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 **18 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

Kraków, czerwiec 2012

Spis treści

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 pomyślne 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 uruchamieniem 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 kończącą 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 klęsce. 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

Adres	http://thejit.org/
Licencja	new BSD license - możliwe kopiowanie, modyfikacja, rozpowszechnianie, sprzedaż z zachowaniem informacji o autorze; więcej: https://github.com/philogb/jit/blob/master/LICENSE
Rozszerzalność	Pod względem prawnym – rozszerzalność możliwa. Pod względem technicznym – kilka tysięcy linii kodu, jednakże brak sformalizowanej dokumentacji – polega ona na komentarzach w kodzie
Inne uwagi	Bardzo ładna, używana m.in. na stronach Mozilli oraz prezydenta USA. Dokumentacja korzystania z biblioteki: http://thejit.org/static/v20/Docs/index/General.html. Źródła: https://github.com/philogb/jit
Ocena	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

Adres	http://code.google.com/p/jsviz/
Licencja	Apache 2.0 - dopuszcza użycie kodu źródłowego zarówno na potrzeby wolnego oprogramowania, jak i zamkniętego oprogramowania komercyjnego.
Rozszerzalność	Licencja zezwala, aczkolwiek brak dokumentacji, jedynie komentarze w kodzie o długości kilku/kilkunastu tysięcy linii.
Inne uwagi	Niedostatecznie przyjemna dla oka.
Ocena	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

Adres	http://www.graphdracula.net/
Licencja	MIT (X11) license - możliwe są kopiowanie, modyfikacja, rozpowszechnianie, w tym sprzedaż.
Rozszerzalność	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.
Inne uwagi	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

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

Tabela 2.4: JavaScript InfoVis Toolkit - Funkcjonalność

2.2.2. JSViz

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

Tabela 2.5: JSViz - Funkcjonalność

2.2.3. Dracula

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

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** 0-3 dla każdego węzła
100	Generuję się od razu, można bez problemu korzystać.	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. Kolejkowanie 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.

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

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

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

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

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

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

Wyświetlanie informacji o każdym z holonów

Cel przypadku użycia: Sprawdzenie przez użytkownika danych pojedyńczego 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

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.

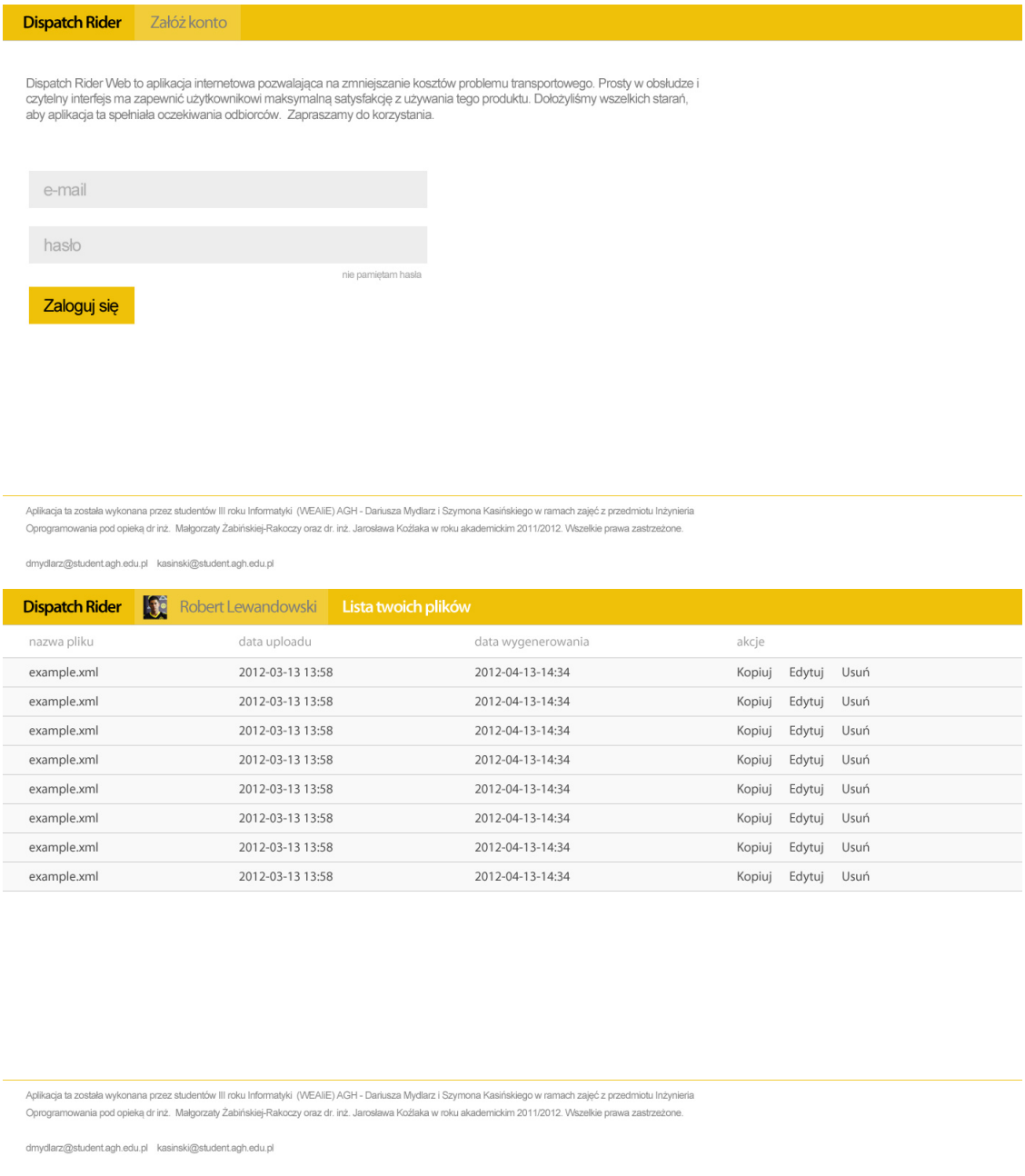
Scenariusz:

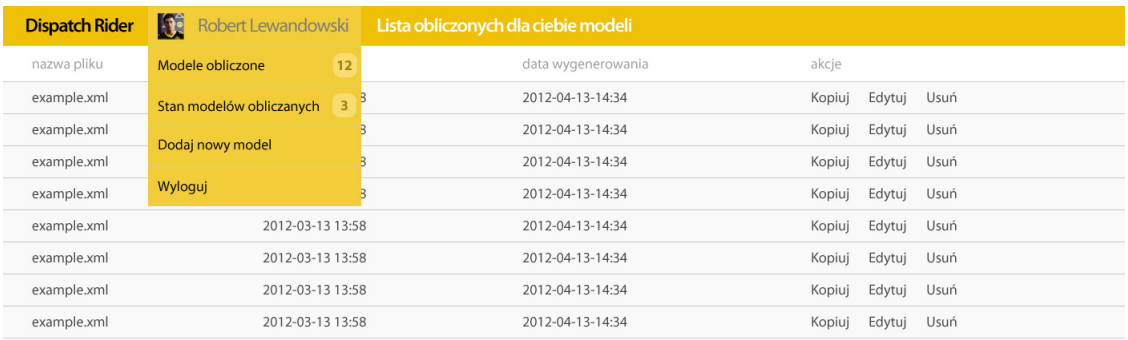
1. Użytkownik przełącza się na zakładkę wprowadzania danych.
2. Użytkownik wybiera plik lub wprowadza dane w pola tabeli

3. Aplikacja umieszcza dane w folderze uploads

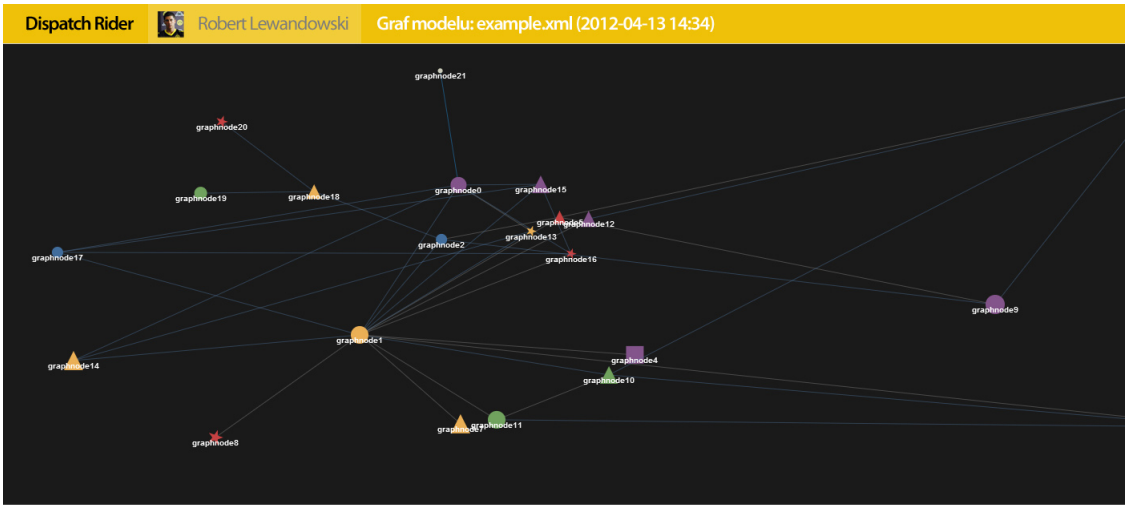
4.4. Projekt GUI

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.





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



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

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

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. 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`.

5.2. Scheduler

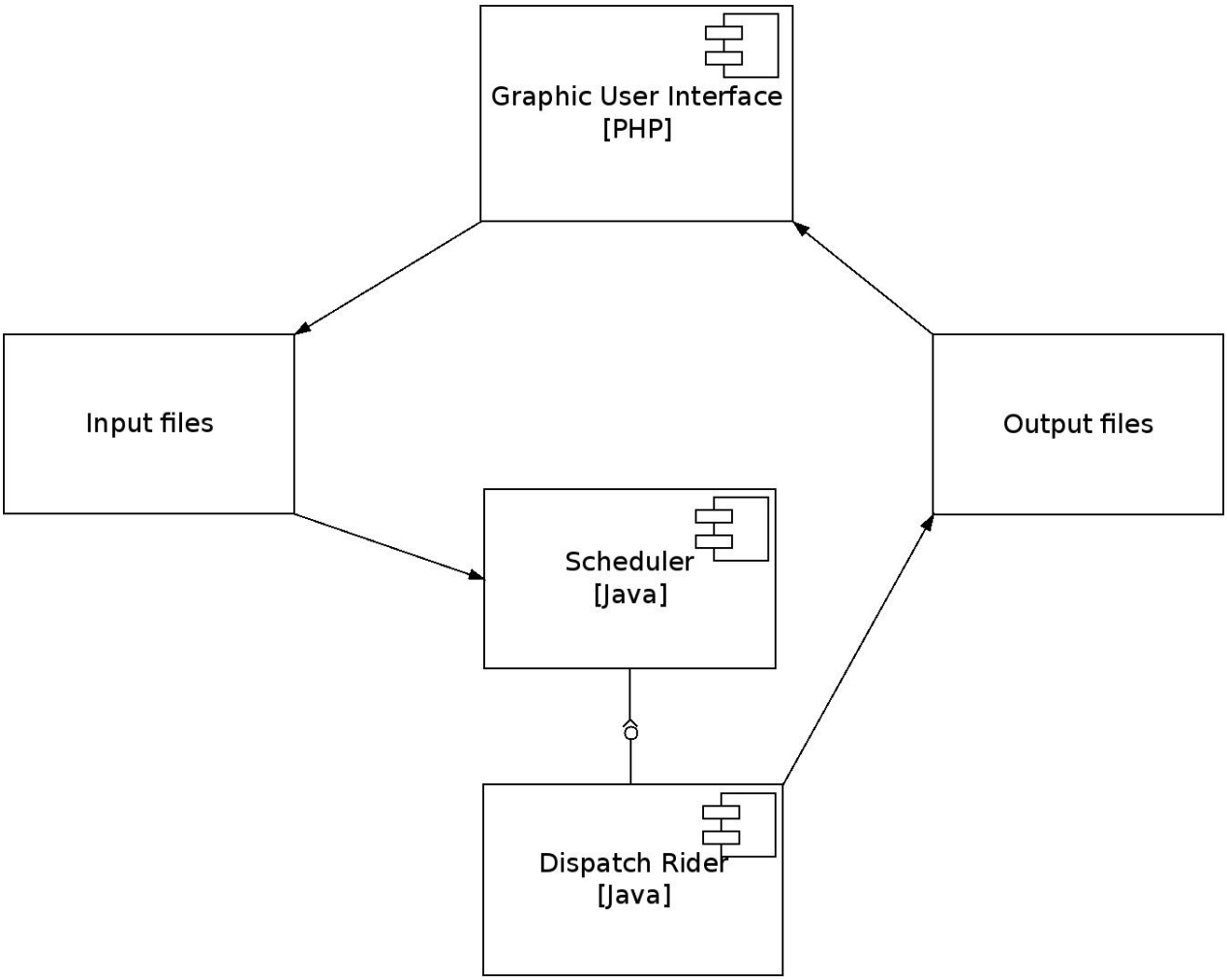
Scheduler to osobna aplikacja napisana w języku Java służąca do kolejkowania zadań przesłanych do systemu przez użytkownika. Skanuje ona zawartość folderów `uploads` i `ready` z aplikacji internetowej i zarządza ich przesyłaniem do Dispatch Ridera. Scheduler jest więc łącznikiem pomiędzy usługą kliencką, a serwerową. W momencie ukończenia przetwarzania problemu przez Dispatch Ridera scheduler przenosi pliki wynikowe do od-

powiedniego podfolderu katalogu `ready` i w sytuacji gdy jakiś problem czeka już w kolejce do obliczeń, przesyła go do Dispatch Ridera.

5.3. 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, nię będziemy ingerować w jej kod źródłowy, a jedynie skorzystamy z jej plików wynikowych. Aplikacja jest uruchamiana i zarządzana przez schedulera.

5.4. Diagram komponentów systemu



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 uruchomienim 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:dtp.jade.test.TestAgent(" + mainPath +"/configuration.xml)
 InfoAgent:dtp.jade.info.InfoAgent
 DistributorAgent:dtp.jade.distributor.DistributorAgent
 CrisisManagerAgent:dtp.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 różnorakich 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. 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 przesyłać 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.

7.1. 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 zlecanych 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

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.

7.2. 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 ge-

nerował 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 otrzymywalismy (na obydwu stacjach roboczych) błąd **Problem with reading file:xmlschemes/configuration.xsd (No such file or directory)** . Autor systemu nie był w stanie nam pomóc, stwierdzając że u niego dana funkcjonalność działa.

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.