Rok akademicki 2015/2016

Politechnika Warszawska
Wydział Elektroniki i Technik Informacyjnych
Instytut Informatyki

PRACA DYPLOMOWA INŻYNIERSKA

Paweł Kaczyński

Możliwości programowalne klocka Lego EV3.

Opiekun pracy
dr inż. Henryk Dobrowolski

Ocena: .....

.....

Podpis Przewodniczącego
Komisji Egzaminu Dyplomowego

Kierunek:	Informatyka
Specjalność:	Inżynieria Systemów Informatycznych
Data urodzenia:	1993.03.01
Data rozpoczęcia studiów:	2012.10.01

Życiorys

Urodziłem się 1 marca 1993 roku w Warszawie. W 2012 roku ukończyłem Liceum Ogólnokształcące im. Tadeusza Czackiego w Warszawie. Uczęszczałem do klasy o profilu informatyczno-fizycznym. W październiku tego samego roku rozpocząłem studia na Wydziale Elektroniki i Technik Informacyjnych Politechniki Warszawskiej na kierunku Informatyka. W trakcie studiów wybrałem specjalizację Inżynieria Systemów Informatycznych prowadzoną przez Instytut Informatyki.

.....
Podpis studenta

EGZAMIN DYPLOMOWY
Złożył egzamin dyplomowy w dniu 2016r
z wynikiem .....
Ogólny wynik studiów: .....
Dodatkowe wnioski i uwagi Komisji: .....
.....
.....

Streszczenie

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin pellentesque nisl vitae tellus tempor aliquam. Nullam consequat laoreet pretium. Vivamus vehicula, lectus eu elementum congue, magna dolor pellentesque quam, quis suscipit urna sem sit amet eros. Vivamus tincidunt, leo id egestas condimentum, diam ligula consequat libero, non malesuada orci enim non dolor. Duis nec dolor sit amet ipsum feugiat malesuada ut quis nibh. Quisque ac feugiat enim, rhoncus aliquam justo. Nulla facilisis convallis mauris id mattis. Nam vestibulum, quam non vehicula scelerisque, arcu arcu porta nulla, eget suscipit nisi mauris vel nibh. Donec condimentum, tellus bibendum semper pulvinar, dui lacus dignissim libero, sit amet aliquam justo lacus et magna.

Słowa kluczowe: LEGO Mindstorms EV3, agent mobilny, zachowanie.

Implementation of multiplayer, turn-based strategic game, available through web browsers

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin pellentesque nisl vitae tellus tempor aliquam. Nullam consequat laoreet pretium. Vivamus vehicula, lectus eu elementum congue, magna dolor pellentesque quam, quis suscipit urna sem sit amet eros. Vivamus tincidunt, leo id egestas condimentum, diam ligula consequat libero, non malesuada orci enim non dolor. Duis nec dolor sit amet ipsum feugiat malesuada ut quis nibh. Quisque ac feugiat enim, rhoncus aliquam justo. Nulla facilisis convallis mauris id mattis. Nam vestibulum, quam non vehicula scelerisque, arcu arcu porta nulla, eget suscipit nisi mauris vel nibh. Donec condimentum, tellus bibendum semper pulvinar, dui lacus dignissim libero, sit amet aliquam justo lacus et magna.

Keywords: LEGO Mindstorms EV3, mobile agent, behaviour

Spis treści

Rozdział 1

Wstęp

Celem niniejszej pracy było zaprojektowanie i implementacja aplikacji kontrolującej zachowanie robotów mobilnych. Jest odpowiedzialna za nadzór nad zachowaniami robotów (agentów), sterowanie sensorami i efektorami oraz komunikację z wyższymi warstwami architektury, weryfikującymi poprawność całego systemu.

...

1.1 Motywacja

Motywacją do podjęcia powyższego tematu było przetestowanie możliwości programowalnych oraz technicznych robota zbudowanego z klocków LEGO Mindstorms EV3. Aplikacja działająca na urządzeniu była napisana z użyciem biblioteki ev3dev [3] w języku C++.

Możliwość dostępu do sterującego klockiem centralnym systemu Linux zdejmuje ograniczenie używania prostych środowisk graficznych i pozwala osiągnąć dużo więcej małym kosztem. Należało zatem sprawdzić, co najnowsza wersja LEGO Mindstorms ma do zaoferowania, w szczególności:

- Wydajność napisanych aplikacji z użyciem ww. bilbioteki.
- Skuteczność komunikacji z wykorzystaniem bezprzewodowej sieci Wi-Fi.
- ...

1.2 Założenia

Zostały przyjęte następujące założenia:

- Każdy agent jest zdolny do wykonywania pewnych konkretnych zachowań niezależnie od pozostałych agentów.
- Zachowania te są reprezentowane za pomocą automatu skończonego.

- Dany robot może, ale nie musi być zdolny do wykonania konkretnej czynności. Jest to zależne od podłączonych do niego sensorów i efektorów.
- Każdy robot samodzielnie generuje sposób wykonania danej akcji, na podstawie dostępnych urządzeń.
- Zachowania mogą być dynamicznie tworzone z użyciem specjalnej składni.
- Agenci komunikują się zarówno z jednostką centralną (nadzorująca) jak i między sobą za pomocą sieci bezprzewodowej.
- Urządzeniem nadzorującym może być inny robot, ale pożądana jest też możliwość kontroli z poziomu zwykłego komputera.
- System powinien dostosować się do braków w łączności, umożliwiając komunikację przez pośrednictwo innych agentów.
- Agent może poruszać się tylko po płaskiej powierzchni.

1.3 Ograniczenia

...

1.4 Zawartość rozdziałów

1 Wstęp - zawiera cel pracy razem z założeniami i ograniczeniami.

2 Opis systemu -

3 Budowa aplikacji -

4 Zachowania -

5 Komunikacja -

6 Testowanie aplikacji -

Rozdział 2

Opis systemu

...

2.1 Wymagania

...

2.2 Cechy

...

2.3 Użyta technologia

Istnieje wiele dostępnych środowisk dedykowanych LEGO Mindstorms EV3, z których każde ma trochę inne zastosowanie. Najbardziej dogodnym rozwiązaniem okazał się projekt ev3dev.

2.3.1 O ev3dev

Projekt ev3dev to dopasowana do potrzeb klocka LEGO dystrybucja Linuxa (Debian Jessie), która jest wgrywana na kartę SD i uruchamiana obok istniejącego systemu. Zawsze istnieje możliwość przywrócenia domyślnego stanu klocka przez wyjęcie karty z systemem. Platforma stworzona w ramach ev3dev zawiera wiele sterowników, nie tylko do akcesoriów zestawu EV3, ale także poprzednich dystrybucji LEGO Mindstorms oraz komponentów wytwarzanych przez osoby trzecie. Możliwe jest programowanie klocka w języku C/C++, ale ev3dev obsługuje też wiele innych języków. To wszystko daje dużą swobodę samego programowania, jak i sposobu tworzenia programu i komunikacji z urządzeniem. Kompilacja aplikacji może odbywać się bezpośrednio na urządzeniu lub na komputerze z wbudowanym

kompilatorem na procesory typu ARM. Komunikacja z klockiem centralnym realizowalna jest na trzy sposoby: Za pomocą Wi-Fi, Bluetooth lub przy użyciu kabla USB.

2.3.2 Konfiguracja

Konfiguracja nowego systemu odbyła się w kilku krokach:

1. Na kartę microSDHC wgrany został specjalne spreparowany obraz systemu, pobrany ze strony głównej projektu ev3dev.
2. Przy użyciu połączenia SSH przez kabel USB, system został skonfigurowany i pobrane zostały wszystkie wymagane pakiety.
3. Dalsza komunikacja odbywała się bezprzewodowo z użyciem urządzenia NETGEAR WNA1100, podłączonego do portu USB klocka centralnego.
4. Aplikacja była kompilowana na laptopie z systemem Ubuntu i synchronizowana zdalnie z robotem.

2.3.3 Biblioteka

W ramach projektu ev3dev dostępne są dwa pliki źródłowe napisane w języku C++. Dostarczają one wymagany interfejs do sterowania klockiem centralnym i podłączonymi do niego urządzeniami.

Wersja użytej biblioteki: 0.9.2-pre, rev 3.

Wprowadzone zmiany

Biblioteka nie była kompatybilna ze wszystkimi urządzeniami dostarczonymi przez LEGO, dlatego wymagane było:

- Dopisanie rozpoznawalnych nazw sterowników dla sensorów.
- Zaimplementowanie własnej obsługi diod LED przedniego panelu, w szczególności funkcji migania.

2.3.4 Możliwości

Użyte środowisko Linux orax język programowania C++ dostarczają praktycznie pełnię możliwości programistycznych, a w szczególności:

- Użycie biblioteki stl i zgodność ze standardem C++11.
- Wątki.

- Polimorfizm.
- Komunikację przez protokół UDP z wykorzystaniem gniazd.

2.3.5 Narzędzia

Projekt aplikacji był rozwijany z wykorzystaniem narzędzie NetBeans. Mimo iż sama aplikacja może być skompilowana z poziomu konsoli i narzędzia Makefile, NetBeans dostarcza także wygodne narzędzia debugujące oraz przyspiesza pisanie kodu.

Konfiguracje aplikacji

Zostały zdefiniowane dwie domyślne konfiguracje:

- D_ARM: konfiguracja przeznaczona na urządzenia z procesorami typu ARM. Domyślnie przeznaczona do uruchamiania na robocie mobilnym.

  Kompilator: arm-linux-gnueabi-g++

  Flagi kompilacji: -D_GLIBCXX_USE_NANOSLEEP -pthread -static-libstdc++ -std=c++11 -DAGENT

- D_DESKTOP: konfiguracja kompilowana z myślą o tradycyjnych komputerach osobistych. Domyślnie przeznaczona do uruchomienia w trybie nadzorcy systemu, komunikującego się zdalnie z robotami.

  Kompilator: g++

  Flagi kompilacji: -D_GLIBCXX_USE_NANOSLEEP -pthread -static-libstdc++ -std=c++11

Obie konfiguracje posiadają dodatkowo wersję z przedrostkiem `R_`, które oznaczają wersję Release zamiast wersji Debug.

Inne użyte narzędzia to przede wszystkim aplikacja Doxygen służąca generowaniu dokumentacji kodu programu, system kontroli wersji Git do zarządzania całym projektem oraz oprogramowanie LaTeX, za pomocą którego wygenerowany został ten dokument.

2.3.6 Konstrukcja robota

W celu przetestowania zaimplementowanych funkcjonalności, wyposażono robota w następujące elementy:

- Dwa duże motory do poruszania się po płaskiej powierzchni.

- Ultradźwiękowy sensor odległości ustawiony przodem do kierunku poruszania się.
- Przedni zderzak oraz sensor dotyku do wykrywania zderzeń.
- Sensor koloru do wykrywania zmian w odcieniu powierzchni.

Rozdział 3

Budowa aplikacji

3.1 Moduły i klasy

W celu uzyskania przejrzystości aplikacji, wydzielone zostały moduły[1], które opisują pewien fragment funkcjonalności programu. Moduły niższych warstw mogą być wykorzystywane przez moduły warstw wyższych lub być tylko zestawem dodatkowych narzędzi. Kolejne punkty opisują w czym dany moduł się specjalizuje i jakie klasy wchodzą w jego skład. Szczegółowe dane na temat klas oraz ich metod i pól znajdują się w dokumentacji kodu.

3.1.1 Komendy

Klasy komend są tak naprawdę nakładką na istniejące mechanizmy biblioteki ev3dev, operujące bezpośrednio na sprzęcie. Oprócz właściwej komendy, będącej poleceniem dla efektora bądź sensora, dana klasa zawiera referencje do obiektu, na którym ma zostać wykonana oraz jej parametry, o ile takowe posiada. Nazewnictwo klas dokładnie odwzorowuje nazwy komend przekazywanej urządzeniom. W obrębie konkretnych komend, definiowane są także stałe opisujące charakter przekazywanych argumentów oraz ich limity.
Komendy zostały podzielone na dwie podgrupy:

Komendy motorów: Klasa bazowa - `CommandMotor`. Zawierają referencje do klasy `Motor` oraz opcjonalnie przechowują także przekazywane parametry. Np przykład: `CommandMotorStop`, `CommandMotorRunForever`.

Komendy sensorów: Klasa bazowa - `CommandSensor`. Zawierają referencje do klasy `Sensor`. Definiują obsługiwane tryby danego sensora. Komendy te nie służą do pobierania wartości, lecz tylko do zmiany ich ustawień. Pobieranie wartości używane jest przy pomocy specjalnej klasy `Devices`.
Przykładowa komenda: `CommandSensorSetMode`.

[1] W kontekście tej pracy *moduł* oznacza pewną grupę skojarzonych ze sobą klas.

Klasą bazową dla wszystkich komend jest `Command`.

3.1.2 Akcje

Akcje są kolejnym stopniem abstrakcji definiowania zachowań robota. Klasy akcji przechowują przede wszystkim sekwencje komend, które mają zostać wykonane. Ponadto, z powodu natychmiastowego charakteru wykonywania wszystkich zgromadzonych komend, akcja może mieć zdefiniowany warunek jej zakończenia. Przyjmuje ona postać funkcji anonimowej, w której następuje zwrócenie wartości prawda lub fałsz na podstawie dowolnie sprecyzowanych instrukcji. Pozwala to wyższej warstwie sterującej sprawdzić, czy kolejna akcja może zostać wykonana. Dodatkowo, akcje mogą deklarować dopuszczalne zdarzenia, które przerywają jej działanie lub zmieniają jej parametry.

Wszystkie dostępne klasy akcji są zdefiniowane w aplikacji i nie istnieje możliwość zwiększenia zbioru o nowe bądź dynamicznego generowania nowych, własnych klas. Ta decyzja implementacyjna jest podyktowana specyfiką konkretnych modeli robotów, różniących się budową oraz podłączonymi akcesoriami, a co za tym idzie, odmiennym sposobem implementacji tych samych czynności.

Klasy opisujące konkretne akcje, np. `ActionDriveDistance`, dziedziczą po klasie `Action`. Wspólnymi elementami każdej z nich są: typ, warunek końcowy, sekwencja komend oraz metody wykonawcze. Różnią się natomiast dodatkowymi parametrami, takimi jak prędkość czy kąt obrotu. Ponadto, istnieje możliwość wygodnego zapętlenia jednej lub wielu akcji dowolną liczbę razy za pomocą specjalnej klasy - `ActionRepeat`. Konstruktor tej klasy przyjmuje liczbę powtórzeń oraz listę akcji, które zostaną wykonane w podanej kolejności.

3.1.3 Zachowania

Definiowanie zachowań jest dużo trudniejsze niż akcji czy komend, gdyż bazują one na schemacie automatu skończonego. Oprócz konkretnych akcji, na jakich dane zachowanie ma się opierać, należy zdefiniować przejścia pomiędzy stanami (akcjami) w toku poprawnego wykonania oraz specjalne warunki zmiany stanu w reakcji na zaistniałe zdarzenia (np. napotkana przeszkoda lub utrata połączenia).

Podobnie jak w przypadku akcji i komend, każde zachowanie zdefiniowane jest w osobnej klasie (np. `BehaviourExplore`), które dziedziczy po wspólnej klasie `Behaviour`. Każde z nich zawiera typ, strukturę stanów automatu złożonych z akcji i przejść oraz funkcje wykonawcze. Ponadto zachowania pochodne zawierają własne parametry, np. maksymalny dystans do przejechania. Więcej szczegółów w rozdziale 4. Zachowania.

3.1.4 Urządzenia

Biblioteka ev3dev dostarcza wygodnego interfejsu do zarządzania urządzeniami przez wygenerowanie drzew klas dla sensorów i efektorów. W ramach tej aplikacji, na każdy typ urządzenia została nałożona specjalna klasa pośrednicząca, w środku której dopiero znajduje się referencja do właściwego obiektu. Są to klasy `Motor` oraz `Sensor`, które po odpowiedniej identyfikacji zostają zmapowane do par port-urządzenie.

Całością nadzoruje klasa `Devices` napisana zgodnie ze wzorcem projektowym Singleton. Ograniczone są w ten sposób nadmierne kopie obiektów i potencjalnie niejednoznaczne odwołania. Ponadto klasy urządzeń są potrzebne w wielu różnych miejscach aplikacji, a wzorzec ten umożliwia taki dostęp za pomocą statycznego wydobycia instancji.

Moduł urządzeń jest także odpowiedzialny za detekcję zdarzeń. Wyższe warstwy mogą zgłaszać zdarzenia, na które klasa `Devices` ma nasłuchiwać. Jeśli dane zdarzenie wystąpi, wysyłane jest do odpowiedniej kolejki dla wyższych warstw do przetworzenia. Zgłaszane mogą być również zdarzenie niezależne, np. niski poziom baterii lub odłączenie urządzenia.

3.1.5 Robot

Jest to najbardziej rozbudowana klasa, ponieważ agreguje w sobie działanie wielu modułów. Zarządza zarówno urządzeniami podłączonymi do klocka centralnego, steruje zachowaniem robota oraz przetwarza przesłane komunikaty oraz zdarzenia. Główna metoda klasy `run` jest uruchamiana w głównym wątku aplikacji i w pętli przetwarza wszystkie dane. Jest bezpośrednio zsynchronizowana z wątkiem komunikacyjnym za pomocą dwóch kolejek wiadomości - nadawczej i odbiorczej. Wyróżnienie dwóch niezależnych ścieżek wykonania umożliwia lepsze zarządzanie zasobami sprzętowymi oraz pozwala robotowi na samodzielne działanie, niezależnie od przesyłanych pakietów.

Robot może być fizycznie zbudowany na wiele różnych sposobów. Dlatego wymagane jest, żeby zdefiniowane były konkretne klasy implementujące szczegóły danego modelu. W związku z powyższym, klasa bazowa `Robot` jest klasą abstrakcyjną, a każdemu wariantowi konstrukcji odpowiada osobna klasa podrzędna. W celu ujednolicenia interfejsu, każdy model deklaruje listę obsługiwanych akcji oraz wymaganych do tego podłączonych urządzeń. W ten sposób można łatwo sprawdzić, czy dane zachowanie jest obsługiwane i jakich pomiarów robot może dostarczyć. Klasy konkretnych modeli, bazując na zdefiniowanych funkcjach wirtualnych, definiują swoje wersje w sposób adekwatny do domniemanego efektu końcowego wymaganych akcji.

Każdy instancja robota posiada również maszynę stanów oraz zdefiniowane warunki przejść pomiędzy nimi. Każdy stan przetwarza tylko konkretne komunikaty i zdarzenia. Rezultatem ich przetworzenia może być wysłanie specjalnej wiadomości

zwrotnej lub zmiana stanu na inny. To ostanie wiąże się jeszcze ze zmianą koloru oraz częstotliwości migania diod LED umieszczonych na przednim panelu klocka centralnego.

3.1.6 Komunikacja

Komunikacja pomiędzy różnymi partiami całego systemu odbywa się na dwóch poziomach:

Zdarzenia

Poszczególne moduły robota, takie jak akcje czy klasa urządzeń, mogą generować zdarzenia. Kolejka zdarzeń, będąca Singletonem, posiada wiele punktów wejścia, ale tylko jedno wyjście - klasę `Robot`. Wszystkie klasy zdarzeń dziedziczą po klasie `Event`. Zgłaszane do kolejki obiekty mogą dotyczyć zarówno zmian wartości sensorów, niespodziewanych zmian w przebiegu zachowania lub wyjątków rzuconych przez aplikację.

Protokół UDP

Komunikacja pomiędzy robotami odbywa się przy użyciu sieci bezprzewodowej oraz protokołu UDP. Każdy wysłany pakiet to zakodowany do postaci znakowej obiekt klasy `Message`. Każdy z nich zawiera pięć elementów: identyfikator wiadomości, nadawcy i odbiorcy, typ oraz listę opcjonalnych parametrów. Separatorem parametrów jest dwukropek. Podstawą identyfikacji wiadomości jest jej identyfikator oraz typ. Pierwszy komponent zapewnia synchronizację pakietów oraz eliminację duplikatów, a drugi steruje zmianami stanów agenta oraz jego zachowań. Więcej szczegółów w rozdziale 5. Komunikacja.

3.1.7 Nadzorca

3.1.8 Moduły dodatkowe

Rozdział 4

Zachowania

Rozdział 5

Komunikacja

Rozdział 6

Testowanie aplikacji

Rozdział 7

Podsumowanie

Bibliografia

Publikacje

[1] TEST 1. "Multidimensional monitoring of computer systems". W: *Proc. of IEEE Symp. and Workshops on Ubiquitous, Autonomic and Trusted Computing* (2009), s. 68–74.

[2] TEST 2. *The NAS Kernel Benchmark Program*. URL: `http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19850024482.pdf`.

Źródła internetowe

[3] *Strona domowa ev3dev*. URL: `http://www.ev3dev.org/`.

Dodatek A

Załączniki

A.1 Słownik pojęć

A.2 Wykaz rysunków