

---

# AKADEMIA GÓRNICZO-HUTNICZA

Wydział Elektrotechniki, Automatyki, Elektroniki i Informatyki



KATEDRA INFORMATYKI

## WEB Dispatch Rider

*Interfejs webowy do systemu do planowania transportu*

Wersja **1.0** z dnia **19 czerwca 2012**

*Kierunek, rok studiów*

Informatyka, rok III

*Przedmiot*

Inżynieria Oprogramowania

*Prowadzący przedmiot*

dr inż. Małgorzata Żabińska-Rakoczy

dr inż. Jarosław Koźlak

*rok akademicki:* 2011/2012

*semestr:* letni

Zespół autorski:

Szymon Kasiński

Dariusz Mydlarz

# Spis treści

<b>1. Sformułowanie zadania projektowego</b>	4
1.1. Zapoznanie się z dotychczasowym systemem	4
1.1.1. Krótki opis założeń i budowy systemu Dispatch Rider	4
1.1.2. Ustalenia	4
1.1.3. Wnioski	5
<b>2. Porównanie bibliotek graficznych w języku JavaScript</b>	7
2.1. Informacje podstawowe	7
2.1.1. JavaScript InfoVis Toolkit	7
2.1.2. JSViz	8
2.1.3. Dracula	8
2.2. Funkcjonalność	9
2.2.1. JavaScript InfoVis Toolkit	9
2.2.2. JSViz	9
2.2.3. Dracula	9
2.3. Wydajność	10
2.3.1. JavaScript InfoVis Toolkit	10
2.3.2. JSViz	10
2.3.3. Dracula	10
2.4. Wnioski	11
<b>3. Analiza pliku wynikowego Dispatch Ridera</b>	12
3.1. Struktura pliku wynikowego Dispatch Ridera	12
3.2. Struktura pliku dla biblioteki graficznej	13
<b>4. Określenie wymagań oraz projekt graficzny systemu</b>	15
4.1. Wymagania funkcjonalne	15
4.2. Wymagania niefunkcjonalne	16
4.3. Przypadki użycia	16
4.3.1. Wyświetlenie zadań oczekujących na wykonanie	16
4.3.2. Wyświetlenie wyników zadań wykonanych	16
4.3.3. Wizualizacja grafu połączeń	17
4.3.4. Generowanie pliku określającego zlecenia	17
4.3.5. Generowanie pliku określającego parametry naczep	17
4.3.6. Generowanie pliku opisującego parametry holonów	18

4.3.7. Wyświetlanie informacji o każdym z holonów . . . . .	18
4.3.8. Dodanie danych do obliczeń . . . . .	19
4.4. Projekt GUI . . . . .	20
<b>5. Architektura systemu . . . . .</b>	<b>23</b>
5.1. Diagram komponentów systemu . . . . .	23
5.2. Aplikacja internetowa . . . . .	24
5.3. Scheduler . . . . .	24
5.4. Dispatch Rider . . . . .	24
5.5. Diagramy sekwencji . . . . .	25
5.5.1. Ładowanie danych . . . . .	25
5.5.2. Główna pętla programu . . . . .	26
<b>6. Oprogramowanie schedulera . . . . .</b>	<b>27</b>
<b>7. Instrukcja obsługi . . . . .</b>	<b>29</b>
7.1. Uruchomienie aplikacji . . . . .	29
7.2. Dodawanie zadania . . . . .	29
7.3. Zadania w trakcie . . . . .	29
7.4. Zadania obliczone . . . . .	30
7.4.1. Wyświetlanie grafu . . . . .	30
7.4.2. Zmiana sposobu wyświetlania grafu . . . . .	30
<b>8. Podsumowanie . . . . .</b>	<b>32</b>
8.1. Analiza spełnienia wymagań funkcjonalnych . . . . .	33
8.2. Napotkane błędy systemu DispatchRider . . . . .	33
<b>Spis rysunków . . . . .</b>	<b>35</b>
<b>Spis tabel . . . . .</b>	<b>36</b>
<b>Bibliografia . . . . .</b>	<b>37</b>

Niniejsze opracowanie powstało w trakcie i jako rezultat zajęć dydaktycznych z przedmiotu wymienionego na stronie tytułowej, prowadzonych w Akademii Górniczo-Hutniczej w Krakowie (AGH) przez osobę (osoby) wymienioną (wymienione) po słowach "Prowadzący zajęcia" i nie może być wykorzystywane w jakikolwiek sposób i do jakichkolwiek celów, w całości lub części, w szczególności publikowane w jakikolwiek sposób i w jakiegokolwiek formie, bez uzyskania uprzedniej, pisemnej zgody tej osoby (tych osób) lub odpowiednich władz AGH.

Copyright ©2012 Akademia Górniczo-Hutnicza (AGH) w Krakowie

# 1. Sformułowanie zadania projektowego

Przedmiotem naszego przedsięwzięcia będzie zbudowanie czytelnego i prostego w obsłudze webowego GUI do istniejącego już systemu *Dispatch Rider* zajmującego się problemem transportowym. Chcielibyśmy stworzyć narzędzie na tyle wygodne, by korzystanie z niego nie stanowiło problemu dla nowych jego użytkowników.

## 1.1. Zapoznanie się z dotychczasowym systemem

### 1.1.1. Krótki opis założeń i budowy systemu Dispatch Rider

Dispatch Rider to projekt mający za zadanie rozwiązywanie problemu transportowego, z wykorzystaniem holonów. System jest w stanie rozwiązywać problem PDPTW (Pickup and Delivery Problem with Time Windows) którego sednem jest znalezienie optymalnych tras przejazdu dla pojazdów należących do firmy transportowej. Każdy pojazd jest określony zespołem parametrów, lokalizacją początkową oraz końcową - czyli siedzibę firmy czy też parkingiem z którego wyruszają pojazdy. Problem ten należy do kategorii problemów statycznych, wszystkie dane znane są przed rozpoczęciem obliczeń, nie dochodzi do zmian danych wejściowych w trakcie działania programu.

Istotnym założeniem jest to że każdy punkt odbioru, dostarczenia przesyłki, tudzież baza transportowa, połączone są ze sobą w tym problemie bezpośrednio.

Implementacja projektu oparta jest na służącym do obliczeń agentowych frameworku JADE. System umożliwia ustawianie przez użytkownika wielu spośród parametrów dotyczących problemu i metod jego rozwiązania, generując po zakończonej pracy kompletne raporty opisujące trasy holonów w poszczególnych krokach czasowych.

### 1.1.2. Ustalenia

Po otrzymaniu kodu oraz plików wykonywalnych przeprowadziliśmy próby działania dotychczas istniejącego interfejsu webowego dla systemu transportowego. Wnioski z prób uruchomienia systemu zawarte są poniżej.

1. Otrzymany projekt nie zachowuje się deterministycznie pod różnymi platformami. Pod Linuxem (Ubuntu/Debian) uruchomione zadanie nie oblicza się, natomiast pod Windowsem (XP/7) GUI uruchamia się w sposób niedeterministyczny - nie możemy

zidentyfikować czynnika który powoduje że program działa lub nie. Nie jesteśmy w stanie określić co powoduje owe problemy, jednakże mogą one uniemożliwić rozszerzenie funkcjonalności GUI. Racjonalnie oceniając sytuację, niemożliwe jest pomyślnie rozwijanie systemu który nie działa. Faktem istotnym jest również to, że aplet zamiast uruchamiać się w przeglądarce, raportuje błąd i nie uruchamia się.

2. W czasie analizy szczególną uwagę przyłożyliśmy do opisu problemów z którymi spotkały się zespoły wcześniej realizujące projekty oparte na DR (np. praca Joanny Baran oraz Bartłomieja Pietrzyka). Z ich uwag wynika iż DispatchRider posiada błędy, które w szczególności mogą skutkować brakiem pliku wynikowego, nieuwzględnieniem sieci transportowej, nieuwzględnieniem czasu utworzenia holonu. Będziemy starać się zniwelować lub ominąć owe problemy, jednak istnieją uzasadnione obawy iż mogą one przeszkodzić w pomyślnej realizacji projektu
3. Dane wykorzystywane do wizualizacji wyników oraz tworzenia statystyk, są przekazywane za pośrednictwem pliku wyjściowego generowanego przez DR. Otwiera to możliwość zmiany koncepcji tworzenia GUI, ponieważ moduł nie jest w bezpośredni sposób powiązany z kodem DispatchRidera. Jednocześnie powoduje to konieczność przemyślanego zaprojektowania metody przekazywania owych danych do GUI.

### 1.1.3. Wnioski

1. W świetle wyżej zarysowanych problemów, chcielibyśmy zaproponować pewną zmianę koncepcji tworzenia graficznego interfejsu użytkownika.
2. Chcielibyśmy rozwiązać problemy z uruchamianiem GUI na różnych systemach. Mimo iż aplety Java powinny uruchamiać się bezproblemowo, empirycznie sprawdziliśmy że GUI nie działa prawidłowo na poszczególnych systemach operacyjnych. Aby zapobiec tym problemom, chcielibyśmy stworzyć nową aplikację webową i wykorzystać przy tym takie technologie jak: HTML5, CSS, JavaScript. Z naszego doświadczenia wynika iż dobrze zaprojektowane z użyciem w.w. GUI działać będzie na każdym systemie i każdej przeglądarce. Powodem dla którego preferujemy akurat te technologie, a nie aplety Javy, jest ich nowoczesność - pragniemy wykorzystać HTML5, zapewniający olbrzymie możliwości, choćby w dziedzinie modelowania i prezentacji grafów. Z przykrością musimy stwierdzić, iż dotychczasowe sposoby wizualizacji grafów w interfejsie DispatchRidera charakteryzowały się pewnym brakiem czytelności, oraz interaktywności.
3. Budowana przez nas aplikacja podobnie jak dotychczasowe projekty GUI, korzystałaby z plików wyjściowych DispatchRidera. Oddzielenie interfejsu użytkownika od głównego ciała programu pozwala na komfortową pracę, bez konieczności zagłębiania się w kod

głównego programu i jego modyfikacji, co mogłoby nawet uniemożliwić jakiekolwiek widoczne postępy w rozwijaniu przez nas dotychczasowego GUI.

4. Projektując GUI z użyciem naszych technologii, chcielibyśmy odtworzyć zaimplementowane w wersji z apletami Javy funkcjonalności, mając jednak stuprocentową pewność że w przeciwieństwie do niej, nasz interfejs będzie w pełni użytkowalny, i będzie możliwe jego uruchomienie na każdym systemie i przeglądarce obsługującej HTML5.
5. Chcielibyśmy skupić się zwłaszcza na problemie prezentacji tras, zwracając dużą uwagę na ergonomię działania programu. Użycie wyżej wymienionych technologii umożliwia tworzenie interaktywnych prezentacji grafu na które kładziemy dużą uwagę, oraz umożliwia właściwie nieograniczone możliwości modyfikowania ich, zmiany perspektywy oraz ilości przekazywanych informacji. W chwili obecnej, interfejs graficzny jest wyjątkowo nieergonomiczny, co wynika choćby z faktu iż jego uruchomienie jest czynnością wyjątkowo skomplikowana i nie zawsze kończąca się z powodzeniem. Mamy nadzieję że nasz pomysł warstwy prezentacji spowoduje jakościową zmianę w sposobie obsługi systemu.
6. Naszym zdaniem, kontynuacja istniejących implementacji graficznego interfejsu może przerodzić się w klasyczny marszu ku kłęsce. Aby temu zapobiec, postulujemy zmianę koncepcji tworzenia GUI i zbudowanie podwalin dobrze zaimplementowanego, udokumentowanego, łatwego w dalszym rozwoju, a przede wszystkim działającego GUI.

## 2. Porównanie bibliotek graficznych w języku JavaScript

Poniżej prezentujemy porównanie wybranych z kilkunastu różnych narzędzi JavaScriptowych, tych najciekawszych którymi można przedstawiać grafy.

### 2.1. Informacje podstawowe

#### 2.1.1. JavaScript InfoVis Toolkit

<b>Adres</b>	<a href="http://thejit.org/">http://thejit.org/</a>
<b>Licencja</b>	new BSD license - możliwe kopiowanie, modyfikacja, rozpowszechnianie, sprzedaż z zachowaniem informacji o autorze; więcej: <a href="https://github.com/philogb/jit/blob/master/LICENSE">https://github.com/philogb/jit/blob/master/LICENSE</a>
<b>Rozszerzalność</b>	Pod względem prawnym – rozszerzalność możliwa. Pod względem technicznym – kilka tysięcy linii kodu, jednakże brak sformalizowanej dokumentacji – polega ona na komentarzach w kodzie
<b>Inne uwagi</b>	Bardzo ładna, używana m.in. na stronach Mozilli oraz prezydenta USA. Dokumentacja korzystania z biblioteki: <a href="http://thejit.org/static/v20/Docs/index/General.html">http://thejit.org/static/v20/Docs/index/General.html</a> . Źródła: <a href="https://github.com/philogb/jit">https://github.com/philogb/jit</a>
<b>Ocena</b>	Jedynym problemem może być tylko maksymalna liczba generowanych węzłów. Poza tym posiada raczej wszystko, czego nam potrzeba, a w dodatku jest bardzo ładna.

Tabela 2.1: JavaScript InfoVis Toolkit - Informacje Podstawowe

### 2.1.2. JSViz

<b>Adres</b>	<a href="http://code.google.com/p/jsviz/">http://code.google.com/p/jsviz/</a>
<b>Licencja</b>	Apache 2.0 - dopuszcza użycie kodu źródłowego zarówno na potrzeby wolnego oprogramowania, jak i zamkniętego oprogramowania komercyjnego.
<b>Rozszerzalność</b>	Licencja zezwala, aczkolwiek brak dokumentacji, jedynie komentarze w kodzie o długości kilku/kilkunastu tysięcy linii.
<b>Inne uwagi</b>	Niedostatecznie przyjemna dla oka.
<b>Ocena</b>	Brak jakichkolwiek etykiet, prawie żadna możliwość wpływu na wygląd generowanego grafu, dość długie i chaotyczne generowanie się grafu, biblioteka z przed 5 lat, nie zachęca wyglądem.

Tabela 2.2: JSViz - Informacje Podstawowe

### 2.1.3. Dracula

<b>Adres</b>	<a href="http://www.graphdracula.net/">http://www.graphdracula.net/</a>
<b>Licencja</b>	MIT (X11) license - możliwe są kopiowanie, modyfikacja, rozpowszechnianie, w tym sprzedaż.
<b>Rozszerzalność</b>	Pod względem prawnym – rozszerzalność możliwa. Pod względem technicznym – ok. 10 tysięcy linii kodu, jednakże brak sformalizowanej dokumentacji – polega ona na komentarzach w kodzie i udostępnieniu pojedynczego przykładu na stronie projektu oraz skomentowanych przykładów możliwych do ściągnięcia.
<b>Inne uwagi</b>	Niewystarczająco wydajna, brak wbudowanych mechanizmów „ładnego” rozkładania grafów w przestrzeni.

Tabela 2.3: Dracula - Informacje Podstawowe



## 2.2. Funkcjonalność

### 2.2.1. JavaScript InfoVis Toolkit

<b>Interaktywne węzły</b>	Tak
<b>Interaktywne ścieżki</b>	Nie
<b>Zoom</b>	Tak
<b>Opis węzłów</b>	Tak
<b>Opis ścieżek</b>	Tak
<b>Automatyczna zmiana kształtu</b>	Nie
<b>Inne uwagi</b>	Możliwość przesuwania grafu. Wczytuje dane w formacie JSON

Tabela 2.4: JavaScript InfoVis Toolkit - Funkcjonalność

### 2.2.2. JSViz

<b>Interaktywne węzły</b>	Tak
<b>Interaktywne ścieżki</b>	Nie
<b>Zoom</b>	Nie
<b>Opis węzłów</b>	Nie
<b>Opis ścieżek</b>	Nie
<b>Automatyczna zmiana kształtu</b>	Nie
<b>Inne uwagi</b>	Wczytuje dane w formacie XML

Tabela 2.5: JSViz - Funkcjonalność

### 2.2.3. Dracula

<b>Interaktywne węzły</b>	Tak
<b>Interaktywne ścieżki</b>	Nie
<b>Zoom</b>	Nie
<b>Opis węzłów</b>	Tak
<b>Opis ścieżek</b>	Tak
<b>Automatyczna zmiana kształtu</b>	Nie
<b>Inne uwagi</b>	W bibliotece zaimplementowane takie algorytmy jak: Bellman-Ford, Dijkstra, Floyd-Warshall, quicksort, selectionsort, mergesort, topologicalsort. W kodzie wiele funkcji czekających na implementację.

Tabela 2.6: Dracula - Funkcjonalność

## 2.3. Wydajność

Dla wyżej wymienionych bibliotek przeprowadziliśmy testy wydajnościowe. Ich wyniki są następujące:

### 2.3.1. JavaScript InfoVis Toolkit

Węzłów	Brak krawędzi	Krawędzie
100	Generuję się od razu, można bez problemu korzystać.	0-3 dla każdego węzła Generuję się kilka sekund, korzysta się wygodnie.
200	Powoduje kilku/kilkunastosekundowe generowanie się grafu.	Podobnie, dodatkowo utrudnia wygodne korzystanie z grafu.
500	Powoduje kilkunastosekundowe generowanie się, korzystanie z grafu raczej niemożliwe.	Generowanie trwa kilka minut (zdecydowanie za długo).
1000	Generowanie trwa kilka minut.	

Tabela 2.7: JavaScript InfoVis Toolkit – Wydajność

### 2.3.2. JSViz

Węzłów	Krawędzie/brak krawędzi (0-3 dla każdego węzła)
100	Generowanie trwa kilkadziesiąt sekund.
200	Generowanie trwa kilkadziesiąt sekund.
500	Generuje się kilka minut, nie można wygodnie korzystać.

Tabela 2.8: JSViz – Wydajność

### 2.3.3. Dracula

Węzłów	Krawędzie/brak krawędzi (0-3 dla każdego węzła)
500	Powoduje zauważalny, ok 3 sek. narzut czasowy na tworzenie grafu.
1000	Zawiesza przeglądarkę.

Tabela 2.9: Dracula – Wydajność

## 2.4. Wnioski

Zgodnie z powyższym porównaniem różnych bibliotek podjęliśmy decyzję, iż w naszym projekcie skorzystamy z możliwości jakie daje nam biblioteka JavaScriptInfoVis Toolkit. Jest ona najmłodszym z pośród recenzowanych projektów, ale jednocześnie jest bardzo rozbudowana i dopracowana. Pozwala na rysowanie wielu rodzajów grafów i robi to w bardzo ładny dla oka sposób. Dobrą rekomendacją dla niej jest także fakt, iż wykorzystywana jest przez Fundację Mozilla na swoich stronach, a także można ją znaleźć na oficjalnej stronie prezydenta Stanów Zjednoczonych. Kod biblioteki został napisany w czytelny sposób, korzystanie z niej nie powinno stanowić większych problemów. Licencja, na której jest udostępniana pozwala na wykorzystanie jej w naszym projekcie.

## 3. Analiza pliku wynikowego Dispatch Ridera

Program Dispatch Rider generuje przy odpowiedniej konfiguracji (recording="true") plik wyjściowy w formacie XML, który przedstawia model przetwarzanego problemu transportowego. W założeniu nasza aplikacja webowa ma otrzymywać taki plik wynikowy, otwierać go, a następnie parsować w celu odczytania istotnych dla wizualizacji danych. Dane te będą następnie przetwarzane do formatu akceptowanego przez wybraną przez nas bibliotekę graficzną (JavaScript InfoVis Toolkit).

### 3.1. Struktura pliku wynikowego Dispatch Ridera

Plik generowany przez Dispatch Ridera ma następującą strukturę:

```
<simulation_measures>
  <measures comId="42" timestamp="0">
    <holon id="0">
      <measure name="NumberOfCommissions">1.0</measure>
      <measure name="AverageDistanceFromCurLocationToBaseForAllCommissions">
        83.2608973717183</measure>
      <measure name="AverageDistPerCommissionBeforeChange">
        34.85191972054854</measure>
    </holon>
  </measures>
  <measures comId="51" timestamp="0">
    <holon id="0">
      <measure name="NumberOfCommissions">1.0</measure>
      <measure name="AverageDistanceFromCurLocationToBaseForAllCommissions">
        82.30488721735766</measure>
      <measure name="AverageDistPerCommissionBeforeChange">
        34.85191972054854</measure>
    </holon>
    <holon id="1">
```

```
<measure name="NumberOfCommissions">1.0</measure>
<measure name="AverageDistanceFromCurLocationToBaseForAllCommissions">
  66.91504977153873</measure>
<measure name="AverageDistPerCommissionBeforeChange">
  21.8026373564547</measure>
</holon>
</measures>
</simulation_measures>
```

Dzięki temu, że jest to plik zgodny ze specyfikacją XML, do parsowania użyjemy JavaScriptu. Uściślając skorzystamy z inwentarza biblioteki jQuery i być może pluginu do przekształcania formatu XML w format JSON (przykładowy sposób zwykłego parsingu przy jej użyciu: <http://think2loud.com/224-reading-xml-with-jquery/>, <http://www.switchonthecode.com/tutorials/xml-parsing-with-jquery> oraz pluginu do konwersji pomiędzy XML i JSON: <http://www.fyneworks.com/jquery/xml-to-json/>).

### 3.2. Struktura pliku dla biblioteki graficznej

Powyższy plik XML będziemy przekształcać do formatu JSON i postaci podobnej do poniższej:

```
var json = [
  {
    "adjacencies": [
      "graphnode21",
      {
        "nodeTo": "graphnode1",
        "nodeFrom": "graphnode0",
        "data": {
          "$color": "#557EAA"
        }
      }
    ], {
      "nodeTo": "graphnode13",
      "nodeFrom": "graphnode0",
      "data": {
        "$color": "#909291"
      }
    }
  ]
],
```

```
    "data": {
      "$color": "#83548B",
      "$type": "circle",
      "$dim": 10
    },
    "id": "graphnode0",
    "name": "graphnode0"
  }, {
    "adjacencies": [
      {
        "nodeTo": "graphnode2",
        "nodeFrom": "graphnode1",
        "data": {
          "$color": "#557EAA"
        }
      }, {
        "nodeTo": "graphnode4",
        "nodeFrom": "graphnode1",
        "data": {
          "$color": "#909291"
        }
      }
    ],
    "data": {
      "$color": "#EBB056",
      "$type": "circle",
      "$dim": 11
    },
    "id": "graphnode1",
    "name": "graphnode1"
  }
];
```

## 4. Określenie wymagań oraz projekt graficzny systemu

### 4.1. Wymagania funkcjonalne

System ma współpracować z już istniejącym oprogramowaniem Dispatch Rider. W tym celu podjęliśmy się zdefiniowania następujących wymagań funkcjonalnych takich jak:

1. Generowanie plików konfiguracyjnych dla Dispatch Ridera:
  - generowanie pliku określającego zlecenia, o składni: **ilość-pojazdów ładowność prędkość**
  - generowanie pliku określającego czas nadchodzenia zleceń, o składni: nr-lini **czas-zlecenia**
  - generowanie pliku zawierającego liczbę kierowców
  - generowanie pliku określającego parametry ciągników, o składni: **moc niezawodność wygoda zużycie-paliwa typ-zaczepu**
  - generowanie pliku opisującego parametry naczep o składni: **masa pojemność typ-ładunku uniwersalność typ-zaczepu**
  - generowanie pliku opisującego parametry holonów o składni: initialCapacity = **pojemność-pojazdu** mode = **tryb** bases = **ilość-baz** eUnitsCount = **ilość-jednostek**
  - generowanie pliku konfiguracyjnego configuration.xml
2. Kolejowanie zadań zlecanych systemowi.
3. Uruchamianie systemu Dispatch Rider z użyciem dostarczonych z zewnątrz lub wygenerowanych plików konfiguracyjnych
4. Odczyt i parsowanie plików wynikowych Dispatch Ridera, wraz z wizualizacją otrzymanych wyników
  - wizualizacja grafu sieci transportowej
  - wizualizacja tras pokonanych przez pojazdy
  - wyświetlanie informacji o każdym z holonów
  - wyświetlanie zbiorczego podsumowania obliczeń zawierającego koszt, dystans, czas, parametry holonów
5. Przechowywanie wyników obliczeń, w celu zaprezentowania na żądanie

## 4.2. Wymagania niefunkcjonalne

1. Dokumentacja techniczna w postaci strony wiki, dokumentu tekstowego oraz komentarzy w kodzie programu.
2. Przenośność - zapewnienie w pełni poprawnego działania programu na najpopularniejszych przeglądarkach oraz systemach operacyjnych.
3. Płynność działania wizualizacji grafów w oknie przeglądarki.
4. Łatwość i intuicyjność obsługi, umożliwienie komfortowego korzystania z programu bez poznawania szczegółowych instrukcji dotyczących uruchamiania i konfiguracji.

## 4.3. Przypadki użycia

Jedynym aktorem w naszym systemie jest Użytkownik - osoba posługująca się systemem za pośrednictwem strony WWW - jest ona w stanie dodawać nowe zadania oraz oglądać wyniki już obliczone.

### 4.3.1. Wyświetlenie zadań oczekujących na wykonanie

**Cel przypadku użycia:** Sprawdzenie przez użytkownika jakie zadania zostały już wybrane do wykonania, lecz ich wykonanie się nie rozpoczęło.

**Aktor:** Użytkownik

**Warunki początkowe:** Istnieje aplikacja serwera, klient jest do niej podłączony przy pomocy aplikacji klienckiej.

**Scenariusz:**

1. Użytkownik przełącza się na zakładkę zadań oczekujących
2. Aplikacja pobiera listę plików znajdujących się w katalogu uploads
3. Aplikacja wyświetla listę plików w postaci tabelki

### 4.3.2. Wyświetlenie wyników zadań wykonanych

**Cel przypadku użycia:** Sprawdzenie przez użytkownika jakie zadania zostały już wybrane do wykonania, lecz ich wykonanie się nie rozpoczęło.

**Aktor:** Użytkownik

**Warunki początkowe:** Istnieje aplikacja serwera, klient jest do niej podłączony przy pomocy aplikacji klienckiej.

**Scenariusz:**

1. Użytkownik przełącza się na zakładkę zadań oczekujących



2. Aplikacja pobiera listę plików znajdujących się w katalogu uploads
3. Aplikacja wyświetla listę plików w postaci tabelki

#### 4.3.3. Wizualizacja grafu połączeń

**Cel przypadku użycia:** Przedstawienie użytkownikowi grafu obrazującego sposób poruszania się holonów.

**Aktor:** Użytkownik

**Warunki początkowe:** Istnieje aplikacja serwera, klient jest do niej podłączony przy pomocy aplikacji klienckiej. Dany zestaw został obliczony

**Scenariusz:**

1. Użytkownik przełącza się na zakładkę zadań obliczonych
2. Aplikacja pobiera listę plików znajdujących się w katalogu ready i wyświetla je
3. Użytkownik klika na wybrany model obliczeń
4. Aplikacja wyświetla graf ukazujący trasy holonów

#### 4.3.4. Generowanie pliku określającego zlecenia

**Cel przypadku użycia:** Ręczne wprowadzenie przez użytkownika parametrów określających zlecenie.

**Aktor:** Użytkownik

**Warunki początkowe:** Istnieje aplikacja serwera, klient jest do niej podłączony przy pomocy aplikacji klienckiej.

**Scenariusz:**

1. Użytkownik przełącza się na zakładkę wprowadzania danych
2. Użytkownik wybiera ręczne wprowadzanie danych zlecenia
3. Użytkownik wypełnia pola określające poszczególne parametry i klika na zapisz
4. Aplikacja zapisuje do katalogu plik określający zadanie

#### 4.3.5. Generowanie pliku określającego parametry naczep

**Cel przypadku użycia:** Ręczne wprowadzenie przez użytkownika parametrów określających poszczególne naczepy.

**Aktor:** Użytkownik

**Warunki początkowe:** Istnieje aplikacja serwera, klient jest do niej podłączony przy pomocy aplikacji klienckiej.

**Scenariusz:**

1. Użytkownik przełącza się na zakładkę wprowadzania danych
2. Użytkownik wybiera ręczne wprowadzanie danych naczepy
3. Użytkownik wypełnia pola określające poszczególne parametry naczepy i klika na zapisz
4. Aplikacja zapisuje do katalogu plik określający zadanie

#### 4.3.6. Generowanie pliku opisującego parametry holonów

**Cel przypadku użycia:** Ręczne wprowadzenie przez użytkownika parametrów określających holony.

**Aktor:** Użytkownik

**Warunki początkowe:** Istnieje aplikacja serwera, klient jest do niej podłączony przy pomocy aplikacji klienckiej.

**Scenariusz:**

1. Użytkownik przełącza się na zakładkę wprowadzania danych
2. Użytkownik wybiera ręczne wprowadzanie parametrów holonów
3. Użytkownik wypełnia pola initialCapacity, mode, bases, eUnitsCount
4. Aplikacja zapisuje do katalogu plik definiujący parametry holonów

#### 4.3.7. Wyświetlanie informacji o każdym z holonów

**Cel przypadku użycia:** Sprawdzenie przez użytkownika danych pojedynczego holonu.

**Aktor:** Użytkownik

**Warunki początkowe:** Istnieje aplikacja serwera, klient jest do niej podłączony przy pomocy aplikacji klienckiej. Na serwerze zostały już przeprowadzone jakieś obliczenia

**Scenariusz:**

1. Użytkownik przełącza się na zakładkę danych obliczonych.
2. Użytkownik wybiera obliczenie
3. Użytkownik wybiera interesujący go holon

4. Aplikacja wyświetla dane zaznaczonego holonu

#### 4.3.8. Dodanie danych do obliczeń

**Cel przypadku użycia:** Dodanie zadania do obliczeń.

**Aktor:** Użytkownik

**Warunki początkowe:** Istnieje aplikacja serwera, klient jest do niej podłączony przy pomocy aplikacji klienckiej.

**Scenariusz:**

1. Użytkownik przełącza się na zakładkę wprowadzania danych.
2. Użytkownik wybiera plik lub wprowadza dane w pola tabeli
3. Aplikacja umieszcza dane w folderze uploads

## 4.4. Projekt GUI

Projektując GUI webowe staraliśmy się zmaksymalizować czytelność i łatwość korzystania z naszej aplikacji. W związku z tym chcielibyśmy zaimplementować wygląd aplikacji w taki sposób, by informacje w danym momencie nieistotne nie przysłaniały użytkownikowi tych, na których chciałby się skupić. Poniżej prezentujemy pierwsze szkice, które stworzyliśmy.

Dispatch Rider    Załóż konto

Dispatch Rider Web to aplikacja internetowa pozwalająca na zmniejszanie kosztów problemu transportowego. Prosty w obsłudze i czytelny interfejs ma zapewnić użytkownikowi maksymalną satysfakcję z używania tego produktu. Dodażyliśmy wszelkich starań, aby aplikacja ta spełniała oczekiwania odbiorców. Zapraszamy do korzystania.

e-mail

hasło

[nie pamiętam hasła](#)


Zaloguj się

---

Aplikacja ta została wykonana przez studentów III roku Informatyki (WEAIE) AGH - Dariusza Mydlarza i Szymona Kasińskiego w ramach zajęć z przedmiotu Inżynieria Programowania pod opieką dr inż. Małgorzaty Zabińskiej-Rakoczy oraz dr inż. Jarosława Koźłaka w roku akademickim 2011/2012. Wszelkie prawa zastrzeżone.

dmydlarz@student.agh.edu.pl    kasiński@student.agh.edu.pl

Rysunek 4.1: Strony powitalna

Dispatch Rider	 Robert Lewandowski	Lista twoich plików			
nazwa pliku	data uploadu		data wygenerowania	akcje	
example.xml	2012-03-13 13:58		2012-04-13-14:34	Kopiuj	Edytuj Usuń
example.xml	2012-03-13 13:58		2012-04-13-14:34	Kopiuj	Edytuj Usuń
example.xml	2012-03-13 13:58		2012-04-13-14:34	Kopiuj	Edytuj Usuń
example.xml	2012-03-13 13:58		2012-04-13-14:34	Kopiuj	Edytuj Usuń
example.xml	2012-03-13 13:58		2012-04-13-14:34	Kopiuj	Edytuj Usuń
example.xml	2012-03-13 13:58		2012-04-13-14:34	Kopiuj	Edytuj Usuń
example.xml	2012-03-13 13:58		2012-04-13-14:34	Kopiuj	Edytuj Usuń
example.xml	2012-03-13 13:58		2012-04-13-14:34	Kopiuj	Edytuj Usuń

Aplikacja ta została wykonana przez studentów III roku Informatyki (WEAIE) AGH - Dariusza Mydlarz i Szymona Kasiniego w ramach zajęć z przedmiotu Inżynieria Oprogramowania pod opieką dr inż. Małgorzaty Zabińskiej-Rakoczy oraz dr. inż. Jarosława Koźłaka w roku akademickim 2011/2012. Wszelkie prawa zastrzeżone.

dmydlarz@student.agh.edu.pl kasinski@student.agh.edu.pl

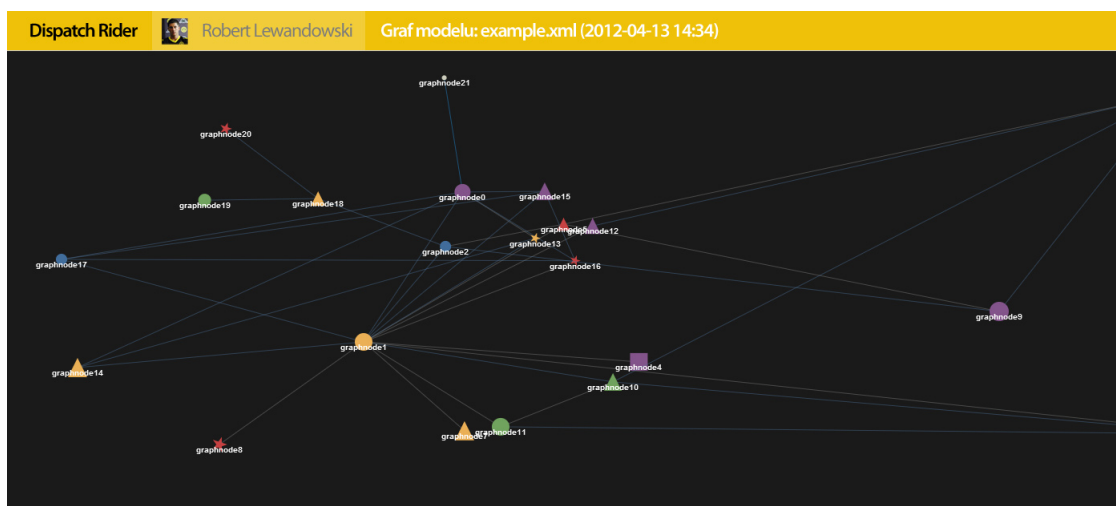
Rysunek 4.2: Lista zadań

Dispatch Rider	 Robert Lewandowski	Lista obliczonych dla ciebie modeli			
nazwa pliku	Modele obliczone 12		data wygenerowania	akcje	
example.xml	Stan modeli obliczanych 3		2012-04-13-14:34	Kopiuj	Edytuj Usuń
example.xml	Dodaj nowy model 3		2012-04-13-14:34	Kopiuj	Edytuj Usuń
example.xml	Wyloguj 3		2012-04-13-14:34	Kopiuj	Edytuj Usuń
example.xml	2012-03-13 13:58		2012-04-13-14:34	Kopiuj	Edytuj Usuń
example.xml	2012-03-13 13:58		2012-04-13-14:34	Kopiuj	Edytuj Usuń
example.xml	2012-03-13 13:58		2012-04-13-14:34	Kopiuj	Edytuj Usuń
example.xml	2012-03-13 13:58		2012-04-13-14:34	Kopiuj	Edytuj Usuń

Aplikacja ta została wykonana przez studentów III roku Informatyki (WEAIE) AGH - Dariusza Mydlarz i Szymona Kasiniego w ramach zajęć z przedmiotu Inżynieria Oprogramowania pod opieką dr inż. Małgorzaty Zabińskiej-Rakoczy oraz dr. inż. Jarosława Koźłaka w roku akademickim 2011/2012. Wszelkie prawa zastrzeżone.

dmydlarz@student.agh.edu.pl kasinski@student.agh.edu.pl

Rysunek 4.3: Menu aplikacji



Aplikacja ta została wykonana przez studentów III roku Informatyki (WEAIE) AGH - Dariusza Mydlarz i Szymona Kasinskiego w ramach zajęć z przedmiotu Inżynieria Oprogramowania pod opieką dr inż. Małgorzaty Zabińskiej-Rakoczy oraz dr. inż. Jarosława Kozłaka w roku akademickim 2011/2012. Wszelkie prawa zastrzeżone.

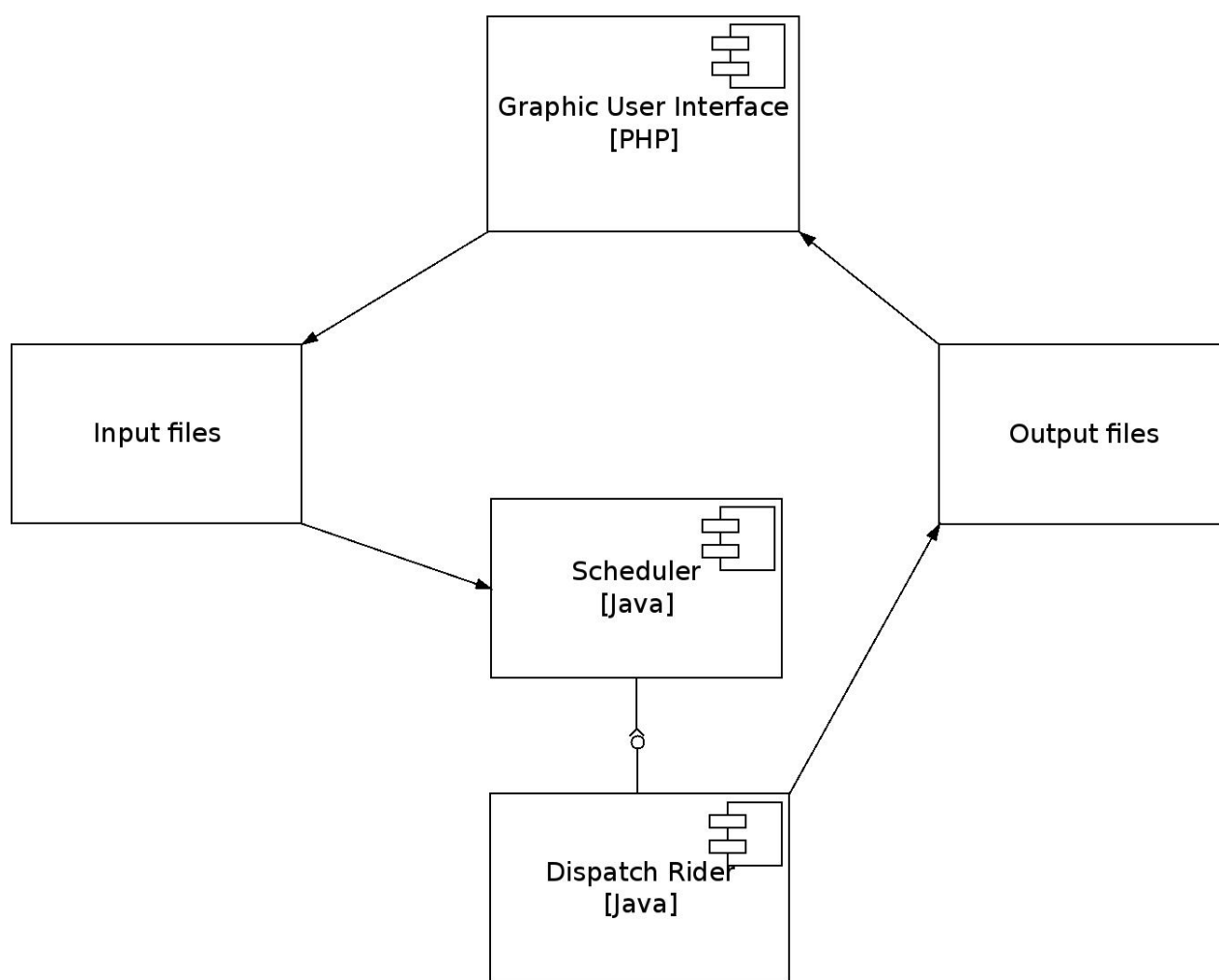
dmydlarz@student.agh.edu.pl kasinski@student.agh.edu.pl

Rysunek 4.4: Widok grafu

## 5. Architektura systemu

Architektura budowanego systemu nie będzie skomplikowana – jej prostota zapewniająca przenośność oraz łatwość uruchomienia jest jednym z naszych założeń. Całość, czyli aplikacja internetowa, scheduler oraz Dispatch Rider, uruchamiana będzie na jednej maszynie – serwerze.

### 5.1. Diagram komponentów systemu



Rysunek 5.1: Diagram komponentów systemu

## 5.2. Aplikacja internetowa

Aplikacja internetowa to implementacja interfejsu, za pomocą której użytkownik będzie mógł łączyć się z systemem i pracować na nim. Zostanie napisana między innymi przy użyciu języka PHP, co oznacza, że na serwerze, na którym zostanie uruchomiona aplikacja konieczna będzie jego obsługa.

Najważniejsze z punktu widzenia całego systemu będą znajdujące się w niej 2 katalogi: **uploads** oraz **ready**. W pierwszym będą pojawiać się pliki z właściwościami i konfiguracją wysyłane przez użytkownika. Każdy zestaw plików w osobnym folderze o nazwie utworzonej z nazwy wybranej przy uploadzie przez użytkownika. Pliki te będzie można edytować z poziomu aplikacji **www** do momentu, aż nie zostaną przejęte przez schedulera do dalszego przetwarzania. Po zakończeniu przetwarzania problemu przez Dispatch Ridera, pliki wynikowe zostaną zdeponowane w folderze **ready**. Aplikacja internetowa będzie przeglądać każdy z tych katalogów w celu wyświetlenia użytkownikowi informacji na temat przebiegu przetwarzania.

Poza tym w aplikacji internetowej będzie można podejrzeć graf problemu zbudowany w oparciu o pliki wynikowe z Dispatch Ridera, co jest głównym celem tego projektu. Graf będzie utworzony przy pomocy biblioteki **JavaScript InfoVis Toolkit**.

## 5.3. Scheduler

Scheduler to osobna aplikacja napisana w języku Java służąca do kolejkovania zadań przesłanych do systemu przez użytkownika. Skanuje ona zawartość folderów **uploads** i **ready** z aplikacji internetowej i zarządza ich przesyłaniem do Dispatch Ridera. Scheduler jest więc łącznikiem pomiędzy usługą kliencką, a serwerową. W momencie ukończenia przetwarzania problemu przez Dispatch Ridera scheduler przenosi pliki wynikowe do odpowiedniego podfolderu katalogu **ready** i w sytuacji gdy jakiś problem czeka już w kolejce do obliczeń, przesyła go do Dispatch Ridera.

## 5.4. Dispatch Rider

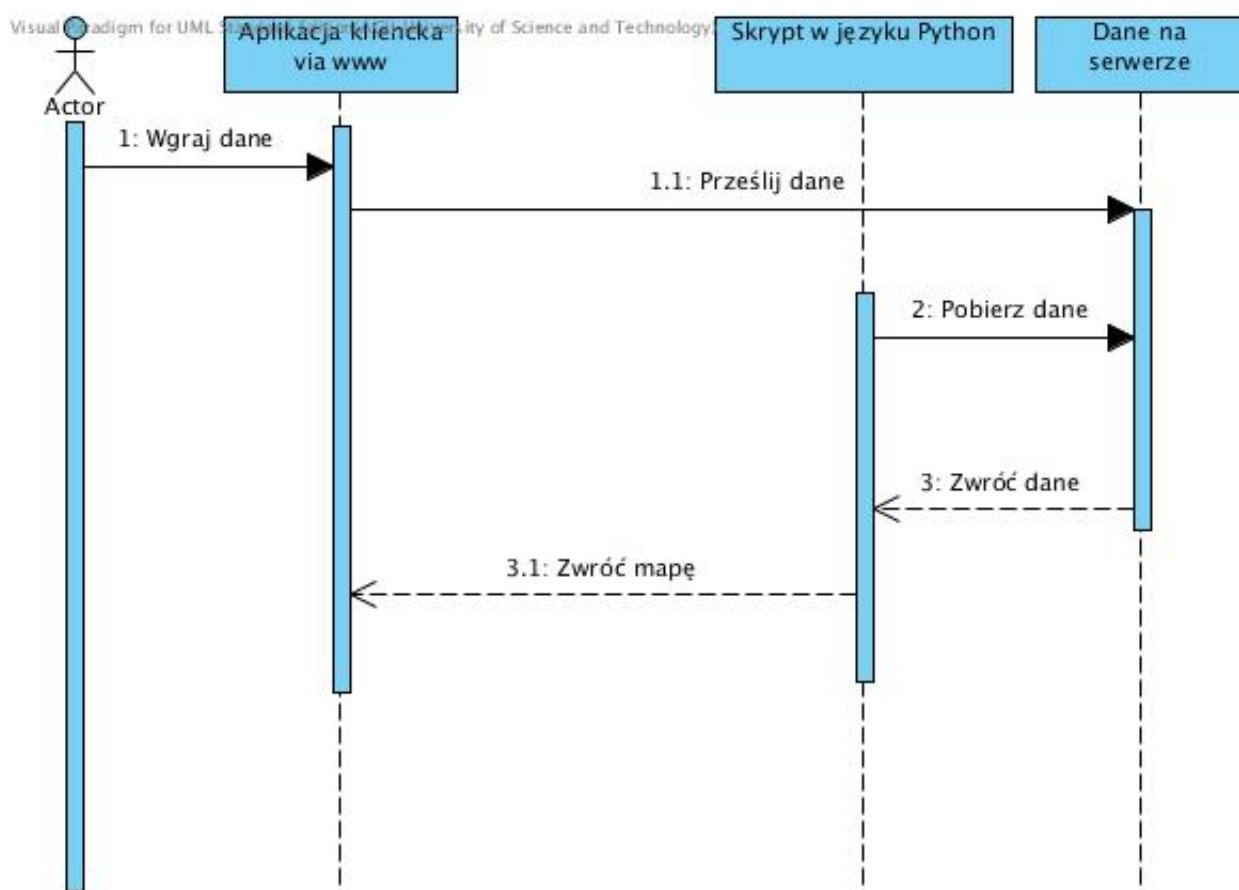
Serce całego systemu. Aplikacja, która zajmuje się wykonywaniem wszystkich obliczeń związanych z danym problemem. Wykorzystana będzie gotowa już aplikacja, napisana przez naszych starszych kolegów, nie będziemy ingerować w jej kod źródłowy, a jedynie skorzystamy z jej plików wynikowych. Aplikacja jest uruchamiana i zarządzana przez schedulera.



## 5.5. Diagramy sekwencji

### 5.5.1. Ładowanie danych

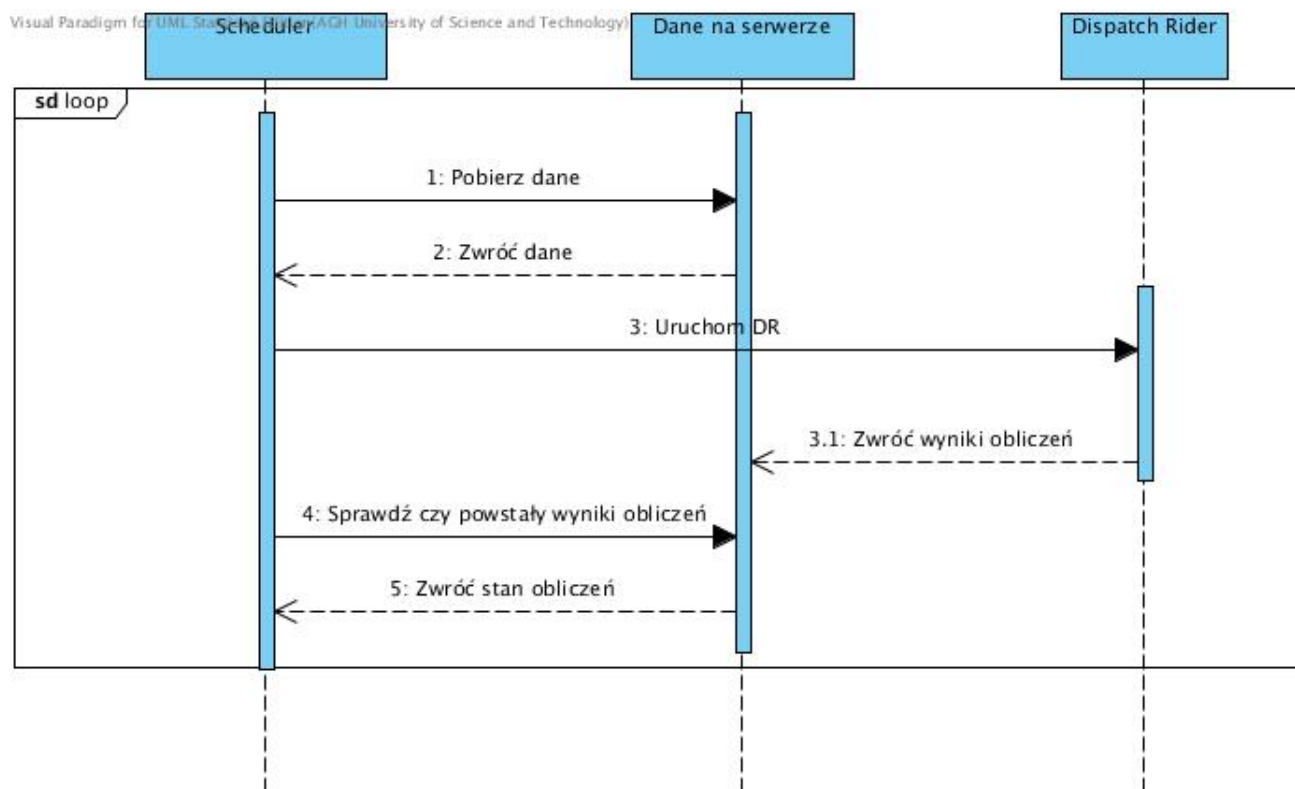
Diagram ukazujący przebieg ładowania danych na serwer z użyciem aplikacji klienckiej. Dane przesyłane są na serwer, gdzie uruchamiany jest napisany w języku Python skrypt, zwracający mapę tras.



Rysunek 5.2: Diagram ładowania danych na serwer

### 5.5.2. Główna pętla programu

Diagram ukazujący główną pętlę w której działa program. Scheduler pobiera dane umieszczone na serwerze za pomocą aplikacji klienta, i uruchamia za ich pomocą Dispatch Ridera. Następnie sprawdza czy powstały wyniki obliczeń, zwraca je do aplikacji klienckiej.



Rysunek 5.3: Diagram działania Schedulera

## 6. Oprogramowanie schedulera

W celu obsługi kolejnych zadań przesyłanych do systemu, powstał szkielet oprogramowania zarządzającego kolejkowaniem oraz uruchamianiem Dispatch Ridera z kolejnymi zestawami danych.

Skrypt przeszukuje podany w parametrach startowych katalog systemu plików, poszukując nowo dodanych zestawów danych. W przypadku gdy taki zestaw się pojawia, umieszcza plik 'configuration.xml' zestawu w kolejce zadań do wykonania.

Następnie scheduler sprawdza czy pojawiły się pliki wynikowe działania programu, co jest jednoznaczne z zakończeniem wykonywania danego zadania. Jeśli pliki znajdują się w katalogu wynikowym, wówczas uruchamiane jest kolejne zadanie z kolejki.

Kwestią istotną jest fakt, iż aby system zadziałał, plik `configuration.xml` musi znajdować się w katalogu głównym Dispatch Ridera. Ów wymóg jest podyktowany zapewne założeniem projektowym lub błędem DR. W związku z tym, plik ten dla każdego nowo uruchamianego zadania jest kopiowany do głównego katalogu przed uruchomieniem procesu.

Kwestią pozostałą do uzgodnienia pozostaje w tej chwili struktura katalogów tworzona podczas ładowania plików z danymi do systemu - scheduler działa poprawnie, jednak konieczne jest wypracowanie polityki zarządzania danymi wejściowymi. Dyskusję dotyczącą tego zagadnienia przeprowadzimy podczas tworzenia systemu wczytywania i parsowania danych wejściowych.

### **Dodane 25.05.2012: Problem z uzyskaniem wyników działania programu**

Jak pisaliśmy powyżej, do uruchomienia Dispatch Ridera wykorzystujemy skrypt napisany w języku Java. Wykonywane jest następujące polecenie systemowe:

```
command = "java -cp " + mainPath + "/jar/DTP.jar jade.Boot -nomtp  
TestAgent:ctp.jade.test.TestAgent(" + mainPath + "/configuration.xml)  
InfoAgent:ctp.jade.info.InfoAgent  
DistributorAgent:ctp.jade.distributor.DistributorAgent  
CrisisManagerAgent:ctp.jade.crisismanager.CrisisManagerAgent";
```

Niestety program uruchamiany w ten sposób - który zgodnie z kodem wcześniej po-

wstałego GUI jest sposobem poprawnym, nie generuje w efekcie żadnych wyników. Pliki które powinny pojawić się w katalogu wyjściowym nie są wogóle tworzone. Problem z całą pewnością leży po stronie oprogramowania DispatchRidera, a konkretnie w sposobie uruchomienia za pomocą linii poleceń, ponieważ ten sam plik configuration.xml uruchomiony przy pomocy frameworka JADE powoduje generowanie wyników. W celu wyjaśnienia tej sytuacji skontaktowaliśmy się z autorami oprogramowania.

### **Dodane 14.06.2012: Problemy z uruchomieniem DR przy pomocy Schedulera - ciąg dalszy**

W celu rozwiązania problemów z uruchamianiem DR nawiązaliśmy kontakt z autorem oprogramowania - Sebastianem Pisarskim. Niestety, nie był on w stanie podać rozwiązania naszego problemu ani sposobu uruchamiania DR z linii poleceń systemu (z podaniem ścieżki do pliku configuration.xml). Co więcej, w chwili obecnej nie jesteśmy nawet w stanie uruchomić z linii poleceń DR, mimo iż we wcześniejszej wersji ( z marca br. ) w losowo wybranych momentach aplikacja odpalana przy pomocy Schedulera zaczynała działać.

Podsumowując prace nad oprogramowaniem Schedulera - mimo poprawności stworzonego przez nas kodu, obsługującego operacje na systemie plików, nie udało nam się uruchomić tej funkcjonalności. Przeszkodą nie do pokonania okazało się w tym przypadku uruchomienie oprogramowania DR z linii komend - sposób wykorzystywany przez autorów wcześniej powstałego GUI webowego nie działa, a sam autor Dispatch Ridera nie był w stanie poinformować nas jaki jest poprawny sposób uruchomienia aplikacji. Również mimo podjętych wielokrotnie prób użycia różnorodnych parametrów, nie udało nam się doświadczalnie uruchomić programu. Uznajemy że wykorzystaliśmy wszelkie możliwości na poznanie sposobu uruchomienia programu, i bez wypracowania polityki uruchomienia przez autora DR, scheduler nie będzie mógł zadziałać. Problem uruchomienia systemu nie leży po stronie naszego webowego klienta, lecz niestety po stronie samego Dispatch Ridera.

## 7. Instrukcja obsługi

### 7.1. Uruchomienie aplikacji

Poniżej przedstawiamy krótką instrukcję obsługi systemu przez nas zbudowanego. Instrukcja dotyczy części webowej. Niestety nie udało nam się uruchomić schedulera z linii poleceń, więc nie opisujemy co zrobić by system działał razem z nim.

Do uruchomienia aplikacji webowej potrzebujemy uruchomiony serwer PHP oraz interpreter Pythona. Całą strukturę katalogów przenosimy do folderu ze stronami naszego serwera PHP (tak, aby móc się do niego dostać poprzez adres `http://localhost/nazwa-katalogu`). Aplikację powinno się uruchamiać w przeglądarce wspierającej HTML5. My testowaliśmy ją w przeglądarce Google Chrome (v. 19), w której sprawowała się bardzo dobrze. Jeśli wszystko pójdzie sprawnie powinniśmy ujrzeć ekran powitalny. Po najechnaniu myszką w napis **Web Dispatch Rider** pojawi nam się menu z trzema pozycjami na liście.

### 7.2. Dodawanie zadania

W tym widoku użytkownik może dodać nowe zadanie do systemu poprzez zdefiniowanie jego nazwy oraz wybranie odpowiednich plików dotyczących zadania. System nie pozwoli na wysłanie plików w niepoprawnym formacie, bądź w sytuacji, w której nie zostaną podane wszystkie pliki przez użytkownika. W przypadku poprawnego wykonania się operacji przesłania plików użytkownik otrzyma na ten temat stosowny komunikat, a skrypt Pythonowski utworzy na serwerze plik w JavaScriptcie ze współrzędnymi odzwierciedlającymi mapkę danego zadania.

### 7.3. Zadania w trakcie

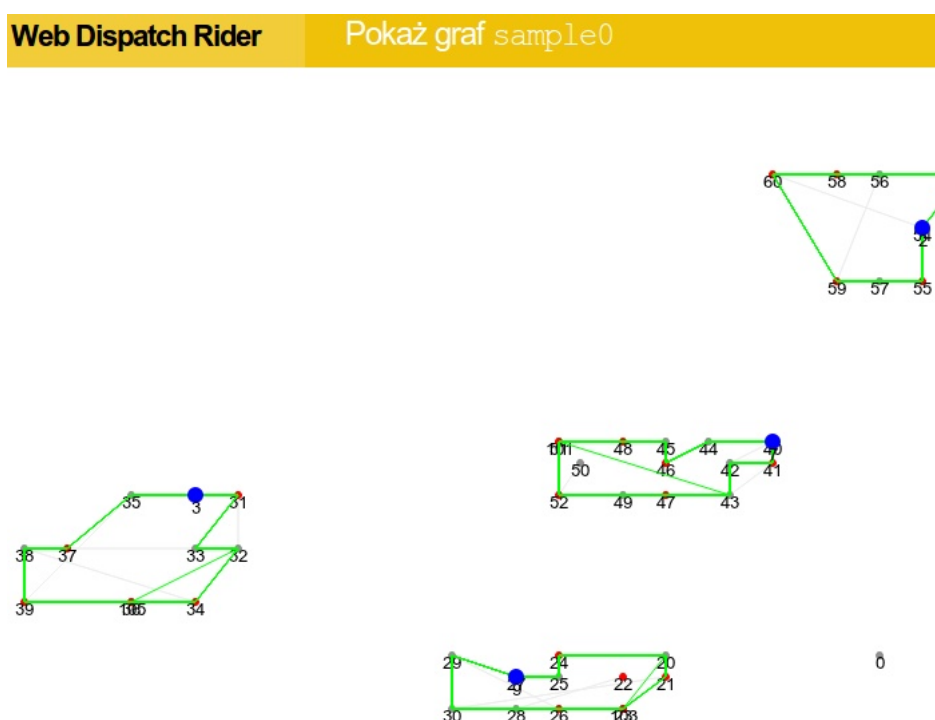
W tym miejscu użytkownik może podejrzeć listę zadań, które czekają w kolejce na bycie przetworzonym przez zewnętrzny moduł Dispatch Ridera. Do momentu rozpoczęcia przetwarzania użytkownik ma możliwość edycji tych plików zgodnie z własną wolą. Kiedy system przejmie zadanie, wówczas edycja nie jest już możliwa.

## 7.4. Zadania obliczone

Tutaj znajduje się lista zadań gotowych do obejrzenia przez użytkownika. Może on obejrzeć graf dotyczący danego zadania lub usunąć zadanie. Dane te są dostępne dla użytkownika cały czas. W przypadku gdy zadanie jest jeszcze liczone, bądź czeka w kolejce użytkownikowi wyświetla się tylko mapka dotycząca danego zadania.

### 7.4.1. Wyświetlanie grafu

Użytkownik wybierając podgląd grafu danego zadania może zobaczyć jego mapkę oraz przebieg tras wszystkich kierowców. Mapa pokazuje miejsca z podziałem na miejsca dostarczenia przesyłek i miejsca ich odbioru.



Rysunek 7.1: Diagram komponentów systemu

### 7.4.2. Zmiana sposobu wyświetlania grafu

W dokumentacji należy się również słowo wyjaśnienia odnośnie sposobu wyświetlania grafu. Na początku semestru uznaliśmy, że skorzystamy z gotowych bibliotek JavaScriptowych analizując je i wybierając naszym zdaniem najlepszą. Wówczas jednak nie byliśmy

do końca świadomi grafu jaki powinniśmy uzyskać w budowanym przez nas systemie. Biblioteka, którą wybraliśmy nie umożliwiała nam rysowanie grafów/węzłów w określonych przez nas miejscach ekranu. W związku z tym postanowiliśmy bliżej zaznajomić się z technologią Canvas, którą dostarcza HTML5. Udało nam się tę znajomość zakończyć pozytywnie i uzyskać poruszający się w przeglądarce graf.

## 8. Podsumowanie

W pracy nad interfejsem webowym dla systemu planowania transportu Dispatch Rider odnieśliśmy częściowy sukces. Udało nam się spełnić większość spośród założonych na początku planowania projektu wymagań funkcjonalnych, jednak z przyczyn od nas niezależnych nie byliśmy w stanie uruchomić całej architektury naszej warstwy prezentacji. Naszym celem było stworzenie lekkiej, możliwie niezależnej od głównego systemu Dispatch Rider, aplikacji klienckiej służącej do uruchamiania DR oraz prezentowania wyników jego pracy.

Decyzja o budowie nowej aplikacji klienckiej zamiast rozwijania dotychczas istniejącej była motywowana problemami z uruchomieniem owej już istniejącej wersji webowego gui. Jak wspominaliśmy na początku dokumentacji, nie byliśmy w stanie uruchomić jej na części systemów operacyjnych/przeglądarek, a nawet po uruchomieniu nie była ona w stanie komunikować się z DR.

W chwili obecnej, bogatsi o doświadczenia z pracy nad aplikacją, możemy stwierdzić iż wina leżała nie tylko po stronie aplikacji webowej, ale i samego Dispatch Ridera.

Program nie posiada wyspecyfikowanych interfejsów umożliwiających komunikację z systemem webowym. W związku z tym, podobnie jak nasi poprzednicy, postanowiliśmy operować na systemie plików, w odpowiedni sposób operując danymi wejściowymi oraz wyjściowymi. Niestety, nie udało się nam uruchomić z powodzeniem DR z linii poleceń systemu - co jest warunkiem koniecznym w przypadku naszej architektury. Mimo wielokrotnych prób, eksperymentów nie udało nam się wypracować sposobu uruchomienia, co więcej, sam autor systemu nie był w stanie poinformować nas o poprawnym sposobie uruchamiania aplikacji Dispatch Rider.

Mimo powyższych problemów - które niestety uniemożliwiają nam zaprezentowanie wszystkich funkcjonalności naszej aplikacji - udało nam się spełnić pozostałe zdefiniowane przez nas jako najważniejsze i zaakceptowane przez prowadzących wymagania funkcjonalne. Aplikacja jest w stanie przysyłać oraz wizualizować dane, i może stać się w przyszłości w pełni użyteczna pod warunkiem udostępnienia przez autorów systemu Dispatch Rider interfejsów umożliwiających pewną, skuteczną komunikację z systemem.

Możemy uznać że prace zakończyliśmy z sukcesem o tyle, o ile pozwoliły nam na to warunki zewnętrzne. Niestety, sprawdziły się przewidywania poczynione na początku zapoznania z systemem, w których wyrażaliśmy obawy iż sygnalizowane już przez autorów poprzedniej



wersji interfejsu webowego problemy z działaniem systemu mogą uniemożliwić działalność części systemu. Za wyjątkowo rozczarowujący uznajemy fakt iż problem sprawiły nam te same kwestie które wymienione były przez autorów wcześniejszej wersji GUI, co oznacza iż nie zostały one naprawione przez rok w którym oprogramowanie DR było ciągle rozwijane.

## 8.1. Analiza spełnienia wymagań funkcjonalnych

Generowanie pliku określającego zlecenia, o składni: ilość-pojazdów ładowność prędkość	TAK
Generowanie pliku określającego czas nadchodzenia zleceń, o składni: nr-lini czas-zlecenia	TAK
Generowanie pliku zawierającego liczbę kierowców	TAK
Generowanie pliku określającego parametry ciągników, o składni: moc niezawodność wygoda zużycie-paliwa typ-zaczepu	TAK
Generowanie pliku opisującego parametry naczep o składni: masa pojemność typ-ładunku uniwersalność typ-zaczepu	TAK
Generowanie pliku opisującego parametry holonów o składni: initialCapacity = pojemność-pojazdu mode = tryb bases = ilość-baz eUnitsCount = ilość-jednostek	TAK
Generowanie pliku konfiguracyjnego configuration.xml	TAK
Kolejkowanie zadań zleczanych systemowi	NIE
Uruchamianie systemu Dispatch Rider z użyciem dostarczonych z zewnątrz lub wygenerowanych plików	NIE
Wizualizacja grafu sieci transportowej	TAK
Wizualizacja tras pokonanych przez pojazdy	TAK
Wyświetlanie informacji o każdym z holonów	TAK
Przechowywanie wyników obliczeń, w celu zaprezentowania na żądanie	TAK

Tabela 8.1: Analiza spełnienia wymagań funkcjonalnych

Jak więc widać z powyższej tabeli większość funkcjonalności udało nam się uzyskać. Te, które się nie powiodły wynikły z problemów związanych z dostarczonym systemem Dispatch Rider, bądź złym rozplanowaniem czasu wykonania projektu. Mimo wszystko, najważniejsze wymagania, takie jak ładowanie zadań do systemu oraz wizualizacja tras jak najbardziej udało nam się uzyskać, dzięki czemu oceniamy projekt jako zakończony sukcesem.

## 8.2. Napotkane błędy systemu DispatchRider

— System nie uruchamia się wogóle przy pomocy linii poleceń w przypadku gdy plik 'configuration.xml' znajduje się gdzie indziej niż katalog główny programu. Nie mają na to

wpływu zawarte w nim ścieżki względne, gdyż nawet po zamianie ich na bezwzględne, plik ten musi zostać skopiowany do katalogu głównego by móc go uruchomić.

- Brak generowania pliku wynikowego. W trakcie prac na Schedulerem napotkaliśmy na (wspomnianą już przez zespół Baran-Patrzyk) sytuację gdy Dispatch Rider nie generował plików wynikowych nawet wówczas gdy udawało nam się uruchomić system. Problem ten w efekcie uniemożliwił nam zrealizowanie wszystkich funkcjonalności systemu.
- Problem uruchomienia GUI wbudowanego w Dispatch Riderze. Do pewnego momentu prac, udawało nam się uruchamiać Dispatch Ridera za pomocą komendy **java -jar GUI.jar**. Niestety, w kolejnej wersji systemu funkcjonalność ta przestała działać, i otrzymywaliśmy (na obydwu stacjach roboczych) błąd **Problem with reading file:xmlschemas/configuration.xsd (No such file or directory)** . Autor systemu nie był w stanie nam pomóc, stwierdzając że u niego dana funkcjonalność działa.

# Spis rysunków

4.1	Strony powitalna . . . . .	20
4.2	Lista zadań . . . . .	21
4.3	Menu aplikacji . . . . .	21
4.4	Widok grafu . . . . .	22
5.1	Diagram komponentów systemu . . . . .	23
5.2	Diagram ładowania danych na serwer . . . . .	25
5.3	Diagram działania Schedulera . . . . .	26
7.1	Diagram komponentów systemu . . . . .	30

# Spis tabel

2.1	JavaScript InfoVis Toolkit - Informacje Podstawowe . . . . .	7
2.2	JSViz - Informacje Podstawowe . . . . .	8
2.3	Dracula - Informacje Podstawowe . . . . .	8
2.4	JavaScript InfoVis Toolkit - Funkcjonalność . . . . .	9
2.5	JSViz - Funkcjonalność . . . . .	9
2.6	Dracula - Funkcjonalność . . . . .	9
2.7	JavaScript InfoVis Toolkit – Wydajność . . . . .	10
2.8	JSViz – Wydajność . . . . .	10
2.9	Dracula – Wydajność . . . . .	10
8.1	Analiza spełnienia wymagań funkcjonalnych . . . . .	33

# Bibliografia

- [1] Gołacki M.: *Modelowanie Transportu z użyciem Holonów*. Kraków, wrzesień 2009.
- [2] Konieczny M.: *Modelowanie i optymalizacja transportu w sytuacjach kryzysowych*. Kraków, 2008.
- [3] Miąsko T., Pisarski S.: *Dispatch Rider. Podejście holoniczne w problemie transportowym*. Dokumentacja projektu KI AGH 2010.