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 **27 maja 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. Ustalenia

1. Otrzymany projekt nie zachowuje się deterministycznie pod różnymi platformami. Pod Linuxem (Ubuntu) uruchomione zadanie nie oblicza się, natomiast pod Windowsem (7) GUI uruchamia się niedeterministycznie. Nie jesteśmy w stanie określić co powoduje owe problemy, jednakże mogą one uniemożliwić rozszerzenie funkcjonalności GUI. 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. Z ich uwag wynika iż DispatchRider posiada błędy, które w szczególności mogą skutkować brakiem pliku wynikowego, nieuwzględnieniem sieci transportowej, nieuwzględnieniem czasu utworzenia holonu.
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.

1.1.2. 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 w.w. 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 niedeterministyczną. 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 przeglądanych 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 dość 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:

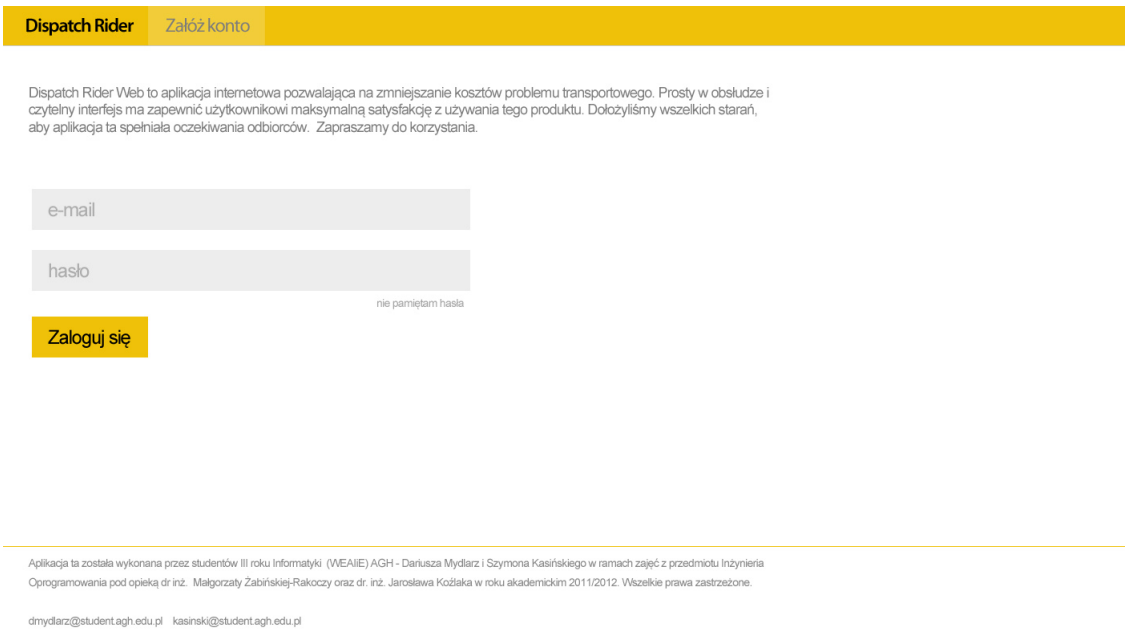
1. Generowanie plików konfiguracyjnych dla Dispatch Ridera:
   - generowanie pliku określającego zlecenia, o składni: **ilość-pojazdów ładowność prędkość**
   - generowanie pliku określającego czas nadchodzenia zleceń, o składni: nr-lini **czas-zlecenia**
   - generowanie pliku zawierającego liczbę kierowców
   - 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
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

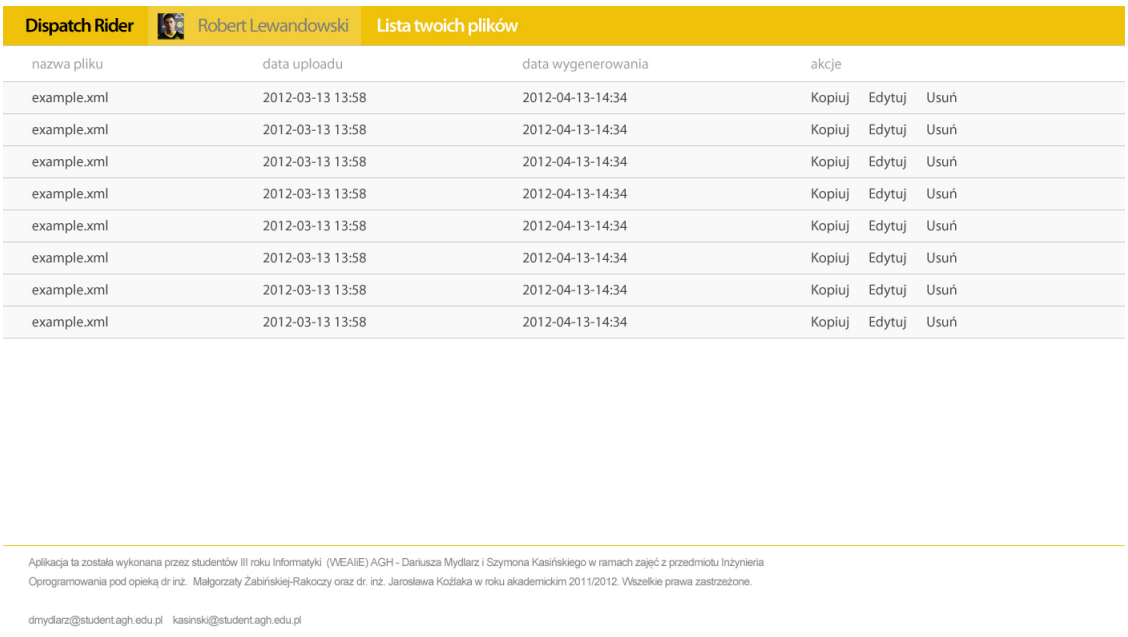
4.2. Wymagania niefunkcjonalne

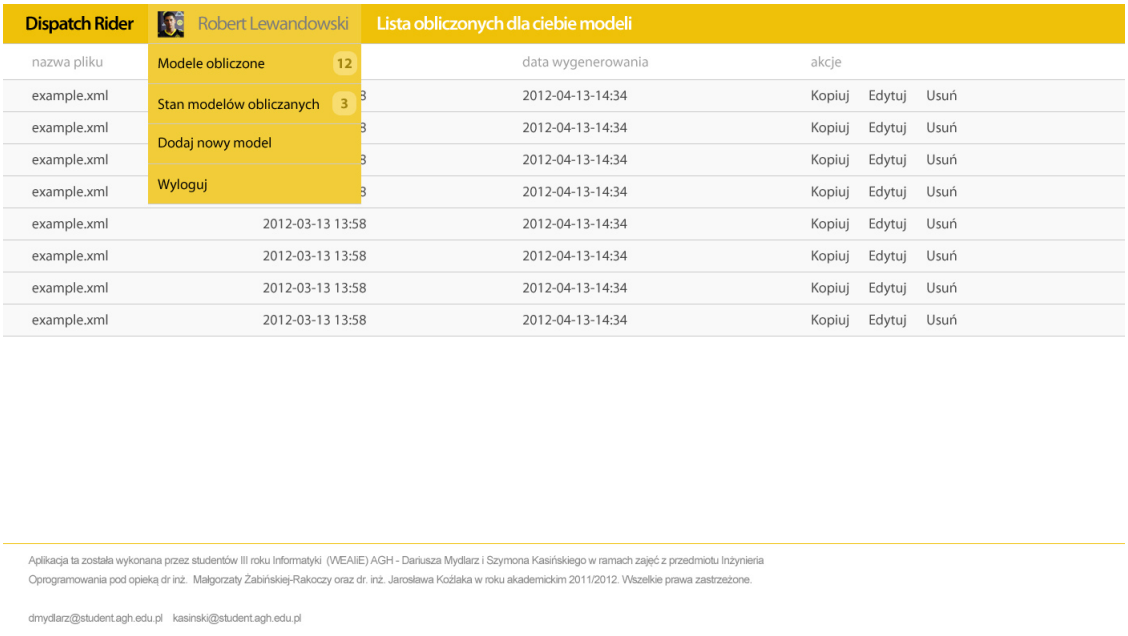
1. Bezpieczeństwo zapewniane poprzez autoryzację użytkowników.
2. Dokumentacja techniczna w postaci strony wiki, dokumentu tekstowego oraz komentarzy w kodzie programu.
3. Przenośność - zapewnienie w pełni poprawnego działania programu na najpopularniejszych przeglądarkach oraz systemach operacyjnych.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

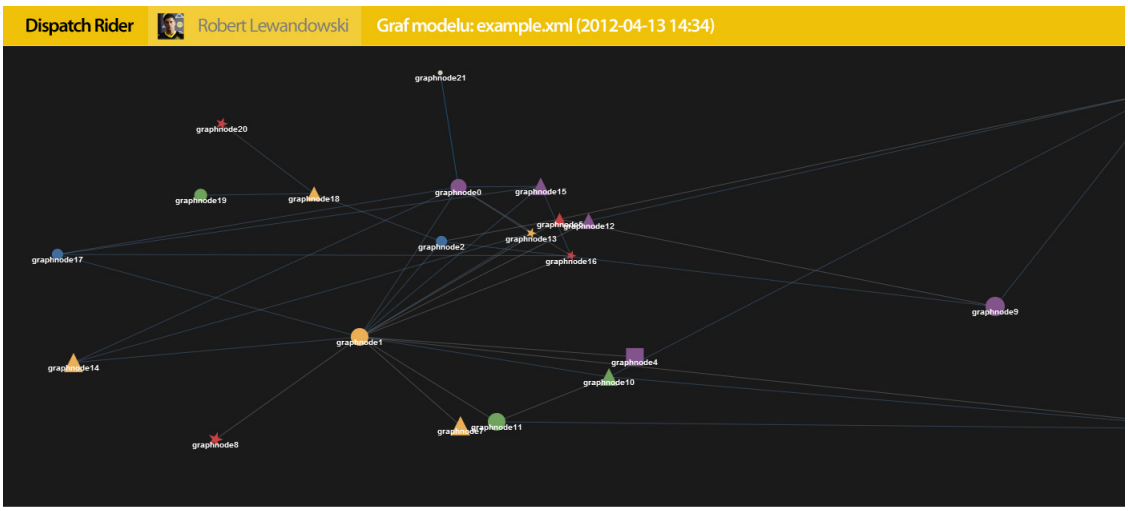
4.3. Projekt GUI

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
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
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


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



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

5. 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 zakonczeniem 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 jest sposobem poprawnym, gdyż powoduje wyświetlenie okna spisu e-unitów oraz wykresu - nie generuje w efekcie żadnych wyników. Pliki które powinny pojawić się w katalogu wyjściowym nie są wogóle tworzone.

Problem z całą pewnością leży w sposobie uruchamiania, ponieważ ten sam plik configuration.xml uruchomiony przy pomocy oryginalnego GUI powoduje generowanie wyników. W celu wyjaśnienia tej sytuacji skontaktowaliśmy się z autorami oprogramowania.

6. Architektura systemu

Architektura budowanego systemu nie będzie skomplikowana. Całość uruchamiana będzie na jednej maszynie – serwerze. Całość, to znaczy: aplikacja internetowa, scheduler oraz Dispatch Rider.

6.1. Aplikacja internetowa

Aplikacja internetowa to miejsce, w którym użytkownik będzie mógł łączyć się z systemem. 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 wylądują 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`.

6.2. Scheduler

Scheduler to osobna aplikacji napisana w Javie 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 odpowiedniego podfolderu katalogu `ready` i jeśli jakiś problem czeka już w kolejce do obliczeń, to przesyła go do Dispatcha.

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

Bibliografia

[1] Gołacki M.: *Modelowanie Transportu z użyciem Holonów*. Kraków, wrzesień 2009.

[2] Konieczny M.: *Modelowanie i optymalizacja transportu w sytuacjach kryzysowych*. Kraków, 2008.