Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie

Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej, Inżynieria Oprogramowania, rok V

Studio projektowe

**Inteligentne Oświetlenie Ulicy**

Prowadzący:

dr inż. Radosław Klimek

zespół projektowy :

Skiba Jakub,

Setner Tomasz.

# Spis treści

[**Środowiska Wieloagentowe**](#_yq4nhihgk7t4) **2**

[Geneza zagadnienia](#_rduwv2m62kxw) 2

[Zastosowanie systemów wieloagentowych](#_9lwr9387edph) 3

[Przykładowe frameworki wieloagentowe](#_81bgyd2484vt) 4

[**Charakterystyka projektu**](#_y9rju7983ook) **5**

[**Identyfikacja Aktorów i Agentów w systemie**](#_nwldmt73a723) **5**

[Aktorzy systemu Inteligentnego Oswietlenia Ulicy](#_j6wl1v6a6ded) 5

[Agenci systemu Inteligentnego Oswietlenia Ulicy](#_hkcikzlxxrrf) 6

[Agenci identyfikujący zdarzenia](#_nimpfb8nk6hd) 6

[Agenci magazynujący dane](#_55977ctwzndt) 7

[Agenci wykonawczy (sterujący oświetleniem)](#_cyas6ya3fd6a) 7

[**Scenariusze oswietleniowe systemu Inteligentnego Oswietlenia Ulicy**](#_5p605xalvcjc) **7**

[Niebezpieczny odcinek drogi](#_6j9aus1v34si) 7

[Scenariusz Główny](#_1pg6hz61wvof) 8

[Scenariusze alternatywne](#_d2hpi7k0gd07) 8

[Pieszy na ulicy](#_mt7pkhl44qkk) 8

[Scenariusz Główny](#_fi1off8u3yvj) 9

[Scenariusze alternatywne](#_dr2ous8bv9lx) 9

[Wydarzenie socjalne](#_32n8prwnevw2) 9

[Scenariusz Główny](#_mc7e9fqw6gzr) 10

[Scenariusze alternatywne](#_mbhwbdsa3f47) 10

[**Diagramy komunikacji pomiędzy agentami**](#_akdj8mfyrj1l) **11**

[**Zastosowane rozwiązania techniczne**](#_bdrwq4uufwgv) **12**

[Opis interfejsu programu](#_d3nmucrco8td) 16

[**Opis Loggera do wypisywania czasu reakcji.**](#_m6of87k7sgwf) **19**

[**Wyniki symulacji**](#_5979ttszfm3p) **20**

[Scenariusz 1: samochód osobowy - brak zachmurzenia - Dzień](#_6kwg3vrdztka) 20

[Scenariusz 2: samochód osobowy - brak zachmurzenia - Noc](#_9ao9zksuws4p) 21

[Scenariusz 3: pieszy przy ulicy - Duże zachmurzenie - Dzień](#_wuhpubjelhwl) 22

[**Słownik pojęć**](#_ele2uzjyg20t) **25**

[**Podsumowanie pracy z Frameworkiem JADE**](#_6sfbry5yadk3) **26**

[**Podsumowanie**](#_a5151zggqjra) **27**

[Napotkane problemy](#_ng698z3fr3rp) 27

[Co jeszcze do zrobienia](#_yazpa2ti5dcf) 28

# Środowiska Wieloagentowe

## Geneza zagadnienia

Z definicji, system wieloagentowy jest to system złożony z komunikujących i współpracujących między sobą agentów, realizujących wspólne cele. Systemy wieloagentowe stanowią alternatywę dla scentralizowanego modelu przetwarzania informacji w systemach z jednym agentem. MAS (Multiagent Systems) wywodzą się z obszaru badań nad DAI (Distributed AI). Podstawa systemu wieloagentowego jest pojedynczy agent zdefiniowany jako jednostka autonomiczna osadzona w określonym otoczeniu, które jest przez nią postrzegane i na które może ona wywierać wpływ, potrafiąca komunikować się z innymi agentami. Działanie agenta określane zachowaniem (implementowane jako rozszerzenia klasy Behaviour na platformie JADE) jest planowane i przeprowadzane w oparciu o umiejętności w jakie agent został wyposażony na etapie projektowania i implementacji lub jakie nabył w trakcie swojego działania (agenci uczący się). Działanie prowadzi do osiągnięcia celu, do którego agent dąży.

System wieloagentowy składa się z wielu agentów komunikujących się ze sobą przy jednoczesnym zachowaniu autonomii działania i podejmowania decyzji przez poszczególnych agentów. Agenci podejmują odpowiednie akcje w oparciu o wiedzę jaką dysponują (wiedza początkowa i wiedza zgromadzona w trakcie działania systemu). Nową jakością wprowadzoną przez MAS jest wzajemna interakcja między agentami, modyfikującymi swoją wiedzę i swoje działania w oparciu o zachowania przejawiane przez innych agentów i informacje jakie otrzymuje agent w procesie komunikacji z innymi jednostkami. Racjonalny agent wyposażony jest w strategię prowadzącą go do osiągnięcia założonego celu. Jako część systemu wieloagentowego może realizować strategię współpracy z innymi agentami, jeśli to przybliża go do celu lub nawet w pełni działać w ramach grupy agentów realizujących określony cel .

System wieloagentowy składa się z wielu agentów komunikujacych sie ze soba przy jednoczesnym zachowaniu autonomii działania i podejmowania decyzji przez poszczególnych agentów.

Systemy MAS umożliwiają lepsze i bardziej naturalne modelowanie wielu dziedzin charakteryzujących się rozproszeniem, komunikacja i interakcjami elementów, współpraca/rywalizacja (np. modelowanie społeczności działającej w warunkach gospodarki wolnorynkowej). Używane są często gdy natura rozwiązywanego problemu jest rozproszona i złożona obliczeniowo (symulacje, wyszukiwanie i analiza informacji w sieci). Inna ważna cecha MAS jest odpornosc na błedy i awarie.

## Zastosowanie systemów wieloagentowych

Systemy wieloagentowe stanowia jedna z najszybciej rozwijających się gałęzi badań nad sztuczna inteligencja, jednocześnie są wykorzystywane w konkretnych zastosowaniach przemysłowych/biznesowych:

* rozwiązywanie problemów, których natura jest rozproszona, podejście agentowe ułatwia modelowanie systemów działających w środowisku rozproszonym, mierzących się z problemami, których złożoność przekracza możliwości jednostki.
* symulacje rzeczywistości, zwłaszcza grup składających się z autonomicznych jednostek, znajduje zastosowania w naukach biologicznych i społecznych
* zarzadzanie wiedza, wyszukiwanie, gromadzenie, analiza informacji znajdujących się w sieci internet.
* agenci mobilni, przemieszczający się w sieci i wykonujący zadania na rzecz użytkowników (wyszukiwanie ofert, powiadomienie, organizacja kalendarza, filtrowanie poczty), agent moze stanowic interfejs między użytkownikiem a maszyną
* zagadnienia sterowania w robotyce, agenci reaktywni mogący wpływać na otoczenie w oparciu o impulsy z niego odbierane
* technologie agentowe wpływają na koncepcje modelowanie oprogramowania dostarczając wysokopoziomowej abstrakcji jaką jest agent

## Przykładowe frameworki wieloagentowe

* JADE (Java Agent DEvelopment Framework) - jest frameworkiem w całości napisanym w Javie. Do swojego działania wymaga środowiska JRE w wersji 5 lub nowszej. Upraszcza implementowanie wieloagentowych systemów. Systemy oparte o ten framework mogą działać równolegle na różnych systemach (rozproszenie) i dzielić między sobą konfigurację za pomocą zdalnego GUI. Ponadto konfiguracja może być zmieniana w trakcie działania całego systemu. Poza możliwością implementowania logiki systemów wieloagentowych, JADE dostarcza proste metody uruchamiania zadań oraz komunikacji między nimi. JADE jest darmowym oprogramowaniem, stworzonym przez Telecom Italia i udostępniany na licencji LGPL (Lesser General Public License). Dodatkowo framework ten posiada dużą, aktywna społeczność, która na bieżąco zgłasza błędy oraz pomaga rozwiązywać problemy początkującym użytkownikom. Do frameworka dostępna jest dokładna dokumentacja oraz tutorial, wraz z licznymi przykładami. Wymiana wiadomosci pomiedzy agentami jest zgodna ze standardem ACL (Agent Communication Language). Wiadomości są asynchroniczne i kolejkowane dla każdego agenta z osobna. Przesyłanie obiektów implementujących interfejs serializable jest automatyczne. Framework sam zajmuje się ich serializacja u nadawcy oraz deserializacja w obiekcie docelowym.
* Cougaar (Cognitive Agent Architecture) - jest opensourcowym frameworkiem opartym na Javie, przeznaczonym dla dużych, skalowalnych, zorientowanych agentowo aplikacji. Architektura tego frameworku korzysta z najnowszych zorientowanych agentowo komponentów oraz posiada bardzo wiele funkcjonalności. Cougaar jest rozprowadzany pod licencja BSD.
* MASON (Multi-Agent Simulator Of Neighborhoods) - jest szybka, wieloagentowa biblioteka przeznaczona do symulacji. Posiada wiele narzędzi ułatwiających symulacje zjawisk, w tym biblioteki do wizualizacji zarówno 2D jak i 3D. Biblioteka została napisana w Javie. Modele w tej bibliotece są niezależne od komponentów odpowiadających za wizualizacje. Modele mogą być dynamicznie dodawane, usuwane lub zmieniane. Wyniki symulacji są niezależne od platformy, na której została uruchomiona. Dodatkowo wyniki symulacji mogą być zapisane w postaci zrzutów PNG, krótkich filmów lub wykresów.

# Charakterystyka projektu

Celem projektu jest stworzenie inteligentnego systemu oswietlenia ulicy oraz zasymulowanie jego działania dla kilku wybranych rzeczywistych scenariuszy. W implementacji zostanie wykorzystany opisany w poprzednim rozdziale framework JADE.

Podstawowe funkcje systemu Inteligentnego Oswietlenia Ulicy to:

* Sterowanie poszczególnymi latarniami ulicznymi.
* Włączenie, wyłączenie oraz ograniczenie mocy poszczególnych lamp.
* Automatyczne zmienianie natężenia oświetlenia w zależności od aktualnej pory dnia oraz warunków na drodze.
* Możliwość tworzenia niezależnych reguł dla latarni znajdujących się w miejscach wymagających szczególnej ostrożności, np. przejścia dla pieszych, szkoły.
* Prowadzenie statystyk o pracy poszczególnych elementów systemu.

# Identyfikacja Aktorów i Agentów w systemie

## Aktorzy systemu Inteligentnego Oswietlenia Ulicy

* Pieszy - osoba wchodząca w interakcje z systemem poprzez znalezienie się w zasięgu jego działania. Akcje podejmowane przez system są zależne od obecnego oraz historycznego zachowania pieszego w obrębie działania systemu. Dane dotyczące pieszego zostaną zarchiwizowane po każdej interakcji z systemem.
* Pojazd - środek lokomocji posiadajacy oznaczenia pozwalajace na jednoznaczna identyfikacje np. motocykl, samochód. Akcje podejmowane przez system są zależne od obecnego oraz historycznego zachowania pojazdu w obrębie działania systemu. Dane dotyczące pojazdu zostają zarchiwizowane po każdej interakcji z systemem.

## Agenci systemu Inteligentnego Oswietlenia Ulicy

* Czujnik światła naturalnego - Fotokomórka umieszczona na szczycie oprawy oświetleniowej – dostarcza informacje na temat poziomu światła naturalnego.
* Zegar astronomiczny - Na podstawie informacji o bieżącej dacie, współrzędnych geograficznych miejsca jego zainstalowania (lokalizacji) samoczynnie wyznacza dobowe, czasowe punkty załączenia i wyłączenia styku zegara zgodne z astronomicznymi czasami wschodu i zachodu słońca.
* Czujnik ruchu - Radarowe lub pasywne czujniki podczerwieni wykrywają pojawienie się użytkownika na naświetlanym obszarze. Czujnik wykrywa jedynie obiekty ważne dla systemu, pozostałe ignoruje.
* Czujnik prędkości i kierunku ruchu - Dostarczają informacji na temat prędkości i kierunku ruchu obiektów znajdujących się w obszarze detekcji co umozliwia realizowanie róznych scenariuszów oswietleniowych.
* Kamera przemysłowa - Dostarcza dane w postaci obrazu, na podstawie, którego dokonywana jest analiza względem identyfikacji zdarzeń oraz aktorów w systemie.

## Agenci identyfikujący zdarzenia

* Moduł analizy obrazu - Reaguje na bieżącą obserwację w oparciu o stan środowiska (agent reaktywny ze stanem). Agent przetwarza obserwacje w celu uzyskania danych identyfikujących aktorów znajdujących się w obszarze działania systemu. Aktorzy identyfikowani są w następujący sposób:

1. Pojazd - przeprowadzenia analizy tablicy rejestracyjnej zawartej na obserwacji.
2. Pieszy - identyfikacja twarzy aktora na podstawie analizy pomiarów (kształtu, owalu, rozstawu źrenic itp.)
3. Obserwacja jest pobierana od agenta ”Kamera przemysłowa”.

* Moduł analizy ruchu w systemie - Reaguje na bieżącą obserwację w oparciu o stan środowiska (agent reaktywny ze stanem). Agent przetwarza obserwacje w celu uzyskania danych identyfikujących ruch w systemie. Ustalane zostają parametry ruchu: kierunek, prędkość, przyspieszenie. Moduł wspierany jest przez ”Moduł analizy obrazu” co pozwala na identyfikację zdarzenia zależnego od aktora znajdującego się aktualnie w systemie. Obserwacja jest pobierana od agentów: ”Czujnik ruchu”, ”Czujnik prędkości i kierunku ruchu”, ”Kamera przemysłowa”.
* Moduł analizy parametrów lokalnych - Reaguje na bieżącą obserwację w oparciu o stan środowiska (agent reaktywny ze stanem). Agent przetwarza obserwacje w celu uzyskania danych identyfikujących zdarzenie aktywacji/dezaktywacji latarni w obszarze działania systemu. Obserwacja jest pobierana od agentów: ”Czujnik światła naturalnego”, ”Zegar astronomiczny”.

## Agenci magazynujący dane

Moduł integracyjny z baza danych Agent jest pośrednikiem pomiędzy agentami identyfikującymi zdarzenia oraz baza danych. Odpowiedzialny jest za migrację danych identyfikacyjnych do bazy danych. Baza danych Agent odpowiedzialny za archiwizację danych uzyskanych w systemie. Informacje przechowywane sa zgodnie z konwencja reprezentacji wiedzy w systemie Inteligentnego Oswietlenia Ulicy.

## Agenci wykonawczy (sterujący oświetleniem)

* Latarnia Fizyczna reprezentacja agenta wykonawczego. Sygnały sterujące są wysyłane do Latarni z ”Modułu sterującego”.
* Moduł sterujący - Agent odpowiadający za bezpośrednie sterowanie agentem ”Latarnia”. Sygnały sterujące są przekazywane do modułu sterującego z grupy agentów identyfikujących zdarzenia w systemie. Odebrane sygnały są weryfikowane z informacjami historycznymi zawartymi w bazie danych.

# Scenariusze oswietleniowe systemu Inteligentnego Oswietlenia Ulicy

## Niebezpieczny odcinek drogi

|  |  |
| --- | --- |
| Nazwa | Oświetlenie niebezpiecznego odcinka drogi. |
| Aktorzy | Pojazd |
| Opis | Na obszar objęty działaniem systemu wjeżdża pojazd, który kieruje sie w strone niebezpiecznego odcinka drogi. |
| Wyzwalacze | Detekcja pojazdu w obszarze działania systemu. |
| Warunki początkowe | Pojazd znajduje się w obszarze działania systemu. |
| Warunki końcowe | System Inteligentnego Oswietlenia Ulicy zwieksza poziom oświetlenia nad niebezpiecznym odcinkiem drogi. |

### Scenariusz Główny

1. Na obszarze objętym działaniem systemu Inteligentnego Oświetlenia Ulicy pojawia się nowy, ruchomy obiekt.
2. System identyfikuje nowy obiekt jako pojazd – samochód osobowy.
3. System mierzy prędkość samochodu i oblicza kiedy pojazd znajdzie się na niebezpiecznym odcinku drogi.
4. System zwiększa poziom oświetlenia nad niebezpiecznym zakrętem.

### Scenariusze alternatywne

1. Na obszarze objętym działaniem systemu Inteligentnego Oświetlenia Ulicy pojawił się nowy, ruchomy obiekt.
2. System identyfikuje nowy obiekt jako pojazd – samochód osobowy.
3. Systemowi nie udało się zmierzyć prędkości samochodu.
4. System zwiększa poziom oświetlenia na całym odcinku drogi.

## Pieszy na ulicy

|  |  |
| --- | --- |
| Nazwa | Oświetlenie drogi pieszego |
| Aktorzy | Pieszy |
| Opis | Na obszarze objętym działaniem systemu Inteligentnego  Oświetlenia Ulicy pojawi się pieszy i kieruje się w stronę  sklepików osiedlowych. |
| Wyzwalacze | Detekcja osoby obszarze działania systemu. |
| Warunki początkowe | Osoba znajduje się w obszarze działania systemu. |
| Warunki końcowe | System Inteligentnego Oswietlenia Ulicy zwieksza poziom oświetlenia trasie pokonywanej przez pieszego. |

### Scenariusz Główny

1. Na obszarze objętym działaniem systemu Inteligentnego Oświetlenia Ulicy pojawia się nowy ruchomy obiekt.
2. System identyfikuje nowy obiekt jako pieszego.
3. System wnioskuje, że pieszy zbliża się do przejścia dla pieszych
4. System zwiększa poziom oświetlenia na trasie osoby X oraz zmniejsza je na odcinku przez nią pokonanym.

### Scenariusze alternatywne

Scenariusz A

1. System Inteligentnego Oswietlenia Ulicy identyfikuje nowy obiekt jako pieszego.
2. System zbiera informacje na temat wyglądu/zachowania/kierunku ruchu pieszego.
3. Na podstawie zebranych informacji System wnioskuje, że pieszy będzie się poruszał wzdłuż drogi.
4. System zwiększa poziom oświetlenia na trasie osoby X oraz zmniejsza je na odcinku przez nią pokonanym.

Scenariusz B

1. System Inteligentnego Oswietlenia Ulicy identyfikuje nowy obiekt jako pieszego.
2. Osoba zmienia kierunek ruchu. System reaguje na zmianę kierunku przez osobę X (zwiększa poziom oświetlenia na nowo wybranej trasie).
3. System aktualizuje informacje na temat osoby X i nowej trasy przez nią wybranej.

## Wydarzenie socjalne

|  |  |
| --- | --- |
| Nazwa | Oświetlenia zdarzenia socjalnego. |
| Aktorzy | Pieszy |
| Opis | Na obszarze objętym działaniem systemu Inteligentnego  Oswietlenia Ulicy odbywa sie zdarzenie rozrywkowe w, którymudział bierze pieszy. |
| Wyzwalacze | Rozpoczęcie wydarzenia socjalnego. |
| Warunki początkowe | Pieszy znajduje się w obszarze działania systemu. |
| Warunki końcowe | Zakończenie wydarzenia. |

### Scenariusz Główny

1. Na obszarze objetym działaniem systemu Inteligentnego Oswietlenia Ulicy rozpoczyna sie wydarzenie socjalne.
2. System identyfikuje wzmozony ruch na obszarze swojego działania.
3. System wnioskuje wystapienie wydarzenia, na podstawie obserwacji. System zwieksza poziom oswietlenia na obszarze wystapienia wydarzenia oraz drogach prowadzacych do niego.
4. Wydarzenie socjalne konczy sie zgodnie z zaplanowanym harmonogramem.
5. System dostosowuje poziom oswietlenia do aktualnie prowadzonych obserwacji w systemie.

### Scenariusze alternatywne

Scenariusz A

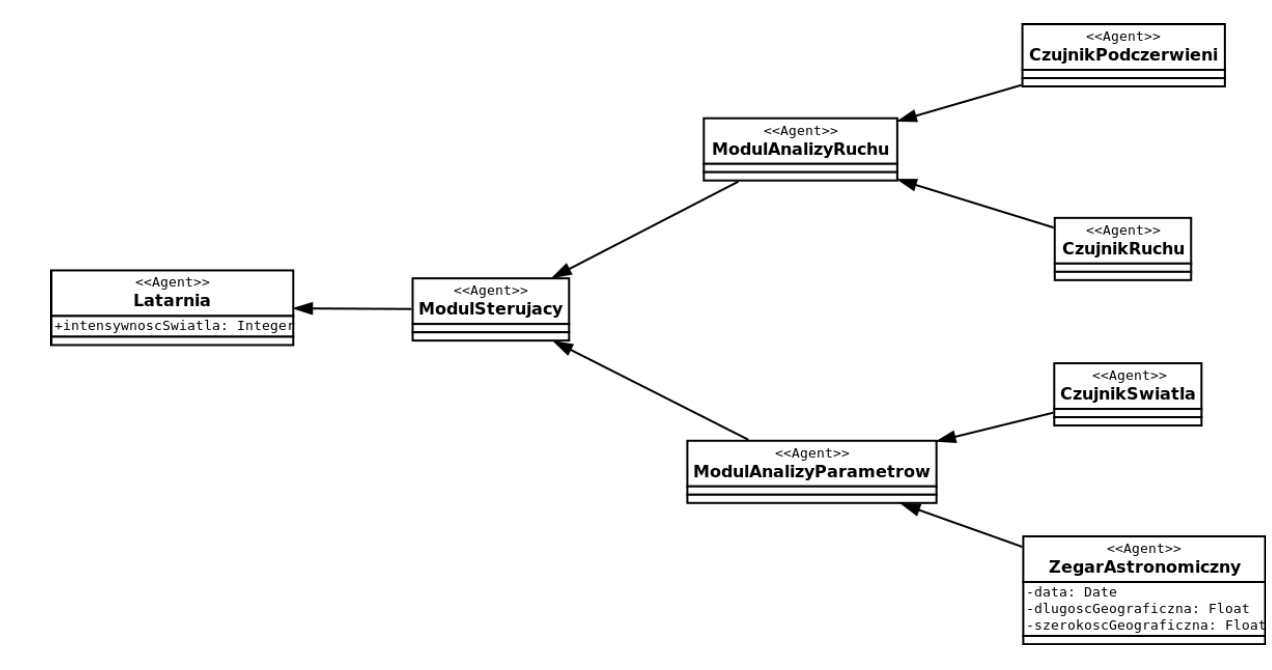
1. Na obszarze objętym działaniem systemu Inteligentnego Oświetlenia Ulicy rozpoczyna się wydarzenie socjalne.
2. System identyfikuje wzmożony ruch na obszarze swojego działania.
3. System wnioskuje wystąpienie wydarzenia, na podstawie obserwacji. System zwiększa poziom oświetlenia na obszarze występowania wydarzenia oraz drogach prowadzących do niego.
4. Wydarzenie socjalne kończy się rozbieżnie z zaplanowanym harmonogramem – złe warunki atmosferyczne, problemy organizacyjne.
5. System przerywa wysyłanie sygnałów sterujących odpowiadających wydarzeniu socjalnemu.
6. System powraca do stanu natężenia oświetlenia poprzedzającego wydarzenie socjalne.

Scenariusz B

1. Na obszarze objętym działaniem systemu Inteligentnego Oświetlenia Ulicy rozpoczyna się wydarzenie socjalne.
2. System podejmuje decyzje o braku podjęcia akcji, ze względu na panujące warunki oświetleniowe (dostarczające oświetlenie, aby wydarzenie zostało poprowadzone, bez pomocy światła sztucznego).
3. System kontynuuje działanie w trybie sprzed identyfikacji warunków oświetleniowych.

# Diagramy komunikacji pomiędzy agentami

Poniżej przedstawiony jest schemat wymiany informacji pomiędzy poszczególnymi agentami obecnymi w symulacji.



Rysunek 1: Wymiana informacji pomiędzy agentami

Każdej latarni w systemie przyporządkowany jest zestaw czujników:

* czujnik prędkości - czujnik skierowany na drogę, wykrywa poruszające się obiekty
* czujnik podczerwieni - czujnik wykrywający przechodniów, w odróżnieniu od czujnika predkosci, moze wykryc także nieruchoma osobę
* czujnik światła - czujnik mierzący natężenie światła w otoczeniu danej latarni

Na diagramie jako agent dostarczający dane, został przedstawiony także zegar astronomiczny. W odróżnieniu od wymienionych wcześniej czujników, występuje tylko jeden zegar astronomiczny w całym systemie. Dostarcza on informacji o bieżącej godzinie i dacie symulacji co pozwala na wyznaczenie przybliżonego czasu wschodu i zachodu słońca. W tym celu brane są także pod uwagę współrzędne geograficzne miejsca, w którym został zainstalowany system.

Kolejną warstwę przepływu danych stanowią moduły. Ich zadaniem jest przetworzenie danych dostarczonych przez agentów zbierających dane. Kolejno są to:

* moduł analizy ruchu - zbiera i analizuje dane otrzymane od czujników wykrywających obiekty istotne z punktu widzenia symulacji (pojazdy oraz pieszych)
* moduł analizy parametrów - zbiera i analizuje dane dotyczące otoczenia symulacji
* moduł sterujący - odbiera wstępnie przetworzone dane z pozostałych modułów oraz na tej podstawie wyznacza moc z jaką powinny świecić poszczególne latarnie

# Zastosowane rozwiązania techniczne

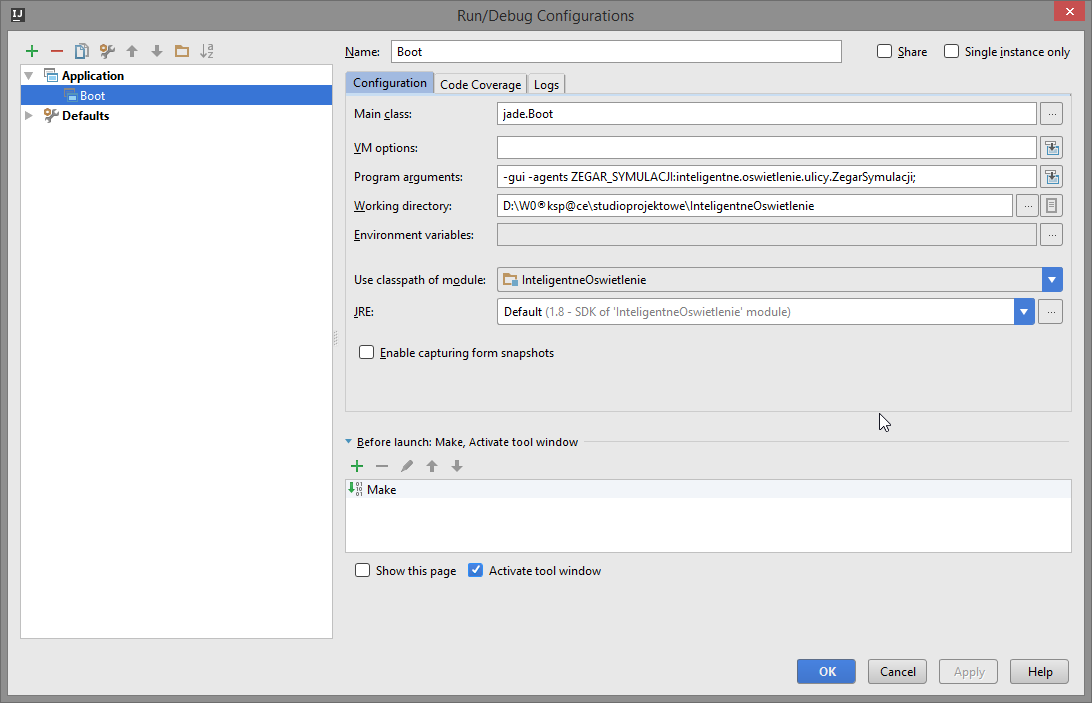
Wszyscy agenci są tworzeni hierarchicznie, począwszy od agenta Zegar Symulacji. Pierwszy agent jest uruchamiany poprzez bezpośrednie wywołanie frameworka z konsoli systemowej poleceniem:

*java -cp libs/jade.jar;libs/jgraphx-2.0.0.1.jar;libs/jdatepicker-1.3.4.jar;bin jade.Boot -gui -agents ZEGAR\_SYMULACJI:inteligentne.oswietlenie.ulicy.ZegarSymulacji;*

Ponadto podczas uruchamiania frameworku należy dodać do zmiennej CLASSPATH ścieżki do wszystkich bibliotek, z których korzysta aplikacja.

Dla konfiguracji środowiska programistycznego można przyjąć za argumenty programu :

“-gui -agents ZEGAR\_SYMULACJI:inteligentne.oswietlenie.ulicy.ZegarSymulacji;”



Rysunek 2: Schemat konfiguracji rozruchowej w programie InteliJ IDEA

Uruchomienie agentów odbywa się poprzez wypisanie po przełączniku

-agents par NAZWA AGENTA:PEŁNA NAZWA KLASY AGENTA.

Kolejni agenci są tworzeni za pośrednictwem istniejących już agentów. Można to osiągnąć za pomocą następującego kodu Java:

|  |
| --- |
| 1. AgentContainer kontener = getContainerController(); 2. AgentController kontroler = kontener.createNewAgent(NAZWA\_AGENTA, 3. PEŁNA\_NAZWA\_KLASY\_AGENTA, **null**); 4. kontroler.start(); |

Funkcja getContainerController() służy do pobrania kontrolera do kontenera, w którym znajduje się obecny agent. Tworzenie wszystkich agentów w jednym kontenerze znacznie ułatwia prace, poniewaz nie wymaga konfiguracji interfejsów sieciowych od wszystkich agentów, w celu umożliwienia komunikacji między nimi. Następnie w obrębie kontenera tworzy się nowego agenta za pomocą metody createNewAgent(). Pierwszym parametrem jest nazwa agenta, a drugim pełna nazwa klasy agenta. Trzeci parametr jest opcjonalny i służy do przekazywania informacji, potrzebnych podczas inicjalizowania nowego agenta. Po utworzeniu agenta należy go jeszcze uruchomić za pomocą funkcji start() wywołanej na kontrolerze kontenera.

Każdy z agentów dostarczających usługi w pierwszej kolejności rejestruje je u agenta

DFService.

|  |
| --- |
| **private void** rejestracjaAgenta() {  DFAgentDescription dfd = **new** DFAgentDescription();  dfd.setName(getAID());  ServiceDescription sd = **new** ServiceDescription();  sd.setName(getLocalName());  sd.setType(**"Latarnia"**);  dfd.addServices(sd);  **try** {  DFService.*register*(**this**, dfd);  }  **catch** (FIPAException fe) {  fe.printStackTrace(System.***out***);  }  } |

Każdy z agentów może wyszukać agenta, do którego wysyła przetworzone dane. Wyszukiwanie odbiorców odbywa się z wykorzystaniem tego samego obiektu.

|  |
| --- |
| **private void** znajdzModulAnalizyRuchu() {  DFAgentDescription template = **new** DFAgentDescription();  ServiceDescription sd = **new** ServiceDescription();  sd.setType(**"Modul Analizy Ruchu"**);  template.addServices(sd);  **try** {  DFAgentDescription[] result = DFService.*search*(**this**, template);  **if** (result.**length** == 0){  **throw new** Exception(**"Nie znaleziono modu�u analizy ruchu"**);  }  **modulAnalizyRuchu** = result[0].getName();  } **catch** (Exception e) {  e.printStackTrace(System.***out***);  }  } |

Wysłanie danych do agenta opiera się na automatycznej serializacji obiektów. Dodatkowo każdej wiadomości przyporządkowane jest pole informujące o typie wiadomości.

|  |
| --- |
| ACLMessage wiadomoscDoGUI = **new** ACLMessage(ACLMessage.***INFORM***);  String numer = **myAgent**.getLocalName().replace(Konfiguracja.*przedrostekNazwAgentowCzujnikNatezeniaSwiatla*, **""**);  wiadomoscDoGUI.setContent(**"NATEZENIE\_OSWIETLENIA@"** + numer + **"#"** + **natezenieOswietlenia**);  wiadomoscDoGUI.addReceiver(AgentInterfejsu.*agentInterfejsu*.getAID());  **myAgent**.send(wiadomoscDoGUI); |

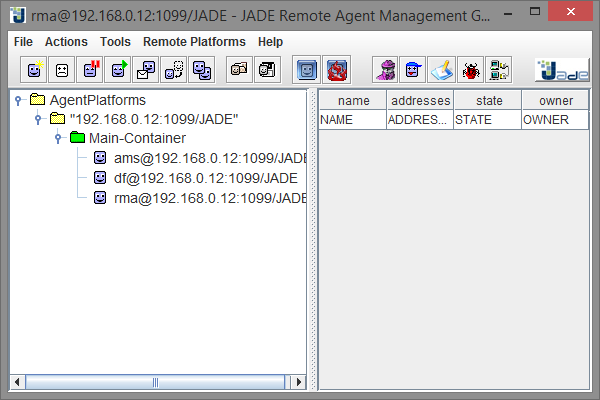
Odbieranie danych polega na wyciąganiu z kolejki wiadomości, które pasują do wzorca. Jeżeli nie istnieje taka wiadomość można wywołać metodę block(), dzięki której agent zostanie uśpiony aż do przyjścia nowej wiadomości.

|  |
| --- |
| **public void** action() {  MessageTemplate mt = MessageTemplate.*MatchPerformative*(ACLMessage.***INFORM***);  ACLMessage msg = **myAgent**.receive(mt);  **if** (msg != **null**) {  **try** {  **intensywnoscSwiatla** = (Integer) msg.getContentObject();    } **catch** (UnreadableException e) {  e.printStackTrace(System.***out***);  }  }  **else** {  block();  }  przekazIntensywnoscDoGUI();  } |

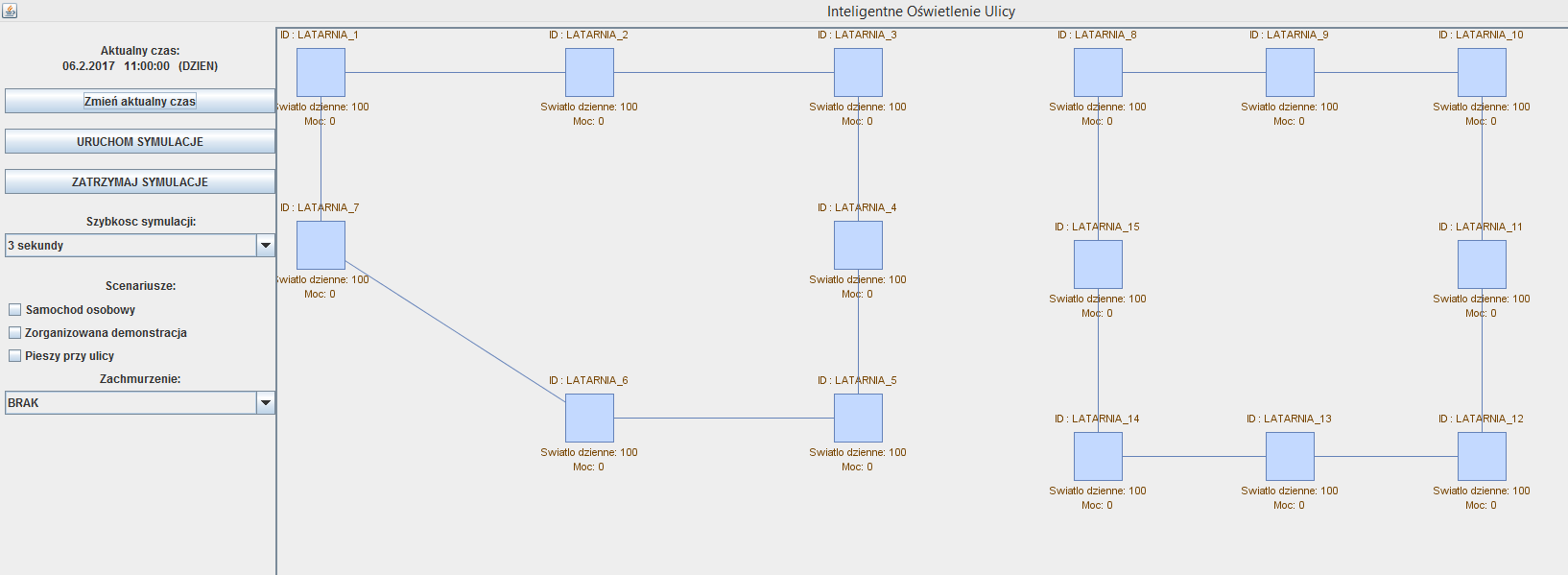
Interfejs programu został oparty na bibliotece Swing. Dodatkowo grafy reprezentujące

wyniki symulacji są rysowane z wykorzystaniem biblioteki JGraphX. Ułatwia ona rysowanie grafów i różnego rodzaju schematów blokowych.

## Opis interfejsu programu



Rysunek 3: GUI JADE



Rysunek 4: Ekran początkowy - okno symulacji

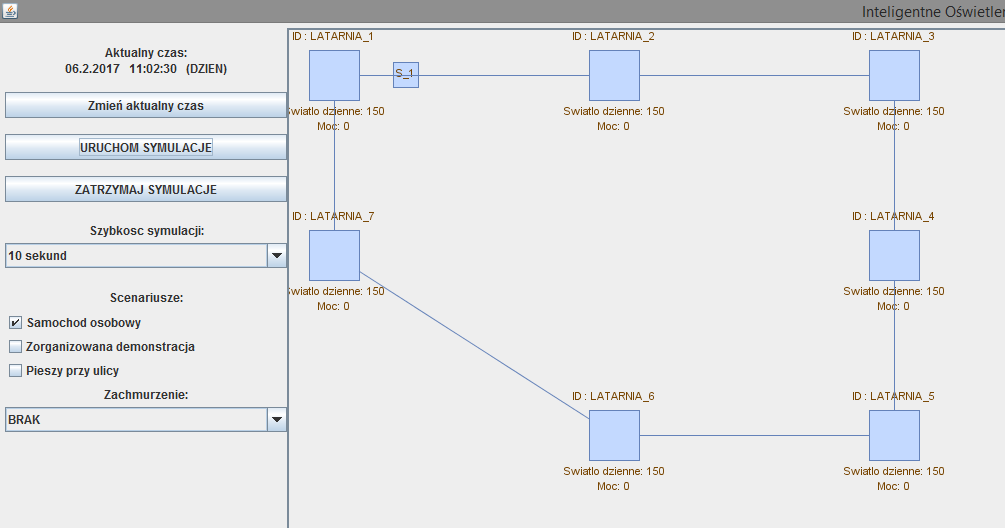
Po uruchomieniu programu pojawia się okno z interfejsem graficznym pozwalającym na zmianę parametrów oraz śledzenie postępu symulacji. W Centralnej części okna narysowany jest graf symulacji, gdzie wierzchołki odpowiadają kolejnym latarniom, natomiast krawędzie symbolizują drogi po których poruszają się różne obiekty. Każda z latarni posiada dwa parametry:

* pierwszy z nich oznaczony literą ”N” dostarcza informacji o natężeniu światła w otoczeniu danej latarni
* drugi oznaczony przez ”M” pokazuje z jaką intensywnością świeci aktualnie dana latarnia.

Natężenie światła uzależnione jest od aktualnego czasu symulacji oraz warunków pogodowych. Zachmurzenie można w kazdej chwili modyfikować wybierając odpowiednia wartosc z listy rozwijalnej po lewej stronie. Mamy do wyboru warunki zmieniające się od braku do pełnego zachmurzenia. Na pasku bocznym z lewej strony przedstawione są dane dotyczące aktualnego stanu symulacji. W każdej chwili można zmienić szybkość symulacji wybierając odpowiednia wartosc z rozwijanej listy od bardzo wolnej (3 sekundy w symulacji odpowiadają 1 sekundzie w świecie rzeczywistym) do bardzo szybkiej (2 godziny w symulacji trwają 1 sekundę w

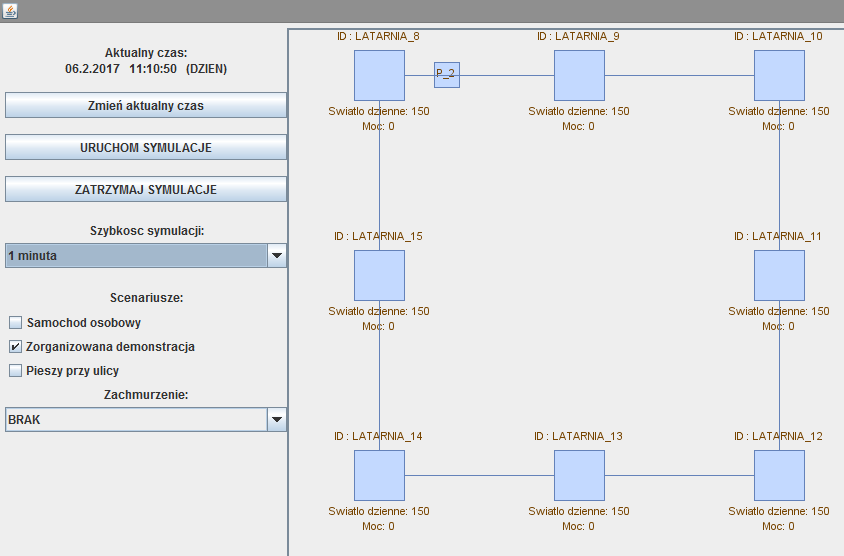
świecie rzeczywistym). Do wyboru mamy dostepne takze 3 zdefiniowane scenariusze. Są to kolejno :

* samochód osobowy - samochód podróżuje ze stałą prędkością poruszając się po krawędziach, zgodnie z ruchem wskazówek zegara.



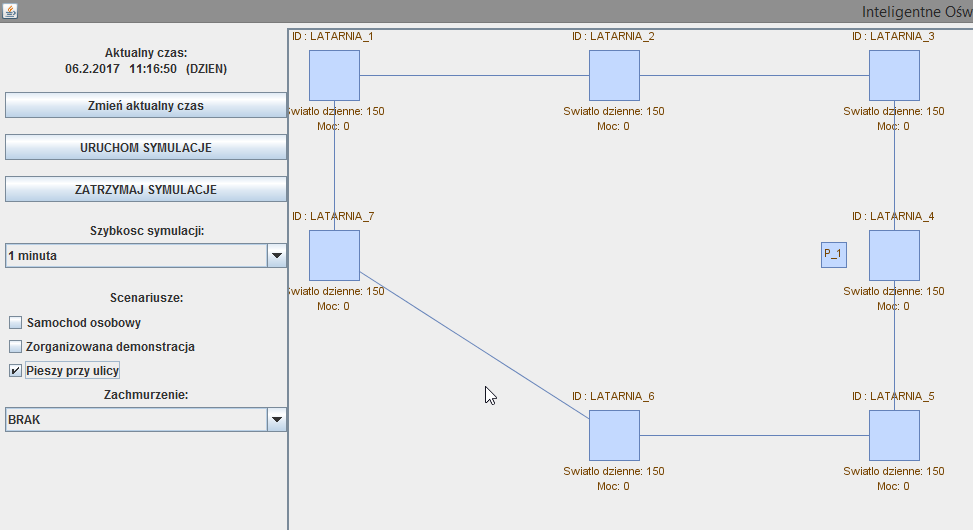
Rysunek 5: Samochód osobowy

* zorganizowana demonstracja - podobnie jak samochód demonstracja przemieszcza się po krawędziach grafu. Demonstracja wykorzystuje prawy graf widoczny w oknie.



Rysunek 6: Zorganizowana demonstracja

* pieszy przy ulicy - pieszy pojawia się przy jednej z latarni i pozostaje nieruchomy przez cały czas symulacji.



Rysunek 7: Pieszy przy ulicy

# Opis Loggera do wypisywania czasu reakcji.

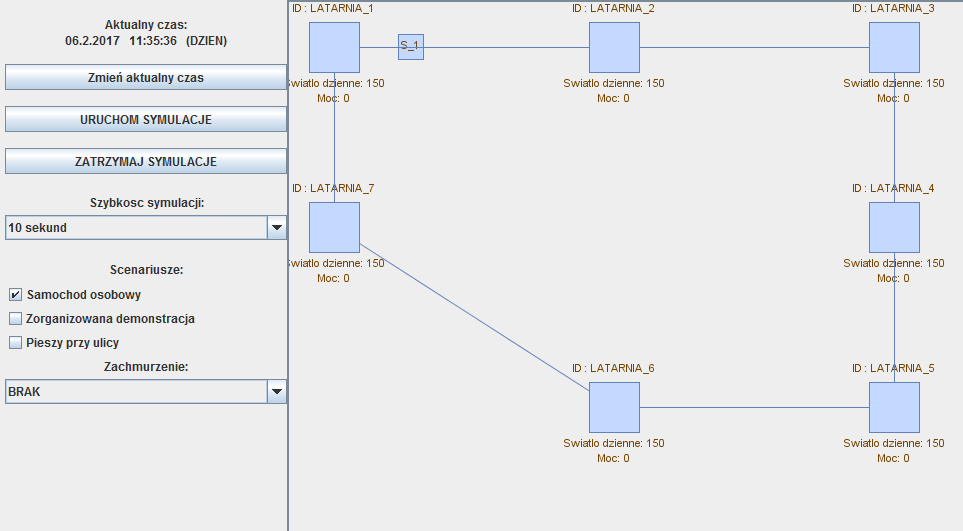
W celach dokumentacji prowadzonych symulacji, przy użyciu biblioteki org.pmw.tinylog.Logger,zaimplementowany został moduł do raportowania czasu reakcji na zmianę scenariusza. Nie było możliwe wykrycie zmiany zachmurzenia ponieważ, sama zmiana jest zachmurzenia jest zaimplementowana w oparciu o obsługę zdarzeń, podobnie jak zmiana szybkości symulacji. W logach konsolowych zawarty został czas uruchomienia symulacji, czas inicjalizacji programu, thread, na którym było wywołane logowanie - widać przez to, że różni agenci posiadają swoje własne wątki.

|  |
| --- |
| Inicjalizacja programu, nazwa threadu w nawiasach []  2017-02-05 19:32:45.606 [ZEGAR\_SYMULACJI] inteligentne.oswietlenie.ulicy.ZegarSymulacji.setup()  INFO: Start simulation initialization  2017-02-05 19:32:47.166 [ZEGAR\_SYMULACJI] inteligentne.oswietlenie.ulicy.ZegarSymulacji.setup()  INFO: End of simulation initialization, time spend initialization = 1561 ms |
| Inicjalizacja symulacji  017-02-05 19:32:48.484 [AWT-EventQueue-0] inteligentne.oswietlenie.ulicy.OknoGlowne.mouseReleased()  INFO: Start simulation  2017-02-05 19:32:52.485 [AWT-EventQueue-0] inteligentne.oswietlenie.ulicy.OknoGlowne.mouseReleased()  INFO: End of starting simulation, time spend: 4001ms |
| Zmiana trybu symulacji  2017-02-05 19:32:53.254 [AWT-EventQueue-0] inteligentne.oswietlenie.ulicy.OknoGlowne.lambda$new$1()  INFO: Change simulation option started  ON: TRASA\_1  2017-02-05 19:32:53.708 [AGENT\_INTERFEJSU] inteligentne.oswietlenie.ulicy.ObslugaKomunikatowInterfejsu.onTick()  INFO: Change simulation option finished |
| Log na czujnikach w INFO: [zawartość wiadomości]  2017-02-05 19:32:54.711 [CZUJNIK\_PREDKOSCI\_2] inteligentne.oswietlenie.ulicy.SterownikCzujnikaPredkosci.action()  INFO: ODLEGLOSC\_DO\_POJAZDU@291.639884960888  2017-02-05 19:32:54.711 [CZUJNIK\_PREDKOSCI\_1] inteligentne.oswietlenie.ulicy.SterownikCzujnikaPredkosci.action()  INFO: ODLEGLOSC\_DO\_POJAZDU@17.257534586377048  2017-02-05 19:33:33.021 [CZUJNIK\_PREDKOSCI\_4] inteligentne.oswietlenie.ulicy.SterownikCzujnikaPredkosci.action()  INFO: ODLEGLOSC\_DO\_POJAZDU@464.2913982619105  2017-02-05 19:33:33.021 [CZUJNIK\_PREDKOSCI\_8] inteligentne.oswietlenie.ulicy.SterownikCzujnikaPredkosci.action()  INFO: ODLEGLOSC\_DO\_POJAZDU@213.36940385163007  2017-02-05 19:33:47.077 [CZUJNIK\_RUCHU\_4] inteligentne.oswietlenie.ulicy.SterownikCzujnikaRuchu.action()  INFO: ODLEGLOSC\_DO\_LATARNI@60.0  CZUJNIK\_RUCHU\_4:60.0  2017-02-05 19:33:48.081 [CZUJNIK\_RUCHU\_4] inteligentne.oswietlenie.ulicy.SterownikCzujnikaRuchu.action()  INFO: ODLEGLOSC\_DO\_LATARNI@60.0  CZUJNIK\_RUCHU\_4:60.0 |

# 

# Wyniki symulacji

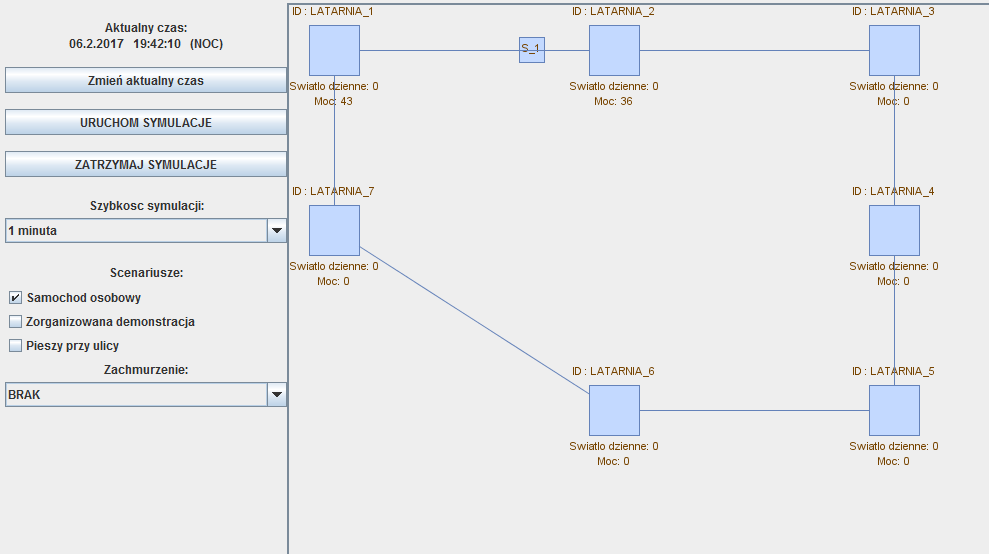
## Scenariusz 1: samochód osobowy - brak zachmurzenia - Dzień



Rysunek 8: Zrzut ekranu aplikacji

Scenariusz dla przemieszczającego się samochodu osobowego “s\_1” poruszającego się pomiędzy obiektami LATARNIA\_1, LATARNIA\_2. Moc na wszystkich latarniach jest utrzymywana na poziomie zerowym, ze względu na bardzo wysoką wartość natężenia światła dziennego. Ta zaś spowodowana jest porą dnia, w której słońce znajduje się “najwyżej na niebie” (najbliżej zenitu).

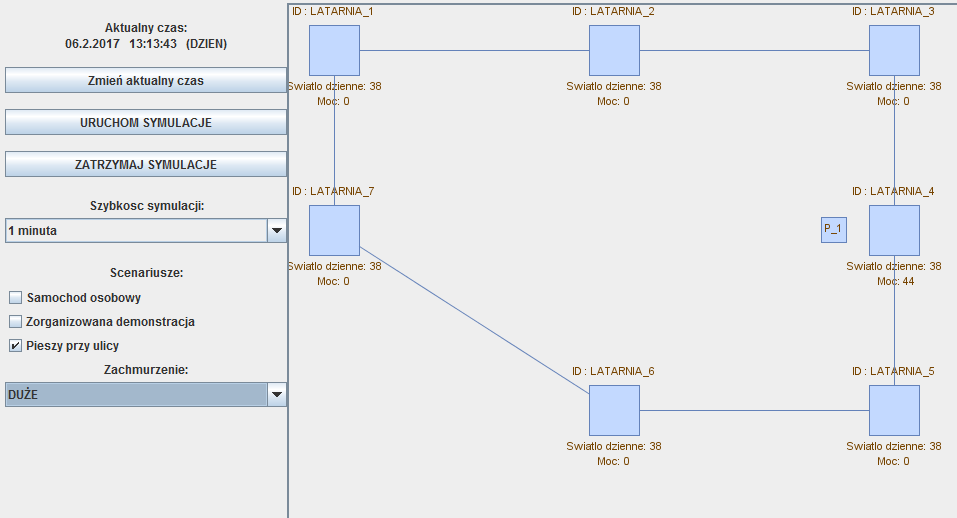
## Scenariusz 2: samochód osobowy - brak zachmurzenia - Noc



Rysunek 9: Zrzut ekranu aplikacji

Samochód osobowy “s\_1” poruszający się po drodze, powoduje, że kolejne latarnie zmieniają moc z jaką świeca. Latarnia od której samochód się oddala, zwiększa w sposób liniowy natężenie światła, natomiast latarnia, do której pojazd się zbliża - zmniejsza. Wynikiem tego, jest w przybliżeniu jednakowe, stałe natężenie światła na całej drodze samochodu.

## Scenariusz 3: pieszy przy ulicy - Duże zachmurzenie - Dzień



Rysunek 10: pieszy przy ulicy

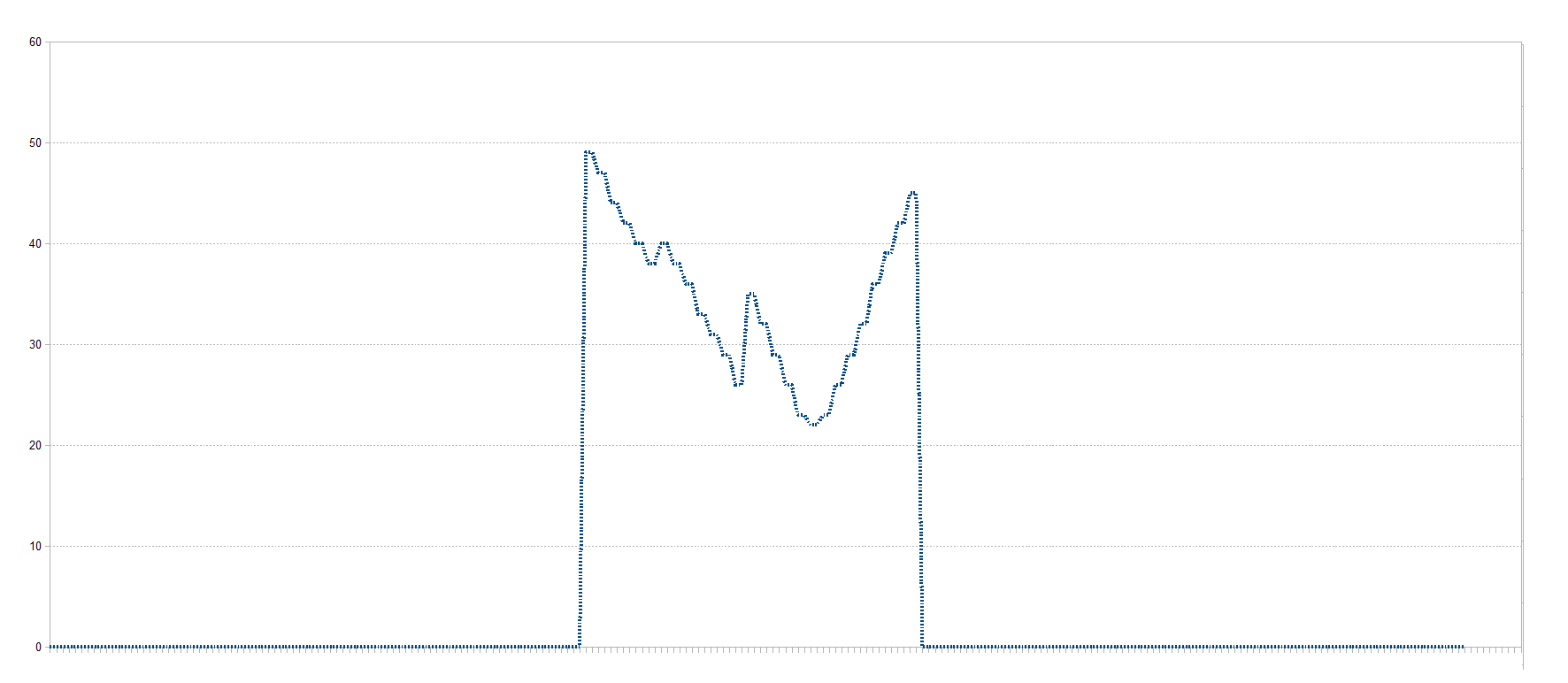
Pieszy stojąc przy ulicy blisko latarni ze względu na duże zachmurzeniem, jest wystarczająco mocno oświetlony. Uwzględniając poziom natężenia światła system wyznaczył moc oświetlenia najbliższej do obiektu P\_1 latarni na 44% mocy, jednocześnie wyłączając pozostałe latarnie.

Obrazowanie wyników na diagramach

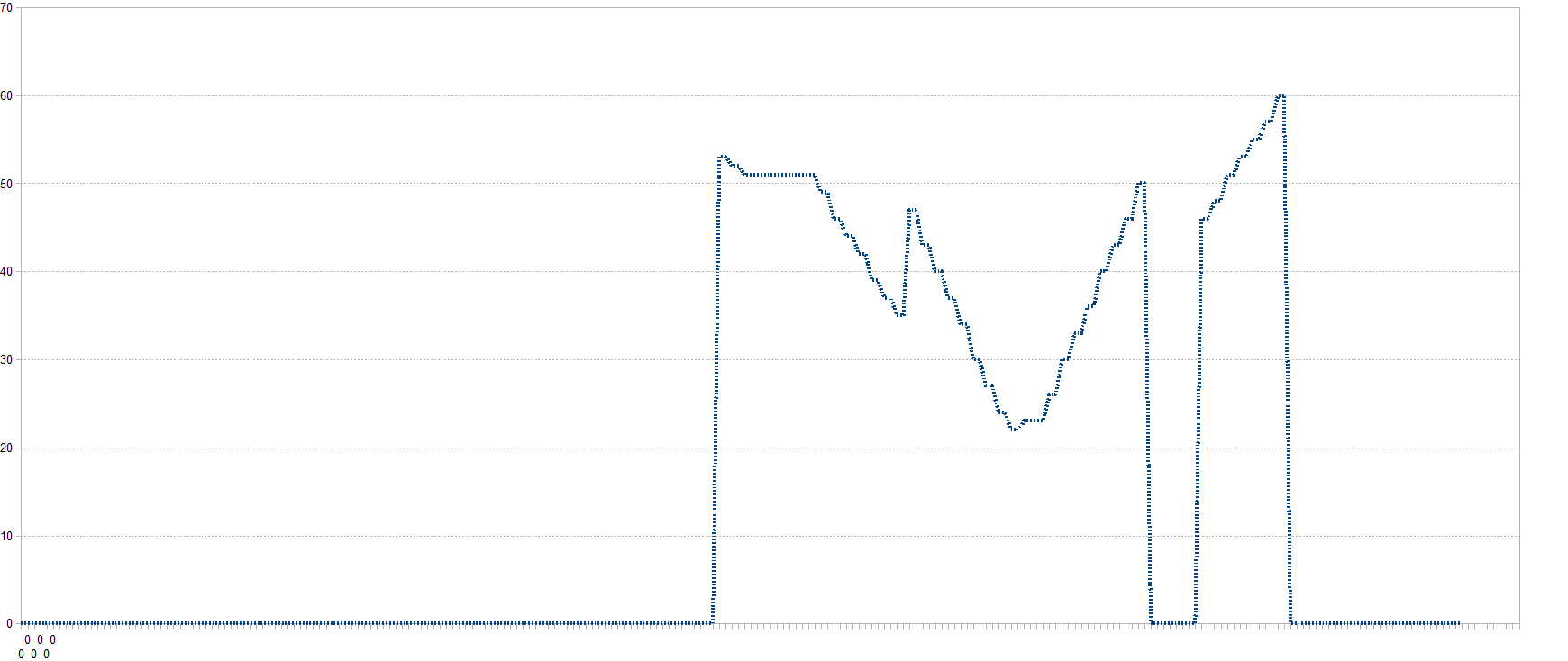
Dane zilustrowane na wykresach zamieszczonych w tym podrozdziale były zbierane w okresie równym jednemu cyklowi podróży samochodu wzdłuż wszystkich krawędzi. Na poniższych wykresach zilustrowane są zmiany natężenia światła dla poszczególnych latarni obecnych w symulacji. Pozostałe, niezamieszczone wykresy wyglądają analogicznie. Jedyną różnicą będzie ich przesunięcie w czasie.



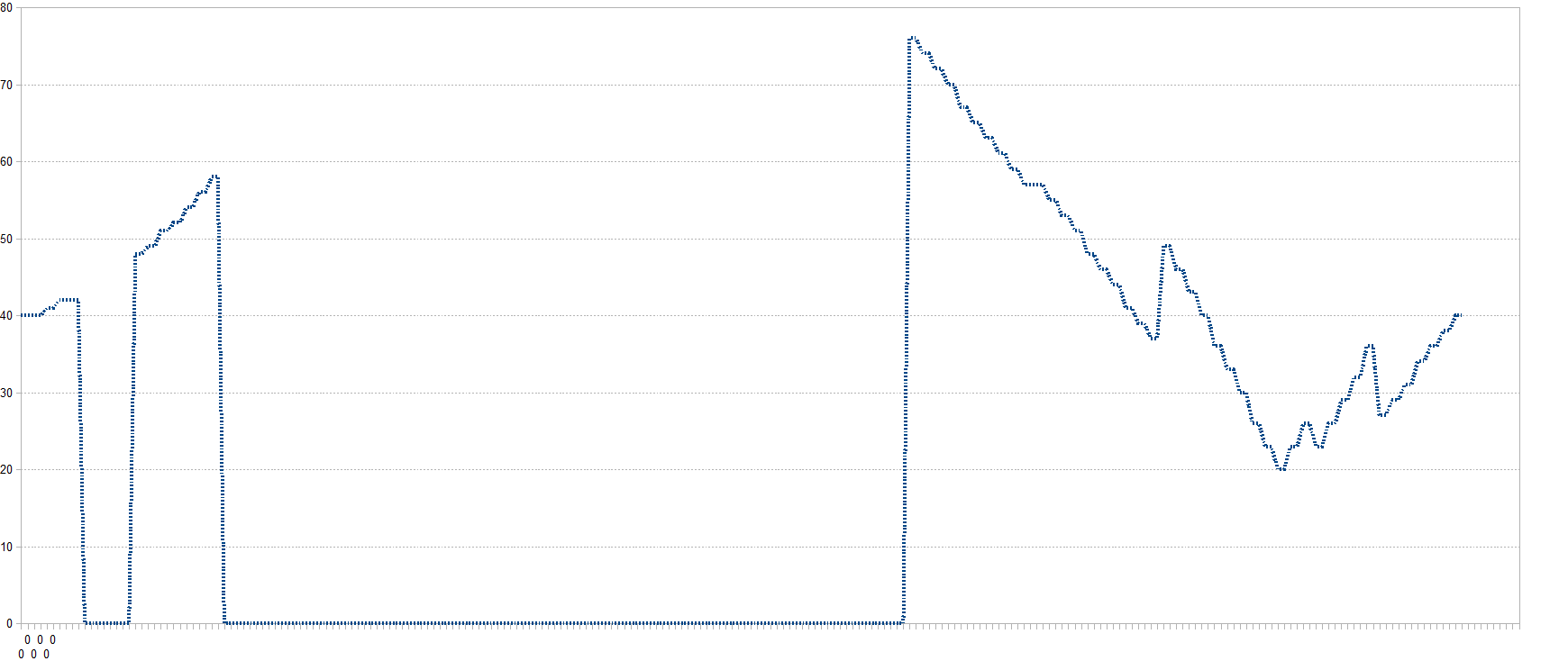
Rysunek 11: wykres natężenia światła w zależności od czasu



Rysunek 12: wykres natężenia światła w zależności od czasu

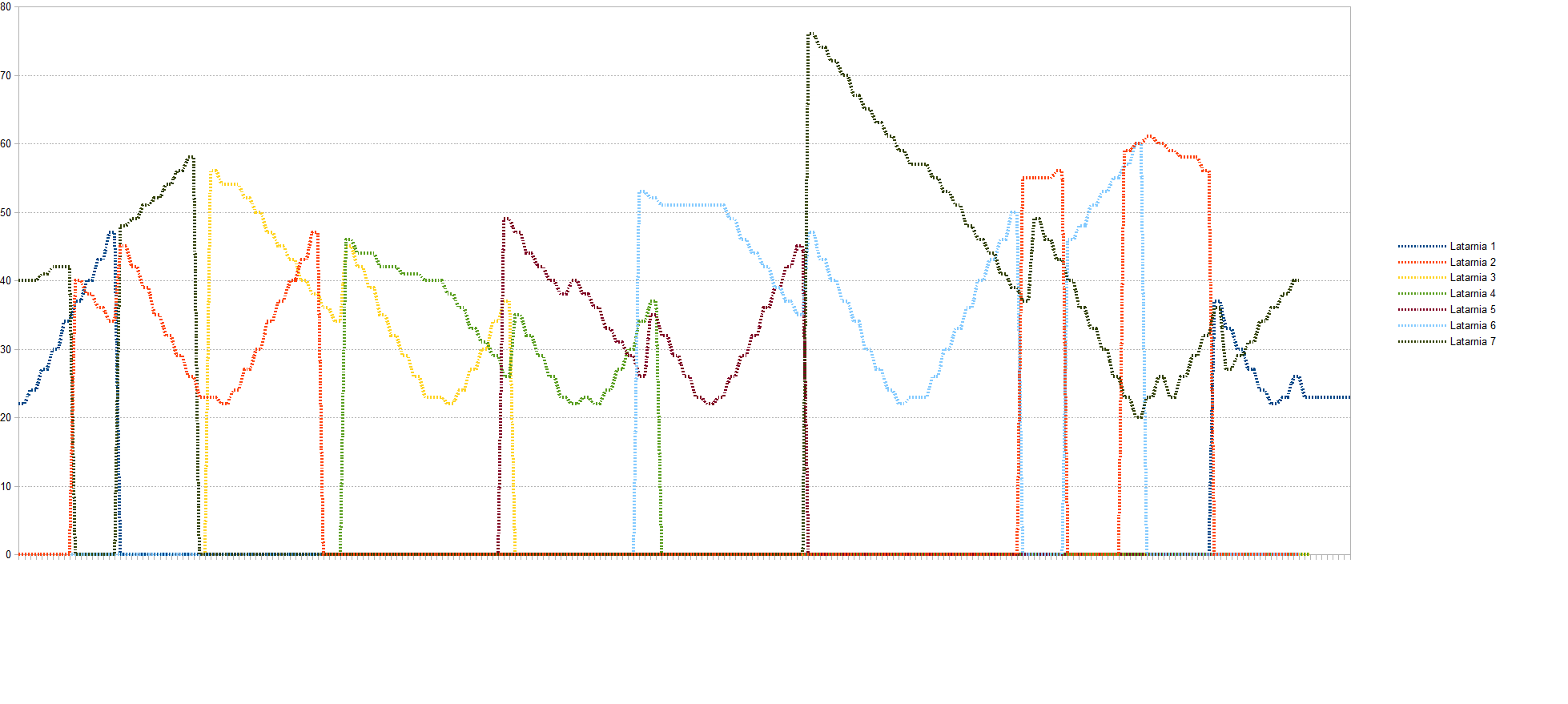


Rysunek 13: wykres natężenia światła w zależności od czasu



Rysunek 14: wykres natężenia światła w zależności od czasu

Różne przebiegi na wykresach spowodowane są przez konfiguracje latarni na przykładowym schemacie. Na kolejnym wykresie zostały zebrane dane ze wszystkich latarni, aby zobrazowac wzgledna zmiane natezenia oswietlenia dla kazdej z nich.



Rysunek 15: wspólny wykres natężenia światła w zależności od czasu.

Łatwo mozna zauwazyc, ze kazda z latarni działa znacznie efektywniej niż gdyby świeciła ze stałą wartością natężenia światła równa 50% jej maksymalnej wartości. Mimo to przejezdzajacy kierowca doswiadcza znacznie lepszego oswietlenia ulicy i pobocza.

# Słownik pojęć

**Agent** - jest to każdy obiekt (jednostka), który obserwuje (odbiera) swoje otoczenie dzięki czujnikom (sensorom) i oddziałuje na to środowisko przy pomocy „efektorów” (wykonuje akcje).

**Moduł** - fragment systemu polegający na wykonaniu wyodrębnionego zadania programistycznego.

Każdy moduł ma ściśle określony interfejs- zestaw obiektów programistycznych,

które realizuje. Jedne moduły mogą korzystać z obiektów zaimplementowanych przez inne.

# Podsumowanie pracy z Frameworkiem JADE

Rozpoczęcie pracy z Frameworkiem JADE nie stwarza wiele problemów dzieki bardzo czytelnemu tutorialowi, który można znaleźć na stronie projektu. Opisano w nim zasadę działania systemów agentowych oraz pokazano na przykładach krótkich fragmentów kodu, jak te zadania są realizowane za pośrednictwem tego frameworku. Ponadto w internecie można znaleźć wiele przykładowych projektów przedstawiających podstawowa funkcjonalność JADE.

Początki pracy z frameworkiem ułatwia graficzna nakładka, pozwalająca uruchomić,

wysyłać wiadomości, testować i sprawdzać stan poszczególnych agentów niezależnie od pozostałych. Problem zaczyna się dopiero, gdy chce się skorzystać z bardziej zaawansowanych funkcjonalności, takich jak uruchamianie frameworku bezpośrednio z kodu Javowej aplikacji, integrowanie z innymi bibliotekami, czy hierarchiczne tworzenie kolejnych agentów. Tutorial opisuje tylko podstawowa funkcjonalność, a szukając pomocy w internecie można znaleźć wiele postów bez odpowiedzi opisujących podobne problemy. Wynika to z faktu

stosunkowo małej społeczności korzystającej i rozwijającej JADE. Często jedyna skuteczna metoda rozwiązywania problemów, było kolejne testowanie wszystkich możliwych konfiguracji. Po poprawnym uruchomieniu wszystkich agentów i ustalenia sposobu komunikacji między nimi, kolejnym, największym problemem jest zadbanie o poprawność logiki całego systemu. Tworzenie dużego systemy zawsze wiąże się z błędami, które zazwyczaj wykrywa się za pomocą Debuggera, jednak JADE nie dostarcza Debuggera, którego by można zintegrować z środowiskiem programistycznym i prześledzić działanie aplikacji. Dlatego jedyna metoda śledzenia wykonywania programu był drukowanie numerów linijek i wartości poszczególnych zmiennych. Bardzo to spowalnia pracę z frameworkiem, szczególnie podczas integrowania z innymi bibliotekami. Dlatego JADE świetnie się nadaje do zrozumienia idei Frameworków wspomagających budowę Systemów Agentowych, jednak nie jest najlepszym rozwiązaniem, do tworzenia rozbudowanej aplikacji, korzystającej z wielu bibliotek.

# Podsumowanie

Podczas pracy nad projektem został przeprowadzony refaktoring kilku wybranych klas i mechanizmów, ponieważ część dostarczonego projektu była wykonana bardzo nieczytelnie oraz bez zrozumienia rozwiązywanego problemu. Refaktoring dotyczy również elementów graficznych aplikacji (GUI). Przywiązywaliśmy wagę do poprawy widoczności danych wyświetlanych podczas symulacji oraz wyróżnienia zależności pomiędzy jej elementami. Ponadto do projektu został dodany moduł logujący zdarzenia, zrealizowany przy pomocy darmowej biblioteki “tinylog”. Bibliotekę zastosowaliśmy do :

* Logowania zdarzeń, które nie wchodzą w skład uruchamianych co tick zegara zdarzeń.
* Ustanowienia poprawnego formatowania czasu zdarzeń w postaci : {czas\_symulacji} [{wątek\_logowania}] {funkcja\_wywołująca} {poziom} {msg}

opcjonalnie można zalogować czas trwania zdarzenia.

#### Napotkane problemy

* Kolejka zdarzeń frameworka JADE oparta jest o java.awt.EventQueue przez co nie było możliwe pełne logowanie właściwej obsługi wszystkich zdarzeń systemu agentowego (komunikacja między agentami)
* Framework JADE powszechnie wykorzystuje mechanizm refleksji, co w przypadku systemów wieloagentowych znacznie utrudnia prace z kodem. Agenci tworzeni są jako odwołanie do rzeczywistego miejsca w pakietach Javy, nawet najnowsze wersję popularnych IDE (jak Intellij IDEA) nie radzą sobie z rozpoznaniem czy dana klasa jest używana w projekcie czy też nie.
* Kod programu jest napisany bardzo niechlujnie, nie ma śladów czystego kodu, brak śladów stosowania jakiejkolwiek konwencji architektonicznej oraz konwencji nazewnictwa. W całej aplikacji pojawiają się fragmenty typu :

|  |  |
| --- | --- |
| **if** (struktCzas.**jestDzien**) {  dzienLubNoc = **"DZIEN"**;  } **else** {  dzienLubNoc = **"NOC"**;  } | setIntensywnoscSwiatla(**new** Integer(0)); |

* Forma obsługi systemu agentowego w frameworku JADE może się sprawdzać przy mniejszych projektach, jednakże w przypadkach bardziej skomplikowanych pojawiają się problemy z tzw. Event Busami - objaśniając, programiście w pewnym momencie ciężko stwierdzić, który obiekt rozmawia z którym, oraz gdzie występują jakiekolwiek interakcję między nimi.

#### Co jeszcze do zrobienia

Naszym zdaniem, jedynym sensownym rozwiązaniem jest napisanie projektu od nowa, jedynie wzorując się na poszczególnych jego sekcjach. Warto byłoby rozejrzeć się za bardziej przyjaznymi frameworkami do obsługi systemów wieloagentowych.