**Skład zespołu J:**

**Prowadzący:**

Kamil Hajduk

Tomasz Śliwiński

Michał Bogucki

Kamil Cichończyk

Michał Andruszkiewicz

**System czujników w węzłach komunikacyjnych**

Projekt Wieloagentowe Systemy Decyzyjne

**Rozdział I – część A**

* Identyfikacja problemu

Życie w wielkim mieście posiada niewątpliwie wiele zalet. Niestety, istnieją też istotne wady. Jedną z nich jest tłok w komunikacji miejskiej, z którym zmagamy się na co dzień. Te duże lokalne skupiska ludzi mają niewykorzystany potencjał (np. reklamowy).

* Propozycja i sprecyzowanie rozwiązania

Propozycja rozwiązania wyżej postawionego problemu polega na umieszczeniu w podłożach węzłów komunikacji miejskiej czujników, których zadaniem jest zliczanie pasażerów w czasie rzeczywistym. Informacja taka pozwoliłaby określić miejsca, w których następuje wzmożony ruch ludzi i umożliwiłaby podjęcie odpowiednich kroków w celu „rozładowania tłumu”. Dodatkową możliwośd jaką oferują chwilowe, duże skupiska ludności, jest użycie tzw. systemu real time bidding. Polegałby on na umożliwieniu klientom (firmom wszelkiego rodzaju) wykupowanie przestrzeni reklamowej w węzłach komunikacyjnych.

* Opis koncepcji systemu

W proponowanym rozwiązaniu problemu, system oparty jest na architekturze wieloagentowej. Zadaniem pierwszego typu agenta jest zbieranie informacji z czujników, dotyczące aktualnej liczby pasażerów w danych węzłach komunikacyjnych. Drugi agent przewiduje zagęszczenia na podstawie zebranych danych. Kolejny typ agenta odpowiedzialny jest za przyjmowanie ofert i wyświetlanie reklam klientów chcących promować swój produkt w danym miejscu i czasie. Ostatnim agentem byłby typ dyspozytora ruchu, który dąży do udrożniania przepływu pasażerów (np. poprzez wysyłanie w dane miejsce większej ilości środków komunikacji publicznej.

Repozytorium GitHub:

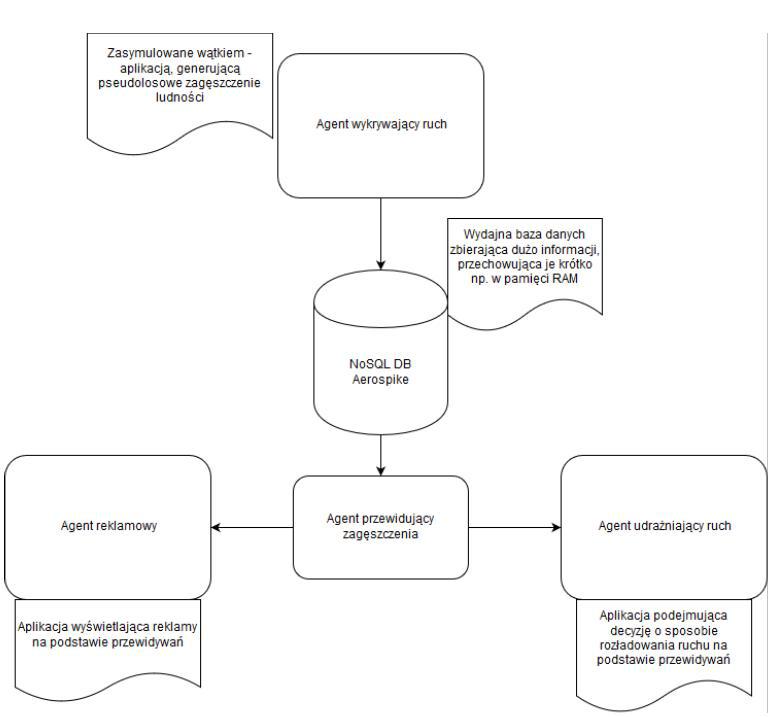
Link do repozytorium: <https://github.com/MichalBogucki/WSD-zespol-J>

**Rozdział I - część B**

* Proponowana architektura rozwiązania

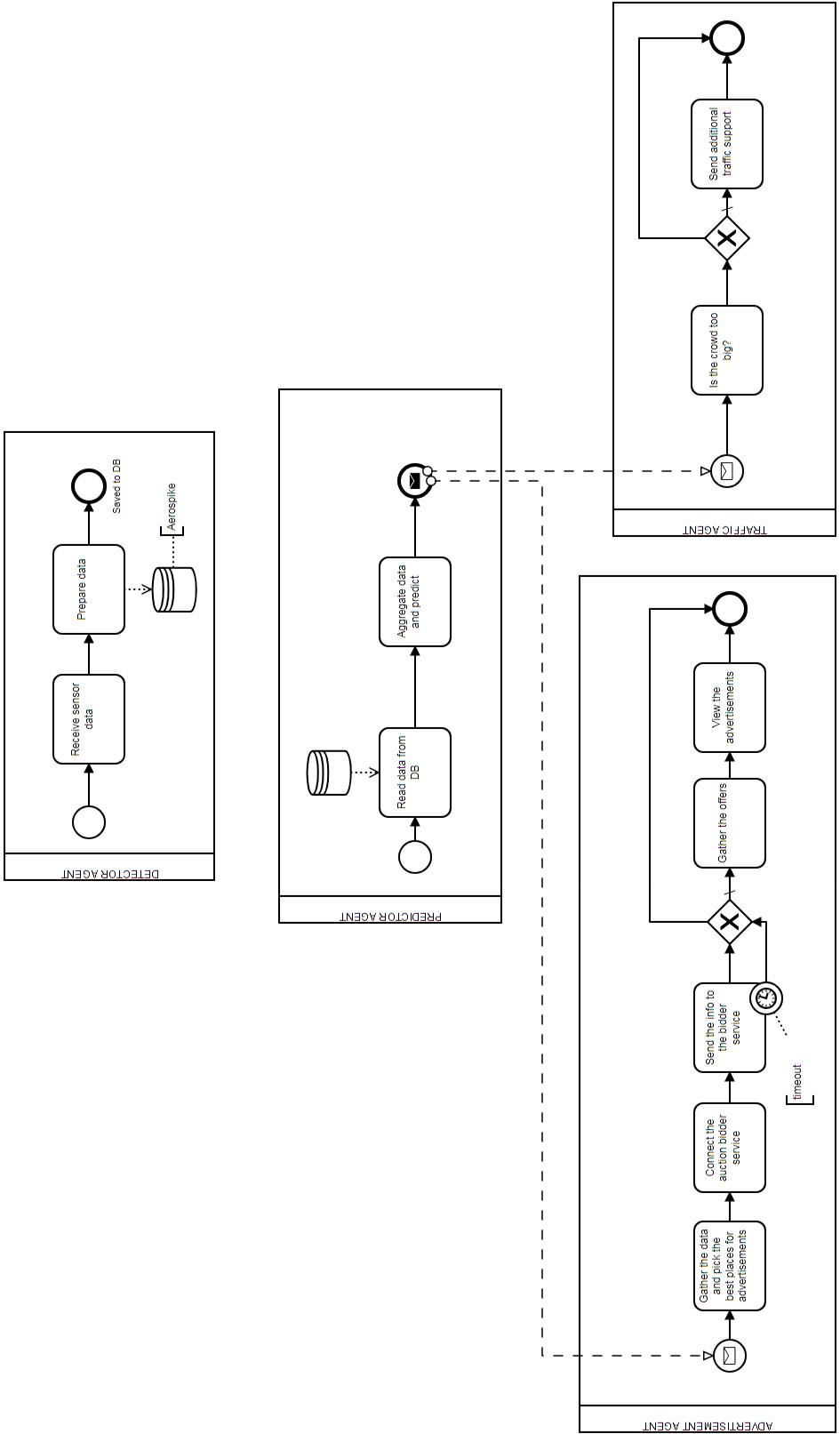
**Rozdział II – część B**

* Projekt systemu wieloagentowego, wykonany w metodologii BPMN (diagram kolaboracji):



**Rozdział II – część B**

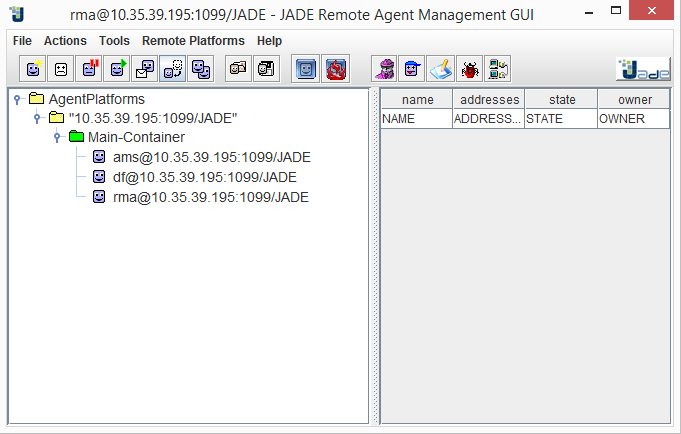
* Projekt systemu wieloagentowego, wykonany w metodologii BPMN (diagram kolaboracji):



**3. Opis implementacji**

Projekt zaimplementowano przy pomocy platformy agentowej JADE (Java Agent DEvelopment Framework). Jest to framework napisany w Javie i wspomaga budowę systemów wieloagentowych. Pozwala on na tworzenie i zarządzanie agentami (zgodnymi ze standardem FIPA - [Foundation for Intelligent Physical Agents](https://pl.wikipedia.org/wiki/Foundation_for_Intelligent_Physical_Agents)).  
  
**1. Elementy składowe JADE:**

* kontener (środowisko uruchomieniowe) - w jego obrębie funkcjonują agenci. Kontenery składają się na platformę. Na platformie wyróżnia się jeden kontener, tzw. kontener główny, który zawsze jest aktywny, i przy pomocy którego rejestrowane są inne kontenery.
* biblioteka klas wspomagająca tworzenie agentów.
* narzędzia graficzne umożliwiające administrację agentami.



Kontener główny (Main-Container) oprócz rejestracji innych kontenerów zawsze zawiera dwóch agentów o specjalnym statusie:

* AMS (Agent Management System) - zarządzanie regułami współżycia agentów na platformie, np. unikalnością nazw agentów.
* DF (Directory Facilitator) – przechowuje spis usług oferowanych przez agenty; agenty mogą rejestrować swoje usługi u agenta DF aby udostępnić je innym agentom.

Nazwy agentów tworzone są według następującego wzorca: <nazwa\_agenta>@<nazwa\_platformy>. Jest to globalny identyfikator agenta.

**2. Dodanie agenta**  
  
Utworzenie nowego agenta polega na rozszerzeniu klasy jade.core.Agent i zaimplementowaniu metod setup() oraz takeDown():

import jade.core.Agent;  
  
public class MyNewAgent extends Agent {  
  protected void setup() {  
    //…  
  }  
  protected void takeDown() { //opcjonalnie  
    //…  
  }  
}

Metoda setup() jest czymś na wzór konstruktora klasy Java. Wykonywana jest ona bezpośrednio po utworzeniu instancji agenta. Zawiera operacje inicjalizujące, takie jak np. dodanie zachowań (o czym mowa później).

Metoda takeDown() jest czymś na wzór destruktora klasy Java. Wykonywana jest ona bezpośrednio przed usunięciem instancji agenta z platformy.

#### 3. Zadania agentów - zachowania.

Właściwe zadania agenta określone są w ramach tzw. zachowań. Zachowaniem jest klasa dziedzicząca po klasie jade.core.behaviours.Behaviour. Wymagane jest przy tym nadpisanie metod *action()* i *done()* tejże klasy.

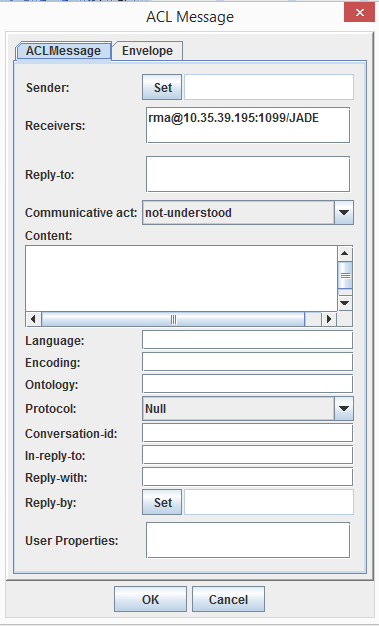
public class ExampleBehaviour extends Behaviour {  
  public void action() {  
    //…  
  }  
  public boolean done() {  
    //…  
  }  
}

Metoda action() określa zadanie agenta. Agent może posiadać wiele zachowań. Przypisanie zachowań odbywa się przy pomocy metody agenta addBehaviour(). Zachowania można dodawać na dwa sposoby: w metodzie *setup()* lub wewnątrz innych zachowań. Metoda *action()* nie może być przerwana przez inne zachowanie danego agenta. Zachowania są wykonywana w ramach pojedynczego wątku. Każdy agent ma zdefiniowaną kolejkę zachowań, które nie wywłaszczają się wzajemnie.   
  
Metoda done() zwraca wartość oznaczającą zakończenie zadanie lub nie (odpowiednio true/false). Może to być np. warunek liczby wywołań metody action():

  public boolean done() {  
    if(n>5) return true; else return false;  
  }

#### 3. Komunikacja

Ważną cechą platformy jest możliwość porozumiewania się agentów między sobą. Agenci mogą wymieniać komunikaty/wiadomości zgodne ze standardem **ACL** (Agent Communication Language) w sposób asynchroniczny. Każdy agent ma swoją kolejkę wiadomości, do której trafiają zaadresowane do niego wiadomości. Gdy przychodzi wiadomość, agent jest powiadamiany o tym fakcie. Standard ACL określa parametry opisujące wiadomość: **nadawca**, **odbiorca**, parametr określający tzw. intencję komunikacyjną (**performative**), **język**, w jakim wyrażona jest wiadomość, **zawartość** i szereg innych. Komunikat reprezentowany jest jako obiekt klasy jade.lang.acl.ACLMessage.

//tu coś można wpisać w te puste pola jak już będziemy mieć konkretnych agentów i podmienić tego screena

**3.1 Wysłanie wiadomości**

Do wysłania wiadomości służy metoda klasy agenta send():

  ACLMessage msg = new ACLMessage(ACLMessage.INFORM);  
  msg.addReceiver(new AID("<nazwa\_odbiorcy>", AID.ISLOCALNAME));  
  msg.setLanguage("Polish");  
  msg.setContent("Znow w Warszawie wieje wiatr");  
   send(msg);

Obiekt AID reprezentuje unikalny identyfikator agenta, jego nazwę, stała AID.ISLOCALNAME wskazuje, że nazwa jest nazwą lokalną, unikalną w obrębie platformy.   
W przykładzie powyżej wybrano performatywę ‘inform’, ustawiono język wiadomości na ‘polski’ i przypisano wiadomości treść ‘Znow w Warszawie wieje wiatr’.  
  
**3.2 Odebranie wiadomości**

Pobierania wiadomości z kolejki wiadomości agenta dokonuje się przy pomocy metody receive():

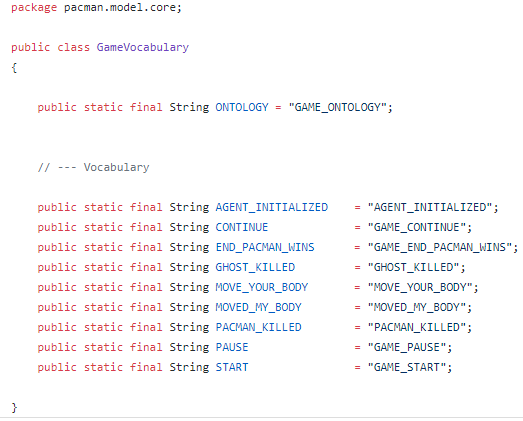
  ACLMessage msg = receive();

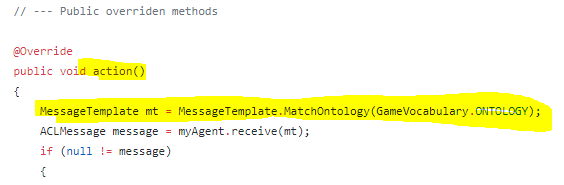
Często przeznaczeniem zachowania agenta jest oczekiwanie na komunikat, odbieranie go i przetwarzanie. W takim wypadku stosuje się blokowanie zachowania w celu zmniejszenia zużycia czasu procesora, który nie musi wykonywać ciągłej pętli w oczekiwaniu na komunikat.

public void action() {  
  ACLMessage msg = myAgent.receive();  
  if (msg != null) {  
    // przetwarzanie wiadomości  
  }  
  else {  
    block();  
  }  
}

W ten sposób zablokowane zachowanie przenoszone jest do kolejki zachowań zablokowanych i nie jest przetwarzane aż do czasu otrzymania wiadomości.  
 **4. Performatywy**

Ktoś coś?? To potrzebne w ogole?

**5. Ontologia  
**

**Potem przypisanie ontologii wygląda chyba tak. MessageTemplate to klasa z importu.  
**

**6. Agent DataGenerator**

**//Opiszemy do czego tej agent jest – jaki ma CEL, dlaczego ON  
//Jakie ma metody, jakie przyjmuje parametry  
//jakie ma zachowania  
//I wszystko inne co skurwiel agent ma według definicji z wikipedii  
//Opisać jakie ma pola – jakie atrybuty**

**7. Agent AdvertisementMerchant**

**// Opiszemy do czego tej agent jest – jaki ma CEL, dlaczego ON  
//Jakie ma metody, jakie przyjmuje parametry  
//jakie ma zachowania  
//I wszystko inne co skurwiel agent ma według definicji z wikipedii  
 //Opisać jakie ma pola – jakie atrybuty**

**8. Komunikacje między agentami  
//opisać negocjacje i dążenie do realizacji celów – sposoby negocjacji  
//użyte performatywy  
//jak wygląda wiadomość  
//Profokoły komunikacyjne  
//Języki treści  
//Opisana ontologia  
//Standard FIPA**

**9. Gui – wygląd apki**

**10. Opis stosowanych algorytmów**