancouver, Kanada 2003.
10. *AUIT* – [http://auit.sourceforge.net]
11. *Genero* – [www.4js.com/en/b_genero/why.php]
12. Przybylski M., Klimaszewski J., Węclewski P.: *Zdalne laboratorium robotyki*. X Konferencja Virtual University 2010, Warszawa, 16–18.06.2010.

PlayerWebClient – open source web application for Player robots' control

The article presents the first open source software with a web interface for Player robots control. Described application, PlayerWebClient, was originally developed for e-learning. Nonetheless, features of this application make it useful for everyday use with laboratory robots. It could also be a part of a human-robot interface. An architecture of the application is based on an original implementation of the abstract user interface concept, that makes the application development faster and easier.

Keywords: Player, mobile robot, web application

prof. nzw. dr hab. inż. Barbara Putz



Absolwentka obecnego Wydziału Mechatroniki PW, pracownik Instytutu Automatyki i Robotyki PW. Zajmuje się zagadnieniami zaawansowanego modelowania geometrycznego w zastosowaniach CAD/CAM, grafiki komputerowej, monitorowania otoczenia w robotyce. Współautorka pierwszego w języku polskim podręcznika z dziedziny modelowania geometrycznego oraz kilku wydań multimedialnych podręczników do nauczania na odległość z dziedziny programowania, algorytmów i struktur danych oraz techniki obrazowej. Członek Rady Programowej Ośrodka Kształcenia na Odległość PW.

e-mail: *b.putz@mchtr.pw.edu.pl*

mgr inż. Maciej Przybylski



Absolwent specjalności Robotyka na Wydziale Mechatroniki PW. W 2007 r. obronił pracę dyplomową magisterską dotyczącą tworzenia trójwymiarowych map otoczenia przez roboty mobilne. Student studiów doktoranckich w Instytucie Automatyki i Robotyki PW. W pracy naukowej zajmuje się zagadnieniem rozpoznawania otoczenia przez roboty mobilne, w szczególności na podstawie pomiarów 3D.

e-mail: *maciej.przybylski@mchtr.pw.edu.pl*