

**Akademia Górniczo-Hutnicza
w Krakowie**

Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki



Katedra Informatyki

*Dispatch Rider.
Uniwersalny interfejs dla potrzeb problemów transportowych*

Dokumentacja projektu

Kierunek, rok studiów:

Informatyka, III rok (2011/2012 – semestr letni)

Przedmiot:

Inżynieria Oprogramowania

Prowadzący zajęcia:

Dr inż. Małgorzata Żabińska-Rakoczy, Dr inż. Jarosław Koźlak

Autorzy:

Jakub Tyrcha, Tomasz Put

1. Cel projektu

Celem naszego projektu jest rozbudowa klasycznego graficznego interfejsu dla procesu symulacji procesów transportowych. Jest więc w całości oparty o już istniejący system agentowy DispatchRider [1][2][3][6], co w znacznej mierze wpływa na specyfikę planowanych prac. Tworzony interfejs powinien na bieżąco informować użytkownika o stanie symulacji i obrazować zależności zachodzące między agentami, oraz graf rozwiązania. Powinien również uwzględniać dodatkowe elementy takie jak miękkie okna czasowe, spóźnienia, graf reprezentujący sieć drogową oraz zmienne czasy przejazdów.

2. Wizja rozwoju

2.1. Koncepcyjna

Na stan obecny naszym planem jest powrót do zarzuconej koncepcji klienta GUI w formie agenta JADE [4], przekazywanego platformie w parametrach uruchomienia. To podejście powinno zapewniać wymaganą elastyczność i pozwolić na komunikację bezpośrednio na poziomie aplikacji, bez potrzeby wykorzystywania zrzucanych przez system plików xml.

2.2. Techniczna

- napisanie modułu (lub zmodyfikowanie obecnych klas), w celu uzyskania funkcjonalności dostępu do danych symulacji po każdym przeprowadzonym kroku
- stworzenie GUI w formie obiektu z wykorzystaniem wzorca singleton, który wykorzystuje ww. moduł do aktualizowania wyświetlanego stanu wraz z postępem symulacji

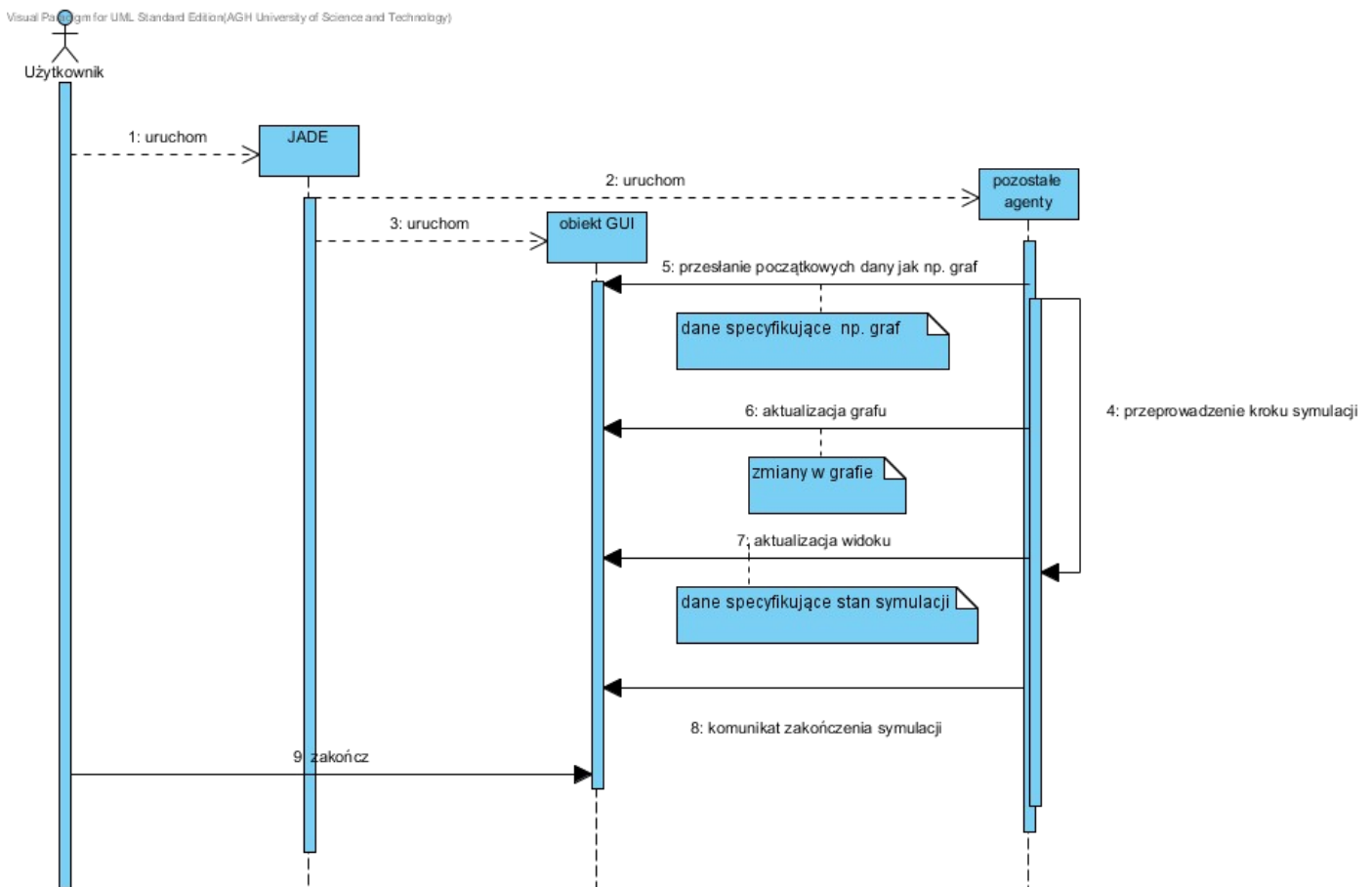
3. Wstępny harmonogram prac

Nazwa	Data rozpoczęcia	Data zakończenia	Opis
Zapoznanie się z działaniem systemu	26 marca	2 kwietnia	Przed przystąpieniem do tworzenia funkcjonalności musimy dokładnie zapoznać się z ogólną zasadą działania systemu, interfejsami, plikami konfiguracyjnymi itp.
Wydobycie z systemu parametrów potrzebnych do rozszerzenia o dodatkowe elementy np. miękkie okna	2 kwietnia	16 kwietnia	Ponieważ naszym zadaniem jest m. in. poszerzenie funkcjonalności, musimy uzupełnić obecne klasy o uwzględnianie istotnych dla wizualizacji parametrów.
Wstępna wersja GUI zsynchronizowana z JADE	16 kwietnia	30 kwietnia	Zapoznanie się z technologią i wstępne wdrożenie agentowego GUI dla JADE.
Rozszerzenie GUI o wymaganą funkcjonalność	30 kwietnia	28 maja	Uzupełnienie GUI o wymagane elementy.
Testy i poprawki	28 maja	18 czerwca	Sprawdzenie poprawności działania i wprowadzenie poprawek.
Tworzenie dokumentacji	16 kwietnia	18 czerwca	Tworzenie dokumentacji modułu na bieżąco, w trakcie prac nad projektem.

4. Koncepcja, cykl życia agenta GUI

Podstawowym zadaniem naszego agenta jest graficzna wizualizacja przebiegu symulacji. W tym celu należy wydzielić logiczną strukturę opisu stanu symulacji w danym kroku, która będzie podstawą interakcji między agentem GUI, a pozostałymi agentami, oraz będzie zapewniała niezbędne dane do wyświetlania pożądaných informacji.

Cykl życia naszego agenta GUI rozpocznie się w momencie włączenia programu Dispatch Rider przez użytkownika, z uwzględnieniem podania odpowiedniej klasy w parametrach uruchomienia, oraz dostarczenia pliku konfiguracyjnego. Za rozpoczęcie działania bezpośrednio odpowiedzialny będzie system JADE. Po inicjalizacji powinno ukazać się główne okno programu wraz ze skalą czasu. W trakcie działania holony będą tworzyły przejściowe rozwiązania, czasowo zhierarchizowane w postaci kroków. Co każdy krok nastąpi przesłanie opisu stanu symulacji do agenta GUI, na którym spoczywa odpowiedzialność za aktualizowanie wyświetlanych danych. Po zakończeniu symulacji okno poinformuje o tym fakcie użytkownika i pozostanie włączone. Decyzja o zamknięciu aplikacji będzie należała do użytkownika.

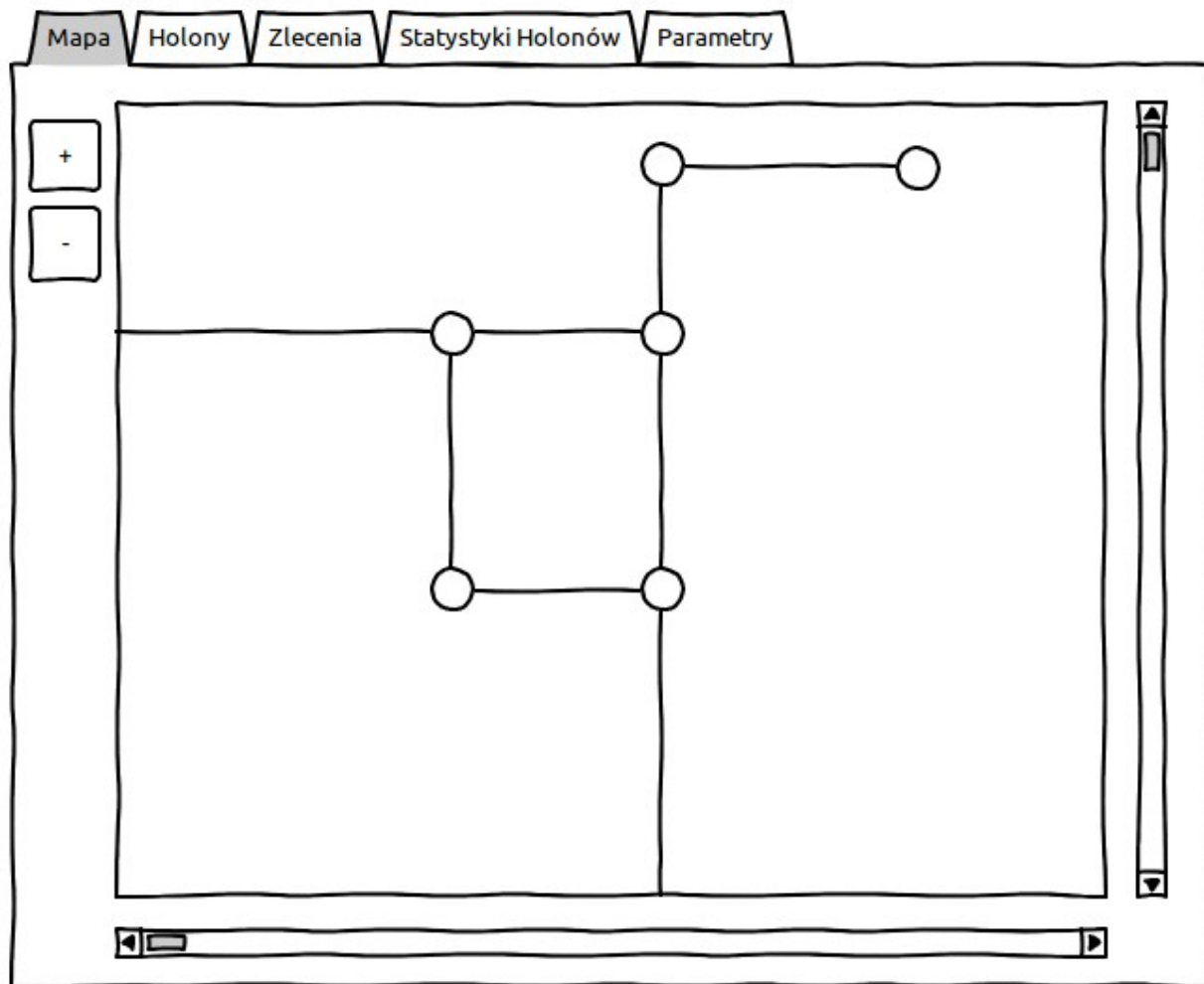


5. Projekt GUI

Po uruchomieniu symulacji planujemy wyświetlić główne okno z pięcioma zakładkami: Mapa, Holony, Zlecenia, Statystyki Holonów oraz Parametry. W trakcie prac projektowych konieczne okazało się utworzenie dodatkowej zakładki „Trasy”, nie jest ona opisana w projekcie, a dopiero w dokumentacji deweloperskiej i użytkownika.

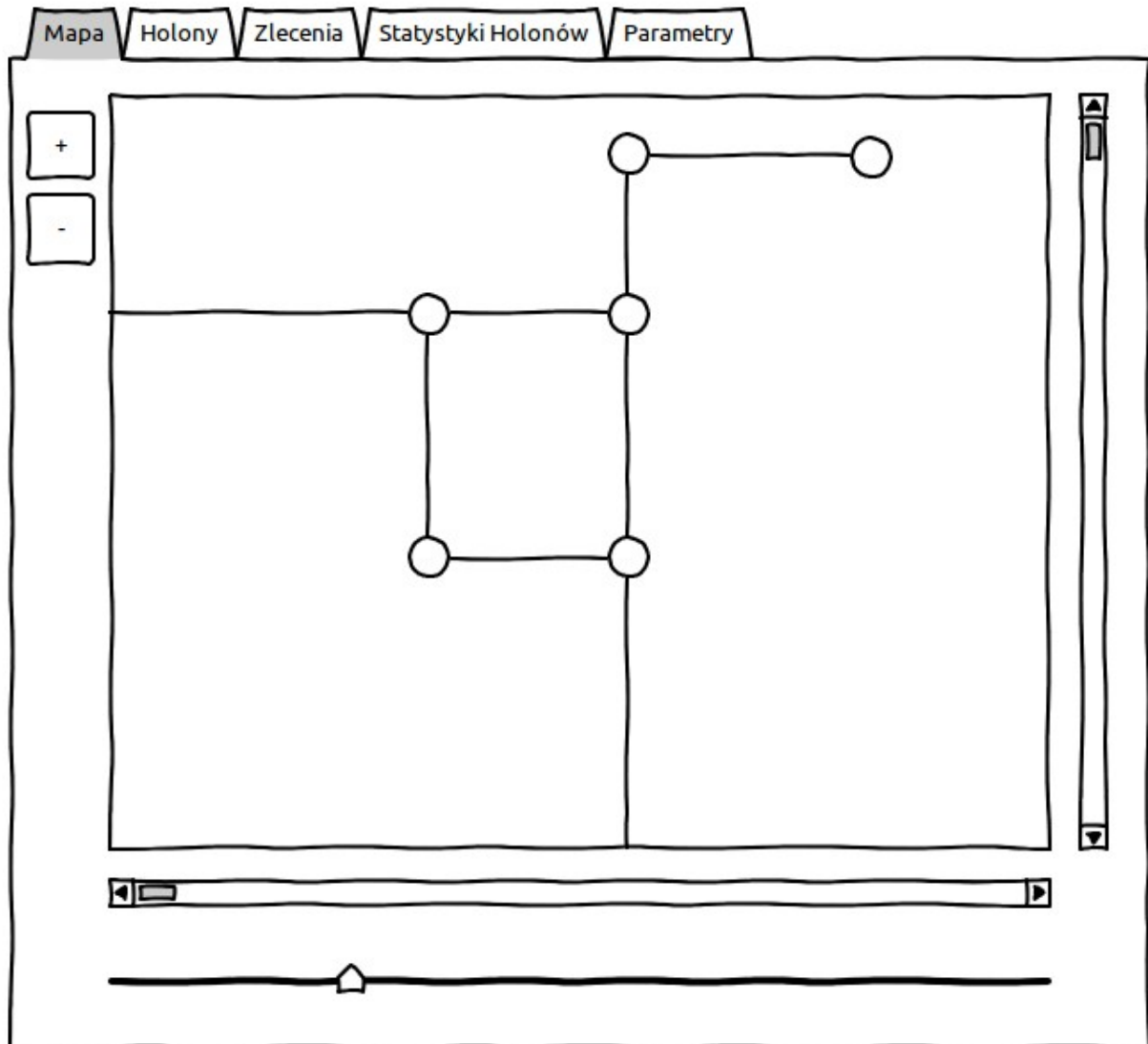
1. Zakładka „Mapa”.

Zakładka „Mapa” odpowiadać ma wizualizacji grafu. W ramach pierwotnych założeń przewidziano następujące funkcjonalności: skalowanie wizualizowanych obiektów przy pomocy odpowiednich przycisków oraz scrollem myszki, przesuwanie aktualnego widoku przy pomocy pasków przewijania. W wersji finalnej cała nawigacja odbywa się za pomocą myszki, paski przewijania zastąpione zostały typową dla tego typu zastosowań metodą przeciągania, a przybliżanie możliwe jest jedynie za pomocą scrolla myszki.



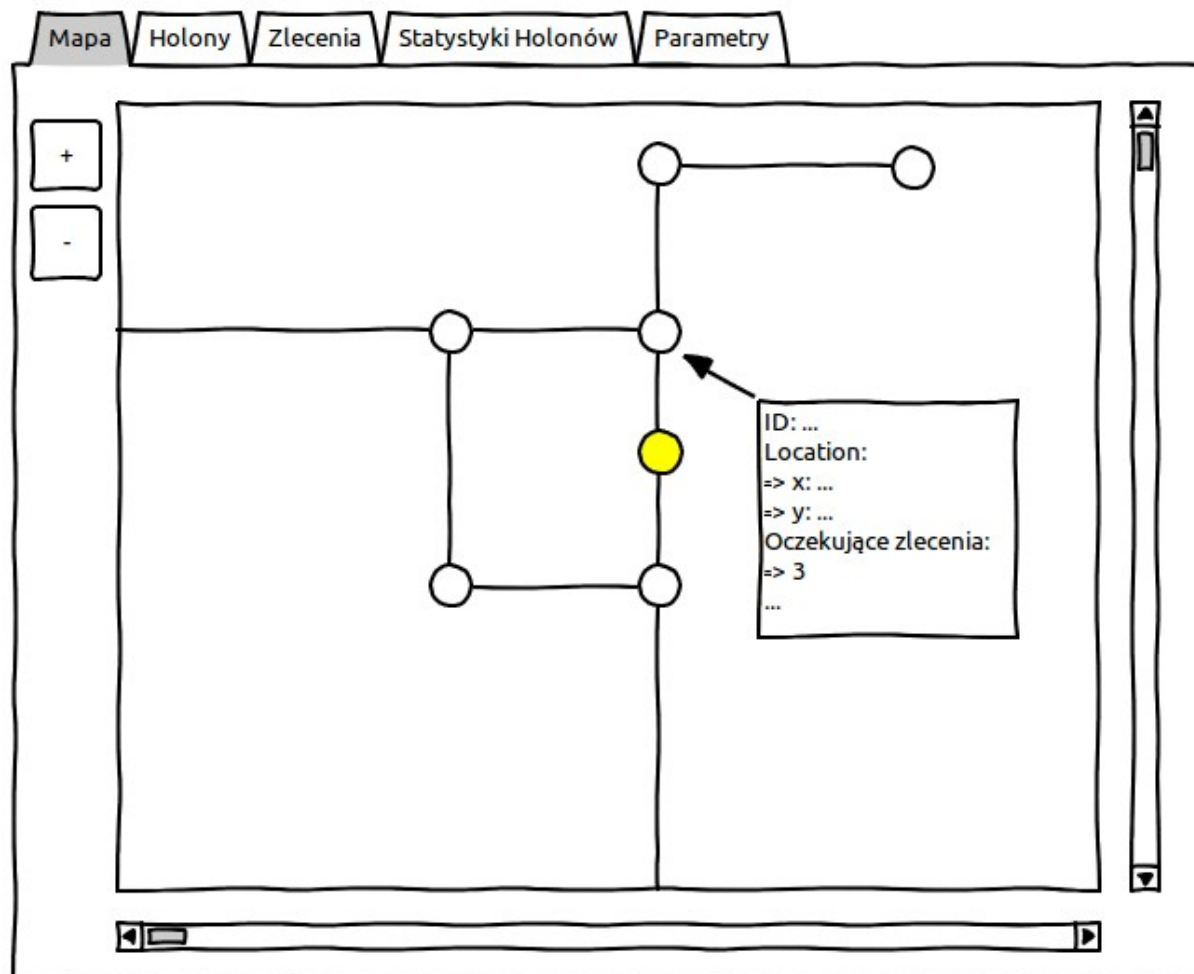
Rysunek 1: Zakładka "Mapa"

Pierwotnie planowaliśmy, aby po zakończeniu symulacji na dole okna pojawiał się kolejny pasek przewijania, pozwalający na przemieszczanie się w czasie i wybranie kroku symulacji, którego wizualizacja ma się ukazać. Alternatywną koncepcją jaką uwzględniliśmy w projekcie gui, a ostatecznie zastosowaliśmy w aplikacji, jest by pasek ten był pokazany od początku i zawierał wszystkie kroki dotychczas wygenerowane. W ten sposób uzyskujemy możliwość skupienia się i dokładnego przeanalizowania danego momentu czasowego symulacji już w trakcie jej trwania.



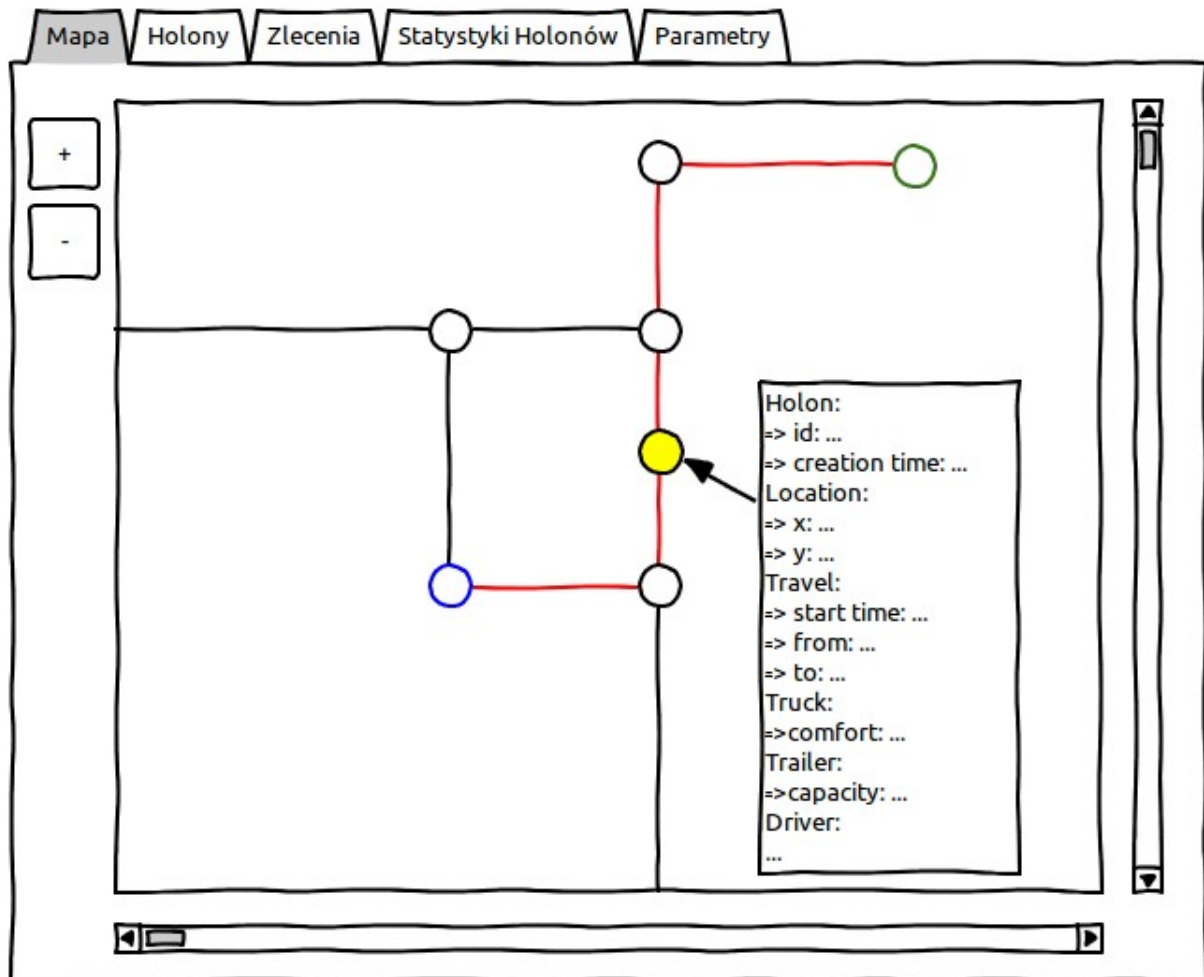
Rysunek 2: Zakładka "Mapa" ze skalą kroków

Na węzły wyświetlanego grafu, można najechać myszką, co spowoduje wyświetlenie informacji na temat danego węzła, takich jak jego położenie czy ilość oczekujących zleceń. Kółka pokolorowane na żółto (na potrzeby projektu GUI) reprezentują przemieszczające się holony.



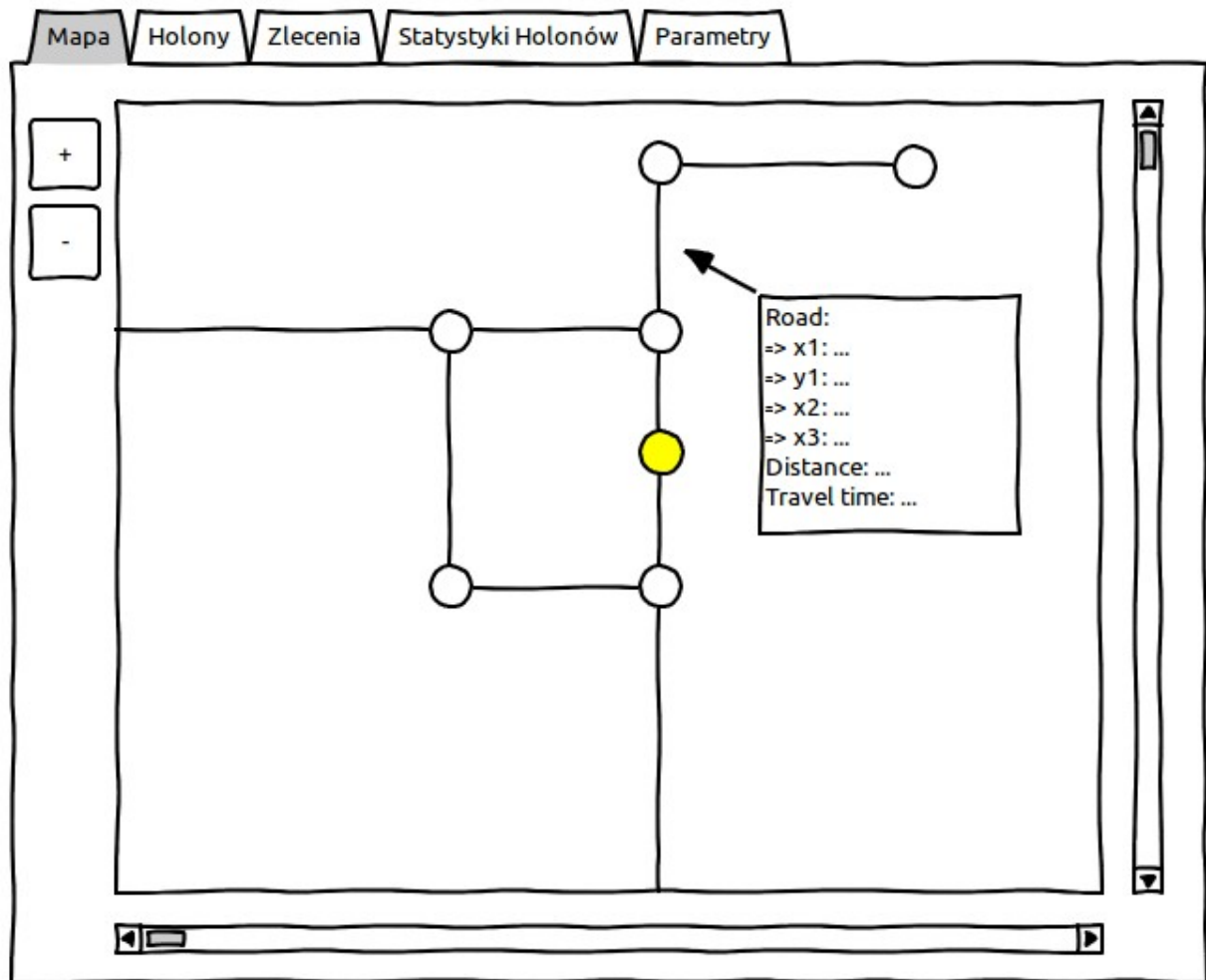
Rysunek 3: Zakładka "Mapa" przy najechaniu kursorem myszki na węzeł grafu

Po najechaniu myszką na holon, ukazuje się szereg informacji dotyczących m. in. położenia holonu, jego aktualnej podróży czy składu. Na czerwono kolorowana jest trasa jaką podąża, a węzły startowy i końcowy wyróżnione odpowiednio kolorem zielonym i niebieskim. Kliknięcie w pojazd powoduje przeniesienie do zakładki „Holony”, gdzie te same informacje wyświetlone są w formie tabeli, w zestawieniu z innymi holonami. Wiersz dotyczący klikniętego holonu będzie podświetlony. W finalnej wersji projektu każdemu holonowi przypisany jest osobny kolor, a więc nie było potrzeby wyświetlania tras dopiero po najechaniu myszką, wszystkie domyślnie widoczne są na mapie.



Rysunek 4: Zakładka "Mapa" przy najechaniu kursorem myszki na holon

Po najechaniu kursorem myszki na trasę między pewnymi dwoma węzłami oczom użytkownika ukazać się podstawowe informacje o tej drodze takie jak jej długość, położenie czy czas potrzebny na jej pokonanie.



Rysunek 5: Zakładka "Mapa" przy najechaniu kursorem myszki na drogę

2. Zakładka „Holony”

Zakładka ta pozwala na uzyskanie szczegółowych informacji na temat poszczególnych holonów, umożliwiając jednocześnie porównywanie ich poszczególnych parametrów. Po kliknięciu na pewien holon i przeniesieniu się do tej zakładki, dany holon będzie wyróżniony np. poprzez zaznaczenie.

Mapa

Holony

Zlecenia

Statystyki Holonów

Parametry

ID	x	y	travelling from	travelling to	Truck comfort	Trailer Capacity	Driver
2	3	7	1	5	110	230	1
1	7	40	5	10	190	420	2
7	12	5	4	2	200	510	4
9	2	8	4	12	245	515	8
...							

Rysunek 6: Zakładka "Holony"

3. Zakładka „Zlecenia”

Zakładka ta zgodnie z sugestią prowadzących zawiera informacje o wszystkich zleceniach, zarówno tych już zrealizowanych, bieżących, jak i pozostających do wykonania. Dostępne są tu informacje na temat czasów realizacji, węzłach początkowych i końcowych czy oknach czasowych. Tabelę można oczywiście poszerzyć o kolejne parametry.

Mapa

Holony

Zlecenia

Statystyki Holonów

Parametry

Klient	Holon	Pobrano o	Dostarczono o	Z węzła	Do węzła	Start okna	Koniec okna	...
2	2	3	7	1	5	110	230	...
1	3	7	40	5	10	190	420	...
7	7	12	-	4	2	200	510	...
9	2	-	-	4	12	245	515	...

Rysunek 7: Zakładka "Zlecenia"

4. Zakładka „Statystyki Holonów”

Zakładka ta zawiera tabelę z wylistowanymi wszystkimi holonami, co pozwala w czytelny sposób podejrzeć i porównać ich statystyki. W odróżnieniu od zakładki „Holony” nie znajdziemy tu informacji o bieżącej sytuacji holonów (jak położenie czy aktualnie realizowane zlecenie), a statystyki dotyczące całego ich dotychczasowego istnienia.

Mapa

Holony

Zlecenia

Statystyki Holonów

Parametry

Holon	Creation Time	Average Load	Average Distance ToBase	Symmary Latency	Max Latency	Wait Time	...
1	10	30	25.342	423.1	57	21	...
...							

Rysunek 8: Zakładka "Statystyki Holonów"

5. Zakładka „Parametry”

Zakładka „Parametry” pozwala podejrzeć parametry algorytmu. Idąc za sugestią prowadzących zmieniliśmy jej format na tabelę, w której to przedstawiona będzie historia zmian parametrów, a nie jedynie aktualna konfiguracja. Parametry mogą, ale nie muszą ulegać zmianie w czasie trwania symulacji.

Mapa

Holony

Zlecenia

Statystyki Holonów

Parametry

Aktualne od	Aktualne do	Algorytm	STDDepth	Time	...
0	50	Simmulated Trading	...	false	...
51	120	Simmulated Trading	...	true	...
...					

Rysunek 9: Zakładka "Parametry"

6. Sposób wydobywania informacji z systemu

1. Sposób pierwotnie zakładany

W celu stworzenia tabel prezentujących aktualne dane symulacyjne, a także do tworzenia grafu „na żywo” potrzebna jest ingerencja w odziedziczony system. Zasadniczej rozbudowie zostanie poddana klasa `xml.elements.SimulationData`. Dane do klasy danych symulacji wprowadzane są z poziomu metody `sendSimulationData` klasy `ntp.jade.eunit.ExecutionUnitAgent`. Dostęp do części danych, takich jak już pobierane dane o kierowcach lub ciężarówkach jest z tego miejsca trywialny.

Większe przeszkody napotykamy, gdy chcemy pobrać dane o miękkich oknach czy niepewnych czasach przejazdów, opisane w rozdziale 15.4.3 pozycji [6] z bibliografii. Głównym problemem jest bardzo uboga dokumentacja samego kodu, która sprawia że poszukując odpowiednich metod i klas błądzimy nieco po omacku, opierając się na podejrzeniach i intuicji. Wydaje nam się, że poszukiwane informacje zawarte są w klasach pakietu `measure`, zaś do ich wyciągnięcia posłużyć może klasa `MeasureHelper`. Jej statyczne metody wymagają jednak podania dwóch parametrów będących listami obiektów klasy `Commision`. Tutaj z kolei, wydaje nam się że parametry te pobrać można z obiektu klasy `Schedule`, do którego dostęp posiadamy już z wnętrza klasy `ExecutionUnitAgent`.

Jak widać, bardzo łatwo jednak o pomyłkę w naszych domysłach. Uważamy więc za konieczne spotkanie z twórcą lub osobą wysoce zaznajomioną z kodem odpowiedzialnym za udostępnianie wspomnianych wyżej parametrów.

Jeśli chodzi o drugą część naszego zadania, zgodnie z sugestią prowadzących planujemy aktualizować nasz graf symulacji z metody `ntp.jade.gui.GUIAgent.addSimulationData`, która towołana jest przy każdym kroku czasowym, a której parametry powinny wystarczyć do stworzenia wizualizacji.

2. Wnioski po spotkaniu w sprawie projektu

Konsultant odradzał podpinanie GUI pod metodę `sendSimulationData` klasy `ExecutionUnitAgent`. Polecał zaprzęcenie do klasy `GUIAgent`. Komunikacja tegoż agenta z holonami następuje poprzez zachowanie `GetSimulationDataBehaviour`, które to filtruje nadchodzące wiadomości względem wartości `CommunicationHelper.SIMMULATION_DATA`. Najlepszym miejscem na aktualizowanie GUI odbieranymi danymi symulacyjnymi byłaby metoda `GUIAgent.addSimulationData`. Przesyłane dane, czyli klasę `SimulationData`, należałoby uzupełnić o obiekt klasy `Schedule` i zadbać o jego wypełnienie w metodzie `ExecutionUnitAgent.sendSimulationData`. Na podobnej zasadzie należałoby się podpiąć pod metodę `GUIAgent.ChangeGraph`, co pozwoli zaktualizować wyświetlany graf (przy niepewnych czasach przejazdu).

Klasa `Schedule` zawiera wyliczone po danym kroku zlecenia dla Holona. Trasę zlecenia można uzyskać poprzez uzyskanie ze zlecenia obiektów `GraphPoint` reprezentujących miejsca załadunku i rozładunku. Aby uzyskać pełny obraz, należy użyć metody `graph.GetLink` (lub `GetPath`) na tych punktach, co zwróci pełną trasę.

Wydobycie dodatkowych parametrów, które powinniśmy wyświetlać, okazało się nie tak proste, jak byśmy sobie tego życzyli. Parametry z pakietu `measure` są dostępne w obiekcie `measureData` Dystrybutora, podobnie jak parametry algorytmu, których zmiany powinniśmy śledzić poprzez metodę `changeConfiguration` Dystrybutora. Są przynajmniej dwie drogi, obie dość złożone przy naszym stanie wiedzy. Pierwsza metoda polega na asynchronicznej komunikacji między agentami Dystrybutora, a `TestAgentem`. Należy utworzyć nowy typ komunikatu w klasie `CommunicationHelper`, następnie nową klasę `Behaviour` filtrującą po tym typie, zadbać o zarejestrowanie zachowania i wywołanie zapytań o dane (ze zwróceniem uwagi na metodę `setConstantObject`, która powinna dostawać pustego `Stringa`), dodać Dystrybutorowi funkcjonalność obsługującą to zachowanie (drobne ułatwienie - otrzymywana wiadomość `msg` ma metodę `GetSender`, na którą można łatwo odesłać dane). Druga metoda, zalecana nam, to blokujące zapytania synchroniczne, o których dokładne informacje dostaniemy poprzez bezpośredni emailowy kontakt z konsultantem. Ostatecznie dane dotyczące parametrów algorytmu wyciągamy z systemu bezpośrednio z klasy `DistributorAgent`, co opisaliśmy w punkcie 8.1 niniejszego dokumentu.

Ponieważ objętość projektu i rozproszenie danych po różnych klasach są dość znaczne, przypuszczamy, że pewne szczegóły i trudności będą się jeszcze wyłaniać w trakcie naszej pracy.

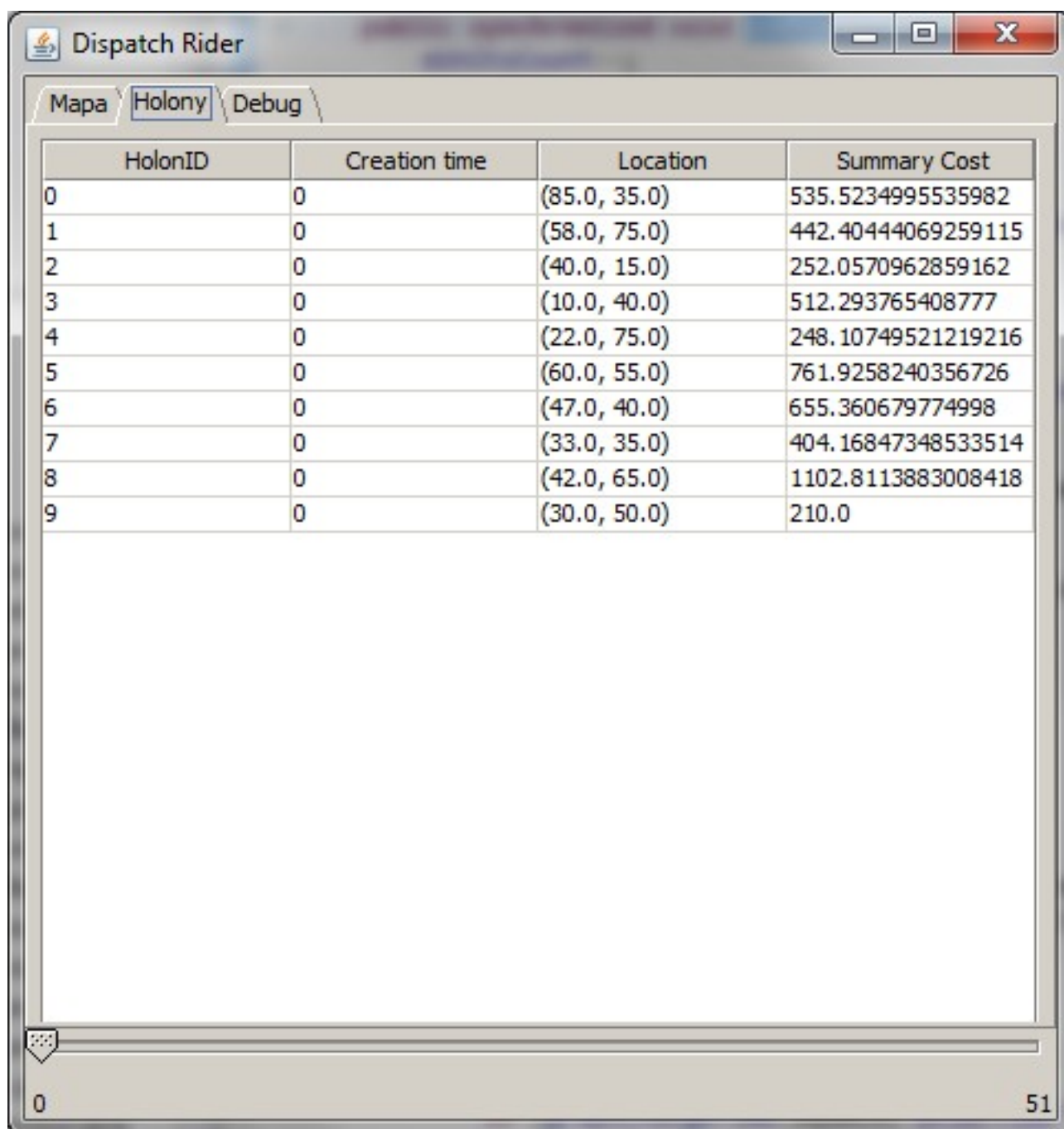
7. Prace nad prototypem

1. Wydobywanie parametrów

W ramach pracy nad projektem i integracją z Dispatch Riderem, mogliśmy skonfrontować zebrane przez nas informacje na temat uzyskiwania danych z rzeczywistością. W trakcie pracy napotkaliśmy na kilka trudności, które spowolniły naszą pracę. W dalszym ciągu nie mamy jasności co do sposobu wykorzystania wywołań blokujących, niezbędnych do uzyskania parametrów algorytmu, czy danych z klas MeasureData, jednakże komunikujemy się z Panem Sebastianem Pisarskim w celu pokonania tej przeszkody. Bez większych problemów udało nam się uzyskać dane na temat holonów i grafu podawanego w pliku konfiguracyjnym, jednakże grafy te są zwykle bardzo gęste, więc konieczne będzie odchudzenie wyświetlanych danych. Nie mamy pewności co do wyświetlania wierzchołków grafu euklidesowego – przypuszczamy, że cały ich zbiór powinniśmy uzyskać z punktów startowych i końcowych zleceń.

2. Prace nad GUI

Poniższy screenshot przedstawia wygląd zakładki holonów, z przykładowymi danymi do wyświetlenia. Dodanie kolejnych kolumn nie powinno stanowić najmniejszych problemów, dlatego póki co skoncentrowaliśmy się na innych, pilniejszych zadaniach. Nie należy sugerować się obecnością zakładki „Debug”, ma nam jedynie pomóc podczas prac nad projektem.

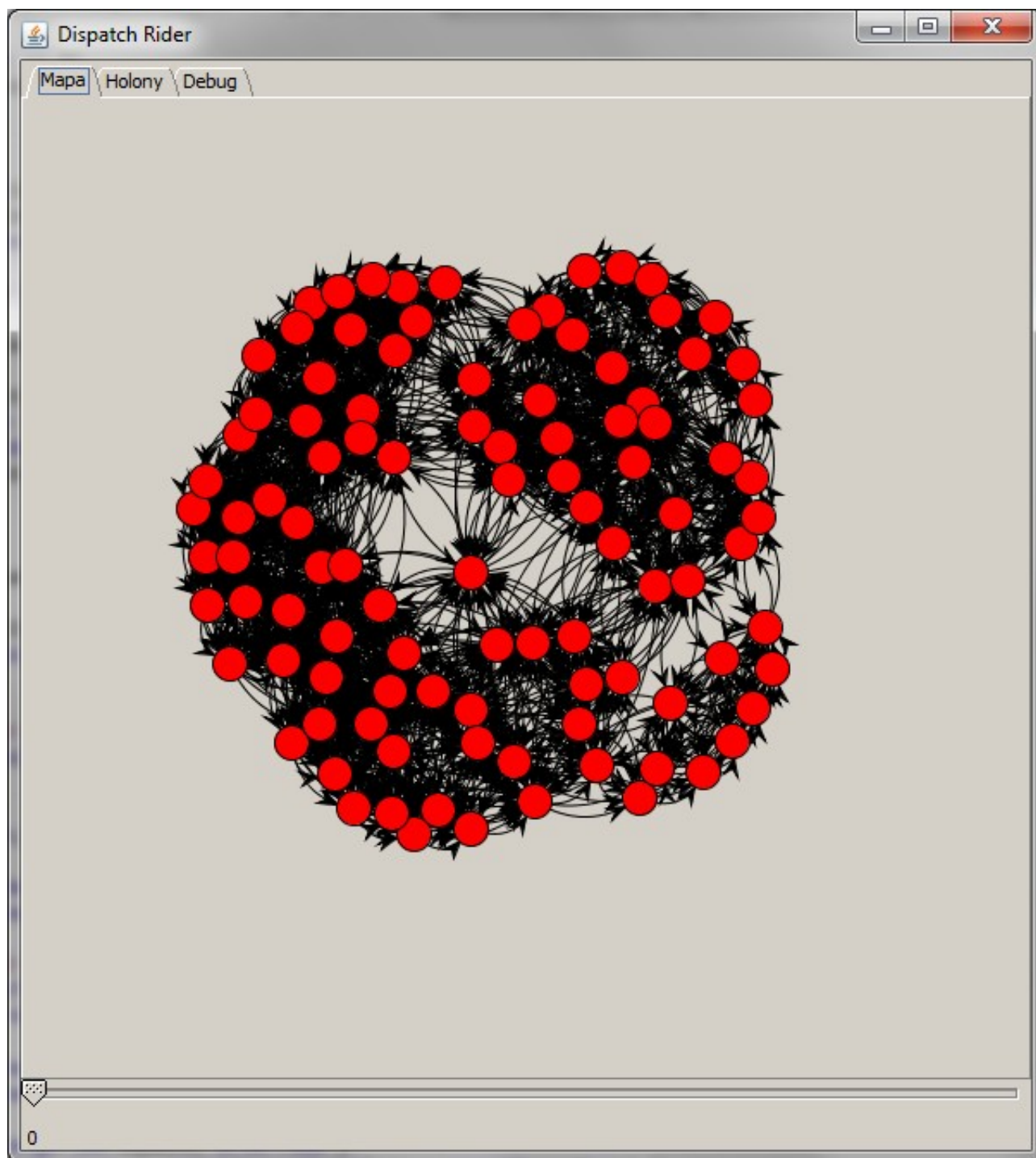


The screenshot shows a window titled "Dispatch Rider" with three tabs: "Mapa", "Holony", and "Debug". The "Holony" tab is selected, displaying a table with four columns: "HolonID", "Creation time", "Location", and "Summary Cost". The table contains 10 rows of data, indexed 0 to 9. Below the table is a large empty rectangular area, and at the bottom left is a small icon of a triangle with dots. The bottom right corner of the window displays the number "51".

HolonID	Creation time	Location	Summary Cost
0	0	(85.0, 35.0)	535.5234995535982
1	0	(58.0, 75.0)	442.40444069259115
2	0	(40.0, 15.0)	252.0570962859162
3	0	(10.0, 40.0)	512.293765408777
4	0	(22.0, 75.0)	248.10749521219216
5	0	(60.0, 55.0)	761.9258240356726
6	0	(47.0, 40.0)	655.360679774998
7	0	(33.0, 35.0)	404.16847348533514
8	0	(42.0, 65.0)	1102.8113883008418
9	0	(30.0, 50.0)	210.0

Poniższy screenshot pokazuje wstępny wygląd grafu, który oczywiście będzie wymagał dalszej obróbki z naszej strony. Póki co wyświetlamy tylko graf przekazywany w pliku

konfiguracyjnym. W najbliższym czasie planujemy wprowadzić możliwość nawigowania po nim (przybliżania, przesuwania), jak również stopniowo wzbogacać go o ustalone podczas poprzednich spotkań elementy.



Rysunek 11: Prototyp zakładki mapy

3. Wnioski i dalszy przebieg prac

Prace nad prototypem pozwoliły nam na dokładniejsze zapoznanie się z systemem. Niestety, mimo spotkania konsultacyjnego, często w trakcie prac pojawiają się wątpliwości co do uzyskiwania danych jak i działania kodu Dispatch Ridera, których rozwikłanie zajmuje istotną część pracy nad GUI.

Ogromnym problemem jest również niezrozumiałe, różne zachowywanie się systemu w zależności od maszyny na której zostaje uruchomiony. Niestety, największe problemy napotykamy na jedynym laptopie do którego mamy dostęp, na którym możemy pokazywać efekty naszych prac. Bardzo problematyczne jest również tworzenie i testowanie kodu w przypadku, gdy jedno uruchomienie programu, dla najmniejszego możliwego do wygenerowania przypadku testowego, trwa około godziny, jak to ma miejsce gdy graf czytany jest z pliku.

Problemy te udało nam się na szczęście rozwiązać w trakcie prac nad projektem, w większości spowodowane były błędną konfiguracją systemu.

8. Dokumentacja deweloperska

1. Zmiany w dostarczonym kodzie

Nasz moduł, jako graficzny interfejs dołączony do już istniejącej, dużej aplikacji wymagał oczywiście podpięcia w pewien, pośredni bądź nie, sposób do modułu obliczeniowego. W tym celu, aby uniknąć problemów z komunikacją, zarzuciliśmy początkowo rozważaną koncepcję utworzenia nowego agenta JADE, a wykorzystaliśmy już istniejącą klasę GUIAgent, z której to wywołujemy metody aktualizujące utworzony przez nas interfejs. Opakowany jest on w prostą klasę SingletonGUI, dzięki której nasz moduł widoczny jest w prosty i spójny sposób z całego systemu. Punkty w których wspomniane wcześniej aktualizacje są wywoływane to (można je łatwo poznać po charakterystycznym "SingletonGUI.getInstance(...)"):

- 1) metoda simulationStart, wywoływana tylko raz, w której to dodaliśmy jedną linię przesyłającą do naszego modułu obiekt klasy SimInfo, zawierający ogólne dane nieulegające zmianie w trakcie symulacji.
- 2) metoda sendTimestamp, wywoływana z każdym nowym krokiem czasowym. Dodaliśmy w niej linię, która odpowiada za poinformowanie modułu GUI o przejściu do kolejnego kroku symulacji (wizualizacji)
- 3) metoda addSimulationData, wywoływana w każdym kroku czasowym dla każdego holonu. W niej umieściliśmy linię przesyłającą całą gamę danych dotyczącą danego holonu w danym kroku czasowym do naszego modułu. Odbywa się to przy użyciu obiektu klasy SimulationData

Takie podejście pozwoliło nam na uzyskanie niemal wszystkich koniecznych do przedstawienia w interfejsie użytkownika danych. Problemy wyniknęły jedynie w przypadku danych dotyczących parametrów algorytmu, które to nie są związane bezpośrednio z symulacją, a co za tym idzie nie są dostępne z poziomu klasy GUIAgent. Dane te, po dłuższych konsultacjach z osobą najlepiej zorientowaną w systemie DispatchRider, wyłuskujemy ostatecznie z klasy DistributorAgent, z poziomu publicznej metody setCommissions. W celu przesłania koniecznych danych do naszego modułu utworzyliśmy klasę DRParams, której to pola są we wspomnianej metodzie wypełniane, a następnie obiekt klasy DRParams jest przesyłany do naszego modułu przy pomocy pojedynczego wywołania.

2. Znaczenie utworzonych pakietów i klas

1. Pakiet gui.main

Jest to pakiet w którym znajdują się 2 klasy. Są to prosta SingletonGUI implementująca jedynie wzorzec singleton i dziedzicząca po drugiej klasie, WindowGUI, które to jest jedynym punktem dostępu do naszego modułu z zewnątrz, a zarazem podstawową klasą interfejsu użytkownika.

W klasie WindowGUI zdefiniowane są wszystkie metody wspomniane w “Zmianach w dostarczonym kodzie”, służące do pobierania danych z właściwego systemu DispatchRider. Metody te informują z kolei klasy odpowiadające za poszczególne panele (zakładki) o zachodzących zmianach.

Jeśli chodzi o elementy widoczne dla użytkownika, cała nasza aplikacja, a co za tym idzie klasa WindowGUI napisana jest z użyciem swinga. WindowGUI jako klasa spinająca cały interfejs odpowiada za poprawne wyświetlenie zakładek, odpowiednie przeskalowanie ich zawartości, a także za widoczny we wszystkich zakładkach suwak (oś czasu) i poprawne jego zachowanie, spięcie z wyświetlanymi danymi.

2. Pakiet gui.common

Pakiet ten służy uspoźnieniu struktury klas i metod odpowiadających za wyświetlanie danych w poszczególnych zakładkach. Zawiera klasę TimestampRecord mającą w założeniu przechowywać dowolny zestaw danych potrzebny w danej zakładce, ale danych powiązanych z danym momentem czasowym (np. aktualna pozycja holonów w danym czasie w przypadku mapy). Klasa ta ma utworzony swój Comparator (TimestampRecordComparator), konieczny do poprawnego zachowania suwaka czasu. Pozostałe 2 klasy znajdujące się w tym pakiecie to interfejs Updateable, który implementują klasy odpowiadające za poszczególne zakładki, a także abstrakcyjną klasę TimestampUpdateable, w której przechowywane są implementacje tych metod, których wykonanie powinno być takie samo dla wszystkich zakładek (w celu uniknięcia kopiowania tego samego kodu w kilku miejscach).

3. Pakiet gui.map

Pakiet ten ze względu na odmienną treść wyświetlaną na ekran odbiega znacznie strukturą od pozostałych pakietów odpowiadających za dane w zakładkach. Podstawową klasą notyfikowaną o wszelkich zdarzeniach jest klasa MapHolder. Wywołanie na niej metody repaint() (odbywa się to z poziomu WindowGUI) powoduje odświeżenie zawartości wyświetlanej mapy. Klasa ta posiada również standardowy zestaw metod do pobierania danych do symulacji. Pozostałe metody, w głównej mierze prywatne, odpowiadają za wyświetlanie poszczególnych elementów. Ich nazwy są bardzo intuicyjne, a zadania wyraźnie rozdzielone. Są to wstawienie wierzchołków grafów, holonów, krawędzi, ustawienie wszelkich właściwości takich jak grubość linii czy kształt strzałek itd. Pozostałe klasy z tego pakietu to HolonGraphLink i HolonGraphPoint służące do rozróżnienia przez nas krawędzi i wierzchołków “standardowych” od tych związanych z przebytą trasą czy holonem. W pakiecie tym znajdziemy również klasę InvisibleGraphPoint ułatwiającą nam ładne rysowanie przebytej przez holon trasy, a także klasę ColorCreator, której jedynym zadaniem jest stworzenie możliwie różnych kolorów dla wszystkich holonów.

4. Pakiety gui.commissions, gui.holon, gui.holonstats, gui.parameters

Wszystkie te pakiety są bardzo zbliżone do siebie strukturą i rolą. Każda z zakładek za które

odpowiadają wyświetla pojedynczą tabelę. Poszczególne pakiety posiadają własną klasę *Updateable rozszerzającą TimestampUpdateable, w której to zaimplementowana jest metoda update(SimulationData). Rolą tych klas jest dopasowanie przechowywanych danych do tego, co potrzeba w tabeli wyświetlić. Klasy *TableModel odpowiadają z kolei już za same tabele i odpowiednie zgranie ich z suwakiem czasowym.

9. Podręcznik użytkownika

1. Przed uruchomieniem.

Uruchomienie naszego projektu jest analogiczne do uruchomienia dostarczonego nam rozwiązania Dispatch Ridera, toteż wymaga podobnych przygotowań w warstwie konfiguracji. Najważniejszym elementem jest plik z konfiguracją używany przez system agentowy, przy czym niezwykle istotne jest ustawienie parametrów *commissions* na *recording="true" dynamic="true"*. W przypadku decyzji o korzystaniu ze zdefiniowanego grafu, również konieczne jest ustawienie tego w pliku konfiguracyjnym, w postaci `<roadGraph trackFinder="rodzaj_track_fintera"> plik_z_grafem.xml </roadGraph>`. Przykładem poprawnej konfiguracji jest:

```
<?xml version="1.0" encoding="UTF-8"?>
<tests xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="xml/schemes/configuration.xsd">
  <test>
    <commissions recording="true" dynamic="true" packageSending="false" choosingByCost="
    <defaultAgents>
      <truck power="600" reliability="4" comfort="4" fuelConsumption="20"/>
      <trailer mass="200" capacity="200" cargoType="1" universality="4"/>
    </defaultAgents>
    <configuration>tmp_tests/25_200</configuration>
    <results>tmp_tests/1c101</results>
    <roadGraph trackFinder="Dijkstra">graph.xml</roadGraph>
  </test>
</tests>
```

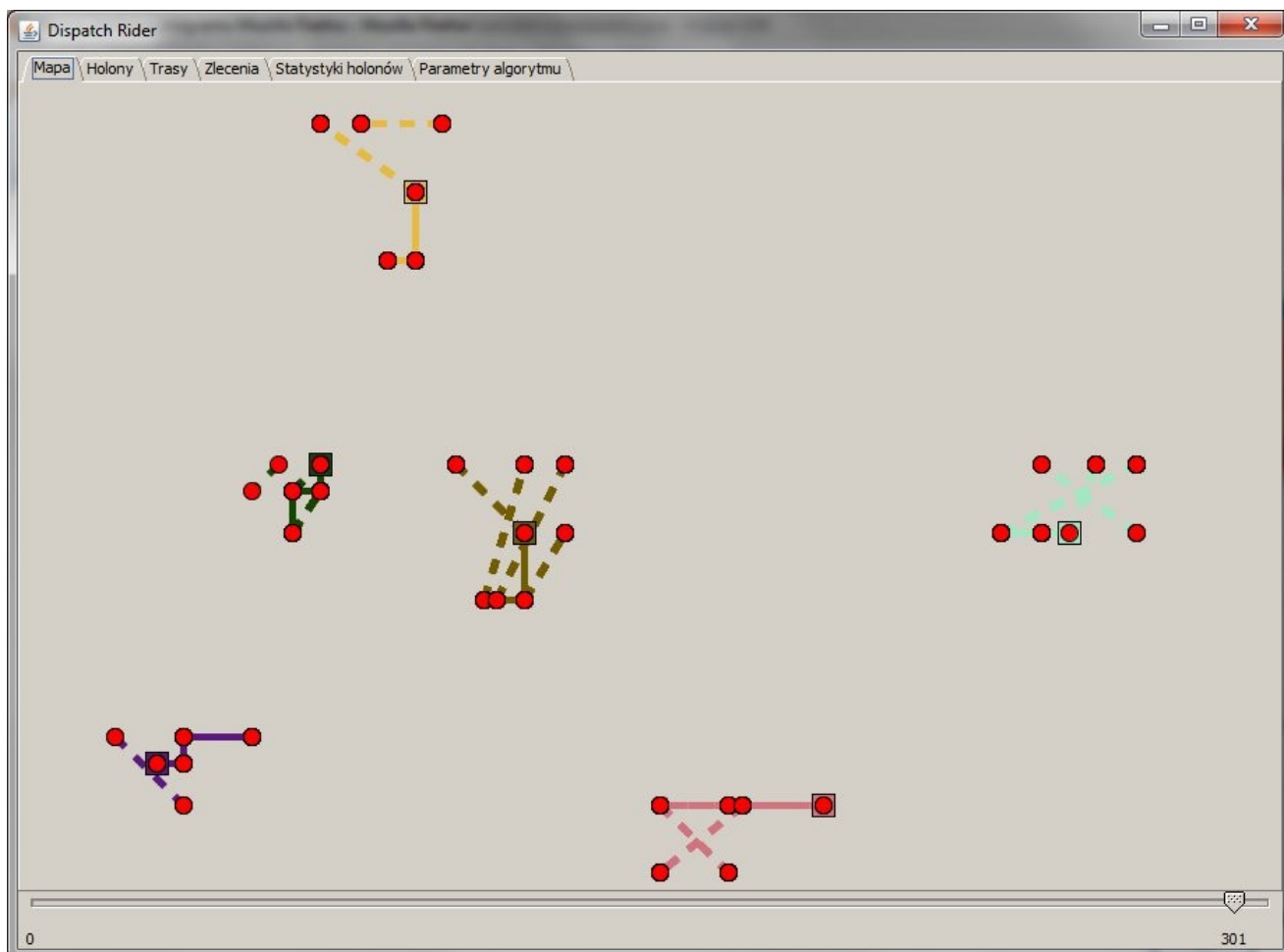
Rysunek 12: Przykładowa konfiguracja

2. Działanie aplikacji.

Nasz program pozwala w czasie zbliżonym do rzeczywistego obserwować efekty pracy agentów Dispatch Ridera. Po dojściu do odpowiedniego momentu w obliczeniach (tzw. timestamp), dane zostają udostępnione użytkownikowi. Do przemieszczania się pomiędzy różnymi momentami czasowymi służy suwak znajdujący się u dołu okna.

1. Suwak czasu.

Suwak został zaprojektowany tak, aby umożliwiać wygodne i intuicyjne obserwowanie interesujących danych. Gdy jest ustawiony na ostatni dostępny timestamp automatycznie przeskakuje na kolejny, gdy ten zostanie przesłany. W innym przypadku suwak pozostanie na swoim miejscu do czasu interakcji użytkownika. Zapobiega to “przeskakiwaniu” danych w nieoczekiwanym momencie.



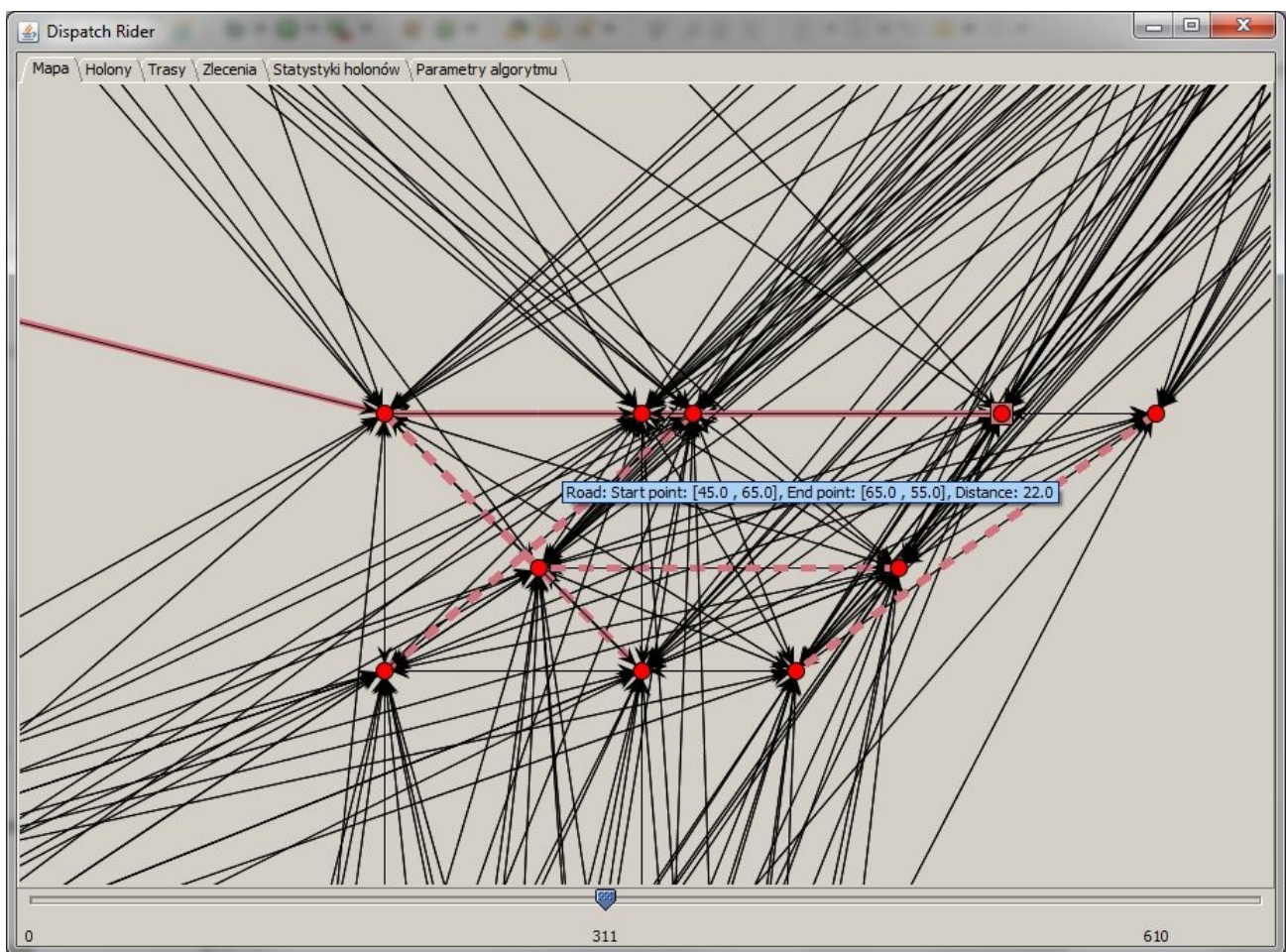
Rysunek 13: Zakładka mapy dla konfiguracji bez grafu

2. Zakładki.

Do przemieszczania się pomiędzy wizualizowanymi danymi służą zakładki. Zakładki pozwalają na szybkie dotarcie do interesujących nas danych, poprzez tematyczny ich podział.

1. Mapa.

Mapa obrazuje rozmieszczenie w przestrzeni kluczowych punktów - holonów, punktów końcowych zleceń. Wskazanie myszką konkretnego elementu prowadzi do wyświetlenia istotnych elementów w postaci tzw. tooltipa. Wyświetlanie jest w pełni konfigurowalne przy pomocy myszki. Za pomocą scrolla można widok przybliżać i oddalać, przy pomocy lewego przycisku można go przesuwać. Linia ciągła reprezentuje zlecenia wykonywane do danego momentu czasowego. Linia przerywana reprezentuje zlecenia planowane w danym momencie czasowym.



Rysunek 14: Zakładka mapy dla konfiguracji z grafem

W skład tooltipa holona wchodzi:

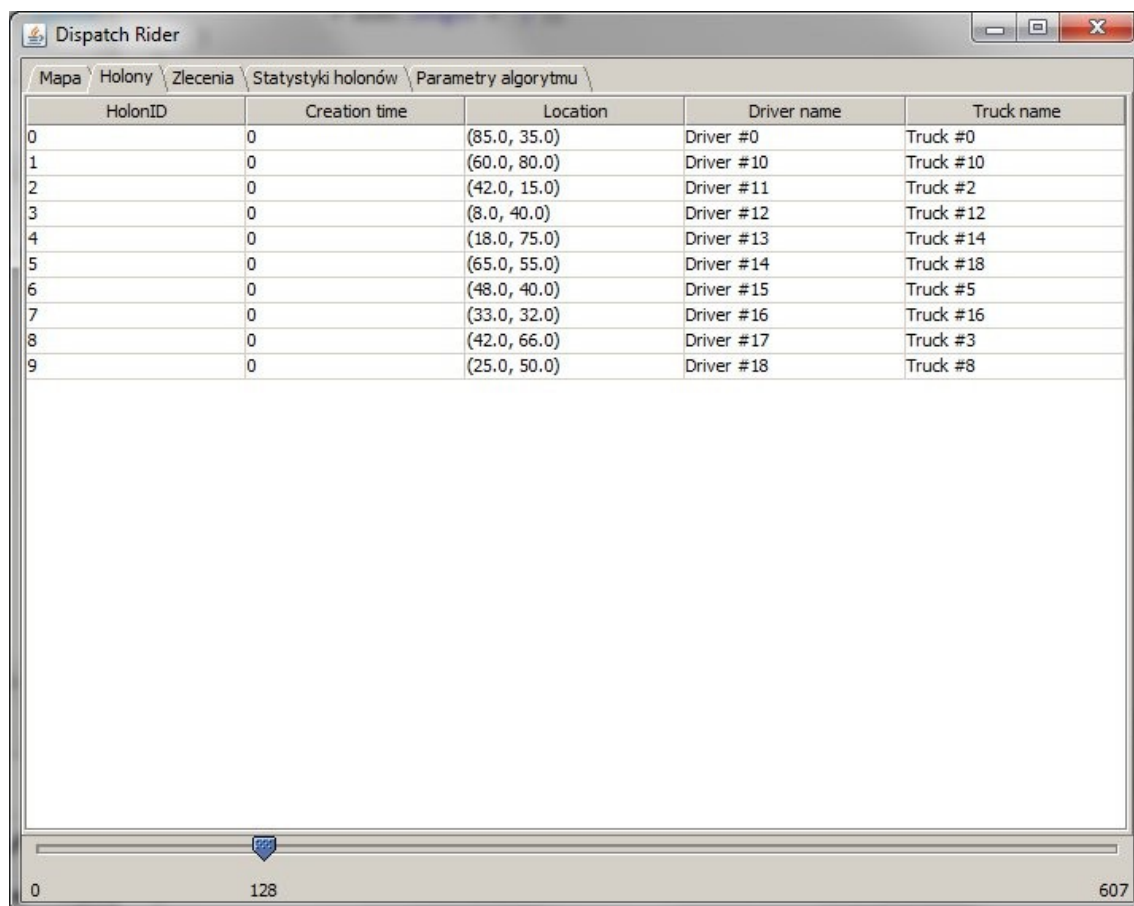
Tabela 1: Składowe tooltipa holona

ID	ID Holona
Creation time	Czas utworzenia
Coordinates	Pozycja na mapie
Truck comfort	Komfort pojazdu
Trailer capacity	Pojemność przyczepy
Driver	Kierowca

W skład tooltipa krawędzi wchodzi punkt początkowy, końcowy, oraz dystans.

2. Tabele.

Pozostałe zakładki wyświetlają parametry wyliczone przez system dla poszczególnych holonów i zleceń, w formie tabel. Dane można łatwo segregować, sortując je po wartościach w kolumnie, bądź modyfikując kolejność kolumn.



HolonID	Creation time	Location	Driver name	Truck name
0	0	(85.0, 35.0)	Driver #0	Truck #0
1	0	(60.0, 80.0)	Driver #10	Truck #10
2	0	(42.0, 15.0)	Driver #11	Truck #2
3	0	(8.0, 40.0)	Driver #12	Truck #12
4	0	(18.0, 75.0)	Driver #13	Truck #14
5	0	(65.0, 55.0)	Driver #14	Truck #18
6	0	(48.0, 40.0)	Driver #15	Truck #5
7	0	(33.0, 32.0)	Driver #16	Truck #16
8	0	(42.0, 66.0)	Driver #17	Truck #3
9	0	(25.0, 50.0)	Driver #18	Truck #8

Rysunek 15: Zakładka holonów

W skład tabeli holonów wchodzi:

Tabela 2: Kolumny tabeli holonów

"HolonID"	Id holona
"Creation time"	Czas utworzenia
"Location"	Pozycja na mapie
"Driver name"	Nazwa kierowcy
"Truck name"	Nazwa ciężarówki

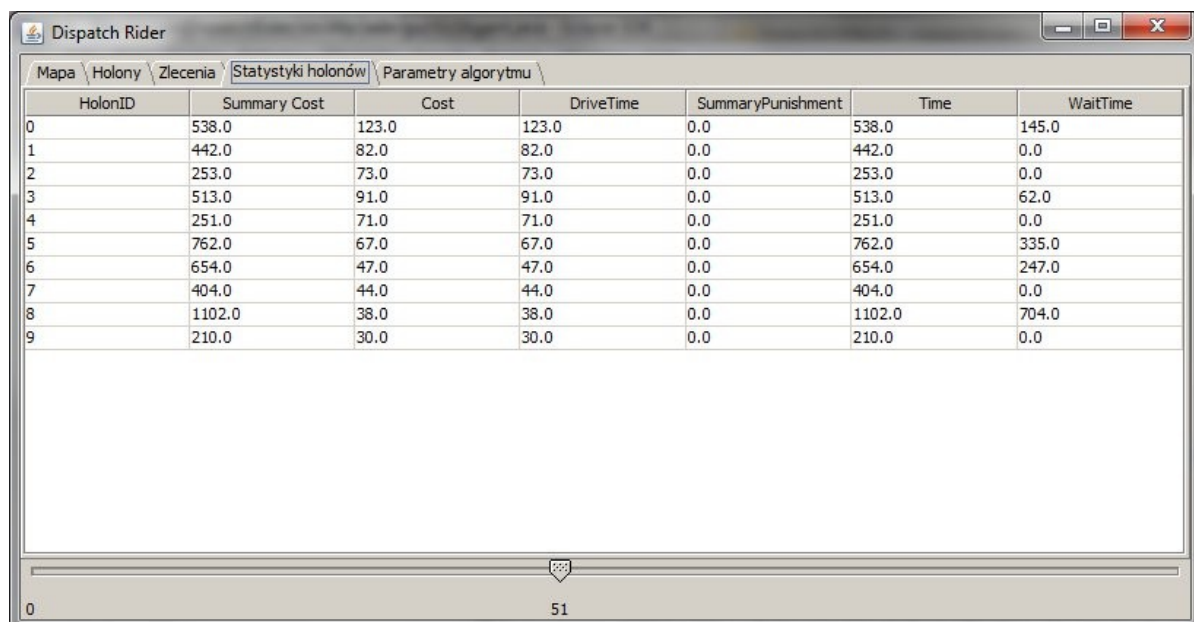
CommissionID	Holon ID	Pickup ID	Pickup X	Pickup Y	Pickup Tim...	Pickup Tim...	Delivery ID	Delivery X	Delivery Y	Delivery Ti...	Delivery Ti...
0	8	3	42.0	66.0	65.0	146.0	75	45.0	65.0	997.0	1068.0
1	8	5	42.0	65.0	15.0	67.0	7	40.0	66.0	170.0	225.0
3	8	8	38.0	68.0	255.0	324.0	10	35.0	66.0	357.0	410.0
6	4	13	22.0	75.0	30.0	92.0	17	18.0	75.0	99.0	148.0
8	4	18	15.0	75.0	179.0	254.0	12	25.0	85.0	652.0	721.0
9	4	19	15.0	80.0	278.0	345.0	15	20.0	80.0	384.0	429.0
10	9	20	30.0	50.0	10.0	73.0	24	25.0	50.0	65.0	144.0
12	9	25	25.0	52.0	169.0	224.0	27	23.0	52.0	261.0	316.0
17	3	33	8.0	40.0	87.0	158.0	37	2.0	40.0	383.0	434.0
16	3	32	10.0	40.0	31.0	100.0	31	10.0	35.0	200.0	237.0
18	3	35	5.0	35.0	283.0	344.0	39	0.0	45.0	567.0	624.0
21	7	42	33.0	32.0	68.0	149.0	40	35.0	30.0	264.0	321.0
22	7	43	33.0	35.0	16.0	80.0	41	35.0	32.0	166.0	235.0
29	2	54	42.0	10.0	186.0	257.0	60	35.0	5.0	562.0	629.0
28	2	53	44.0	5.0	286.0	347.0	58	38.0	5.0	471.0	534.0
31	2	57	40.0	15.0	35.0	87.0	55	42.0	15.0	95.0	158.0
35	6	65	48.0	40.0	76.0	129.0	72	53.0	30.0	450.0	505.0
32	6	62	50.0	35.0	262.0	317.0	68	45.0	30.0	734.0	777.0
33	6	63	50.0	40.0	171.0	218.0	74	53.0	35.0	353.0	412.0
38	0	71	95.0	35.0	293.0	360.0	77	88.0	30.0	574.0	643.0
39	0	76	90.0	35.0	203.0	260.0	73	92.0	30.0	478.0	551.0
37	6	67	47.0	40.0	12.0	77.0	61	50.0	30.0	531.0	610.0
42	0	81	85.0	35.0	47.0	124.0	70	95.0	30.0	387.0	456.0
40	0	78	88.0	35.0	109.0	170.0	104	88.0	35.0	109.0	170.0
46	5	87	65.0	55.0	85.0	144.0	83	72.0	55.0	265.0	338.0
47	5	90	60.0	55.0	20.0	84.0	88	65.0	60.0	645.0	708.0

Rysunek 16: Zakładka zleceń

W skład tabeli zleceń wchodzi:

Tabela 3: Kolumny tabeli zleceń

"CommisionID"	Id zlecenia
"Holon ID"	Id holona
"Pickup ID"	Id załadowania towaru
"Pickup X"	Współrzędna x miejsca załadowania
"Pickup Y"	Współrzędna y miejsca załadowania
"Pickup Time 1"	Czas początkowy załadowania
"Pickup Time 2"	Czas końcowy załadowania
"PickUpServiceTime"	Czas obsługi załadowania
"Delivery ID"	Id wyładowania
"Delivery X"	Współrzędna x miejsca wyładowania
"Delivery Y"	Współrzędna y miejsca wyładowania
"Delivery Time 1"	Czas początkowy wyładowania
"Delivery Time 2"	Czas końcowy wyładowania
"PickUpDeliveryTime"	Czas obsługi wyładowania
"Load"	Załadunek
"ActualLoad"	Rzeczywisty załadunek



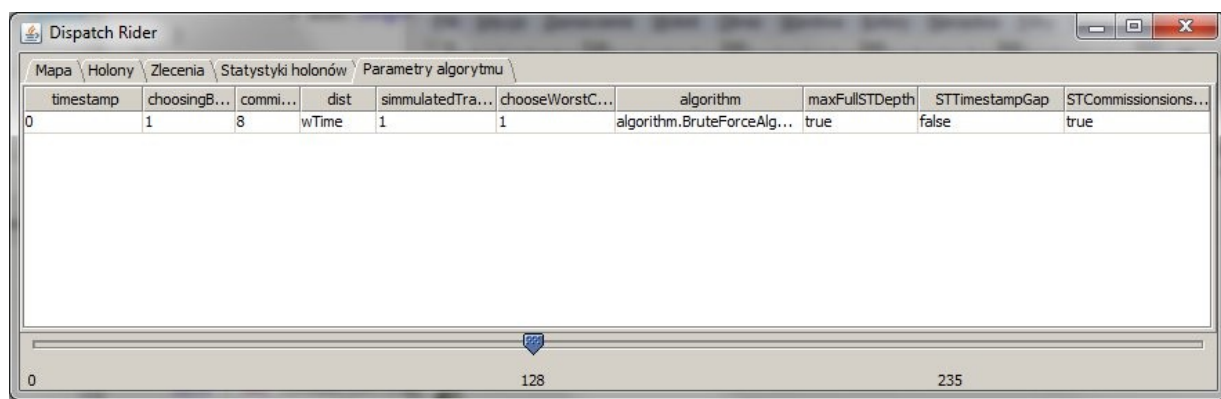
HolonID	Summary Cost	Cost	DriveTime	SummaryPunishment	Time	WaitTime
0	538.0	123.0	123.0	0.0	538.0	145.0
1	442.0	82.0	82.0	0.0	442.0	0.0
2	253.0	73.0	73.0	0.0	253.0	0.0
3	513.0	91.0	91.0	0.0	513.0	62.0
4	251.0	71.0	71.0	0.0	251.0	0.0
5	762.0	67.0	67.0	0.0	762.0	335.0
6	654.0	47.0	47.0	0.0	654.0	247.0
7	404.0	44.0	44.0	0.0	404.0	0.0
8	1102.0	38.0	38.0	0.0	1102.0	704.0
9	210.0	30.0	30.0	0.0	210.0	0.0

Rysunek 17: Zakładka statystyk holonów

W skład tabeli statystyki holonów wchodzi:

Tabela 4: Kolumny tabeli statystyk holonów

"HolonID"	Id holona
"Summary Cost"	Koszt sumaryczny
"Cost"	Koszt
"DriveTime"	Czas transportu
"SummaryPunishment"	Sumaryczna kara
"Time"	Czas sumaryczny
"WaitTime"	Czas postoju



Rysunek 18: Zakładka parametrów algorytmu

W skład tabeli parametrów algorytmu wchodzi:

Tabela 5: Kolumny tabeli parametrów algorytmu

"timestamp"	Punkt czasowy
"choosingByCost"	Tryb przydzielania zleceń
"commissionSendingType"	Sposób wysyłania zleceń (pojedynczo, grupami)
"dist"	Sposób wyliczania kosztu zlecenia (czas/dystans)
"simulatedTradingCount"	Ilość uruchomień Simulated Trading
"chooseWorstCommission"	Sposób wyboru najgorszego zlecenia
"algorithm"	Rodzaj algorytmu rozdziału zleceń
"maxFullSTDepth"	Maksymalny stopień zagłębienia w algorytmie complexST
"STTimestampGap"	Brak opisu w udostępnionej nam dokumentacji
"STCommissionsionsGap"	Brak opisu w udostępnionej nam dokumentacji

Dispatch Rider													
Mapa \ Holony \ Trasy \ Zlecenia \ Statystyki holonów \ Parametry algorytmu \													
timestamp	status	holon location	commission ID	pickup location	delivery location	pickup time 1	pickup time 2	delivery time 1	delivery time 2	pickup id	delivery id	actual load	load
0													
0	EN ROUTE	(40.0, 50.0)	42	(85.0, 35.0)	(95.0, 30.0)	47.0	124.0	387.0	456.0	70	81	0.0	30
7	EN ROUTE	(44.7894736842...	42	(85.0, 35.0)	(95.0, 30.0)	47.0	124.0	387.0	456.0	70	81	0.0	30
10	EN ROUTE	(46.8421052631...	42	(85.0, 35.0)	(95.0, 30.0)	47.0	124.0	387.0	456.0	70	81	0.0	30
18	EN ROUTE	(52.3157894736...	42	(85.0, 35.0)	(95.0, 30.0)	47.0	124.0	387.0	456.0	70	81	0.0	30
27	EN ROUTE	(61.0, 35.0)	42	(85.0, 35.0)	(95.0, 30.0)	47.0	124.0	387.0	456.0	70	81	0.0	30
29	EN ROUTE	(63.0, 35.0)	42	(85.0, 35.0)	(95.0, 30.0)	47.0	124.0	387.0	456.0	70	81	0.0	30
37	EN ROUTE	(71.0, 35.0)	42	(85.0, 35.0)	(95.0, 30.0)	47.0	124.0	387.0	456.0	70	81	0.0	30
51	PICKUP	(85.0, 35.0)	42	(85.0, 35.0)	(95.0, 30.0)	47.0	124.0	387.0	456.0	70	81	0.0	30
111	PICKUP	(85.0, 35.0)	42	(85.0, 35.0)	(95.0, 30.0)	47.0	124.0	387.0	456.0	70	81	0.0	30
113	PICKUP	(85.0, 35.0)	42	(85.0, 35.0)	(95.0, 30.0)	47.0	124.0	387.0	456.0	70	81	0.0	30
115	PICKUP	(85.0, 35.0)	42	(85.0, 35.0)	(95.0, 30.0)	47.0	124.0	387.0	456.0	70	81	0.0	30
121	PICKUP	(85.0, 35.0)	42	(85.0, 35.0)	(95.0, 30.0)	47.0	124.0	387.0	456.0	70	81	0.0	30
128	PICKUP	(85.0, 35.0)	42	(85.0, 35.0)	(95.0, 30.0)	47.0	124.0	387.0	456.0	70	81	0.0	30
145	PICKUP/DELIVERY	(88.0, 35.0)	40	(88.0, 35.0)	(88.0, 35.0)	109.0	170.0	109.0	170.0	104	78	0.0	20
197	PICKUP/DELIVERY	(88.0, 35.0)	40	(88.0, 35.0)	(88.0, 35.0)	109.0	170.0	109.0	170.0	104	78	0.0	20
204	PICKUP/DELIVERY	(88.0, 35.0)	40	(88.0, 35.0)	(88.0, 35.0)	109.0	170.0	109.0	170.0	104	78	0.0	20
220	PICKUP/DELIVERY	(88.0, 35.0)	40	(88.0, 35.0)	(88.0, 35.0)	109.0	170.0	109.0	170.0	104	78	0.0	20
225	PICKUP/DELIVERY	(88.0, 35.0)	40	(88.0, 35.0)	(88.0, 35.0)	109.0	170.0	109.0	170.0	104	78	0.0	20
228	PICKUP/DELIVERY	(88.0, 35.0)	40	(88.0, 35.0)	(88.0, 35.0)	109.0	170.0	109.0	170.0	104	78	0.0	20
235	EN ROUTE	(88.0, 35.0)	39	(90.0, 35.0)	(92.0, 30.0)	203.0	260.0	478.0	551.0	73	76	0.0	10
300	PICKUP	(90.0, 35.0)	39	(90.0, 35.0)	(92.0, 30.0)	203.0	260.0	478.0	551.0	73	76	0.0	10
301	PICKUP	(90.0, 35.0)	39	(90.0, 35.0)	(92.0, 30.0)	203.0	260.0	478.0	551.0	73	76	0.0	10
310	PICKUP	(90.0, 35.0)	39	(90.0, 35.0)	(92.0, 30.0)	203.0	260.0	478.0	551.0	73	76	0.0	10
311	PICKUP	(90.0, 35.0)	39	(90.0, 35.0)	(92.0, 30.0)	203.0	260.0	478.0	551.0	73	76	0.0	10
327	EN ROUTE	(90.0, 35.0)	38	(95.0, 35.0)	(88.0, 30.0)	293.0	360.0	574.0	643.0	77	71	0.0	20
390	PICKUP	(95.0, 35.0)	38	(95.0, 35.0)	(88.0, 30.0)	293.0	360.0	574.0	643.0	77	71	0.0	20

Rysunek 19: Zakładka tras

Zakładka tras pozwala wyświetlić wygenerowaną trasę dla danego holona. W celu wskazania należy wybrać odpowiednie ID z rozwijalnej listy umieszczonej nad tabelą.

W skład tabeli trasy wchodzi:

Tabela 6: Kolumny tabeli tras

"timestamp"	Punkt czasowy
"status"	EN ROUTE – w drodze, PICKUP – trwa odbiór towaru, DELIVERY – trwa dostarczanie towaru
"holon location"	Pozycja holonu na mapie
"commission ID"	ID aktualnie realizowanego zlecenia
"pickup location"	Miejsce załadunku
"delivery location"	Miejsce rozładunku
"pickup Time 1"	Czas początkowy załadunku
"pickup Time 2"	Czas końcowy załadunku
"delivery Time 1"	Czas początkowy rozładunku
"delivery Time 2"	Czas końcowy rozładunku
"pickup ID"	Id załadunku
"delivery ID"	Id rozładunku
"actual load"	Rzeczywisty załadunek
"load"	Załadunek

Mimo dołożenia starań i kontaktu z konsultantem nie udało nam się zapobiec zamykaniu aplikacji po zakończeniu symulacji.

Nasze próby polegały na:

- 1) Utworzeniu w tle wątku, który biernie czekając miał podtrzymać działanie aplikacji, nie obciążając przy tym zasobów.
- 2) Zmodyfikowaniu funkcji `simEnd` wywołującej się na koniec symulacji. Funkcja ta jest obecna w wielu klasach, w tym w `TestAgent` i `DistributorAgent`. O ile obiekt `TestAgent` jej nie wywoływał, to `DistributorAgent` owszem. Dodaliśmy do niej nieskończoną petlę.

W obu przypadkach program zamykał się w krótkim czasie po zakończeniu symulacji.

10. Zmiany wprowadzone w systemie Dispatch Rider

Aby połączyć stworzony przez nas moduł GUI z resztą systemu, musieliśmy dokonać kilku modyfikacji w systemie Dispatch Ridera.

Drobnym zmianom uległy metody `Dtp.jade.gui.GUIAgent.addSimulationData`, `Dtp.jade.gui.GUIAgent.changeGraph`, `Dtp.jade.gui.GUIAgent.simulationStart`, `Dtp.jade.gui.GUIAgent.sendTimestamp`, oraz `dtp.jade.distributor.DistributorAgent.setCommissions`. Dodano w nich kod przesyłający dane prezentowane użytkownikowi. Instrukcje te są postaci `SingletonGUI.getInstance().metoda`. Zmiany te i role poszczególnych wywołań opisane zostały szczegółowo we wcześniejszej części dokumentacji.

11. Podsumowanie

Realizacja projektu pozwoliła nam zetknąć się z problemami powstającymi podczas pracy nad cudzym kodem. Nasz komponent, jako integralna część systemu, musi komunikować się z innymi komponentami i wpływać na działanie całej aplikacji. Było to o tyle utrudnione, że kod jest ciągle rozwijany i brakuje przejrzystej dokumentacji.

Podczas prac napisaliśmy część odpowiedzialną za wyświetlanie danych, jak i część odpowiedzialną za integrację z resztą systemu. Jako, że zdecydowaliśmy się na wzorzec singleton, integracja przebiega zwykle w postaci wstawieniu kilku linii kodu w celu przesłania odpowiednich informacji.

Pomimo napotkanych trudności udało nam się zakończyć projekt w przewidzianym terminie. Produkt finalny spełnia założenia projektowe postawione przed nami przy rozpoczęciu prac, to jest informowanie użytkownika na bieżąco o stanie symulacji i obrazowanie zależności zachodzących między agentami, oraz graf rozwiązania. Uwzględnia przy tym dodatkowe elementy takie jak miękkie okna czasowe, spóźnienia, graf reprezentujący sieć drogową oraz zmienne czasy przejazdów.

Jako główny kierunek dalszego rozwoju z punktu widzenia naszego modułu proponujemy uwzględnienie wizualizacji z podziałem na wiele testów. Możliwa jest również taka rozbudowa naszego projektu, aby przedstawić (zarówno w sposób graficzny jak i tekstowy) aktualnie planowaną trasę holonu, przewidzianą aż do końca symulacji. Dane te nie są udostępnione do wyświetlenia, dlatego należałoby zadbać o ich samodzielne wyliczenie. Musiałoby to oczywiście odbyć się z wykorzystaniem tych samych modułów, co w trakcie faktycznej symulacji, w celu zachowania zgodności ze stanem faktycznym.

12. Zastosowane technologie

Całość systemu odziedziczonego została zaimplementowana w języku Java, wersja 5. Kod jest w pełni kompatybilny ze stosowaną przez nas Javą 6.

Do budowy systemu agentowego został zastosowany jadowy framework JADE (Java Agent Development framework) dostępny na licencji GNU LGPL . Strona internetowa technologii: <http://jade.tilab.com/>

Do stworzenia graficznej wizualizacji grafu symulacji wybraliśmy technologię JUNG [5], która została wykorzystana także w pracy [2].

13. Bibliografia

- [1] Dokumentacja
„dokumentacja 6.03.2012”
- [2] Dudek Maciej, Rafał Pysz. Dokumentacja
„DispatchRider. Opracowanie modułów odpowiedzialnych za wizualizację działania symulacji procesów transportowych ”
- [3] Gołacki Michał. Praca dyplomowa magisterska
„Modelowanie transportu z użyciem holonów”
- [4] JADE
<http://jade.tilab.com/>
- [5] JUNG
<http://jung.sourceforge.net/>
- [6] Konieczny Michał. Praca dyplomowa magisterska
„Modelowanie i optymalizacja transportu w sytuacjach kryzysowych ”