AKADEMIA GÓRNICZO-HUTNICZA

Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki

KATEDRA INFORMATYKI

DISPATCH RIDER

HOLONICZNY SYSTEM DO ROZWIĄZYWANIA DYNAMICZNEGO PROBLEMU TRANSPORTOWEGO

INSTRUKCJA KONFIGURACJI I INSTALACJI SYSTEMU

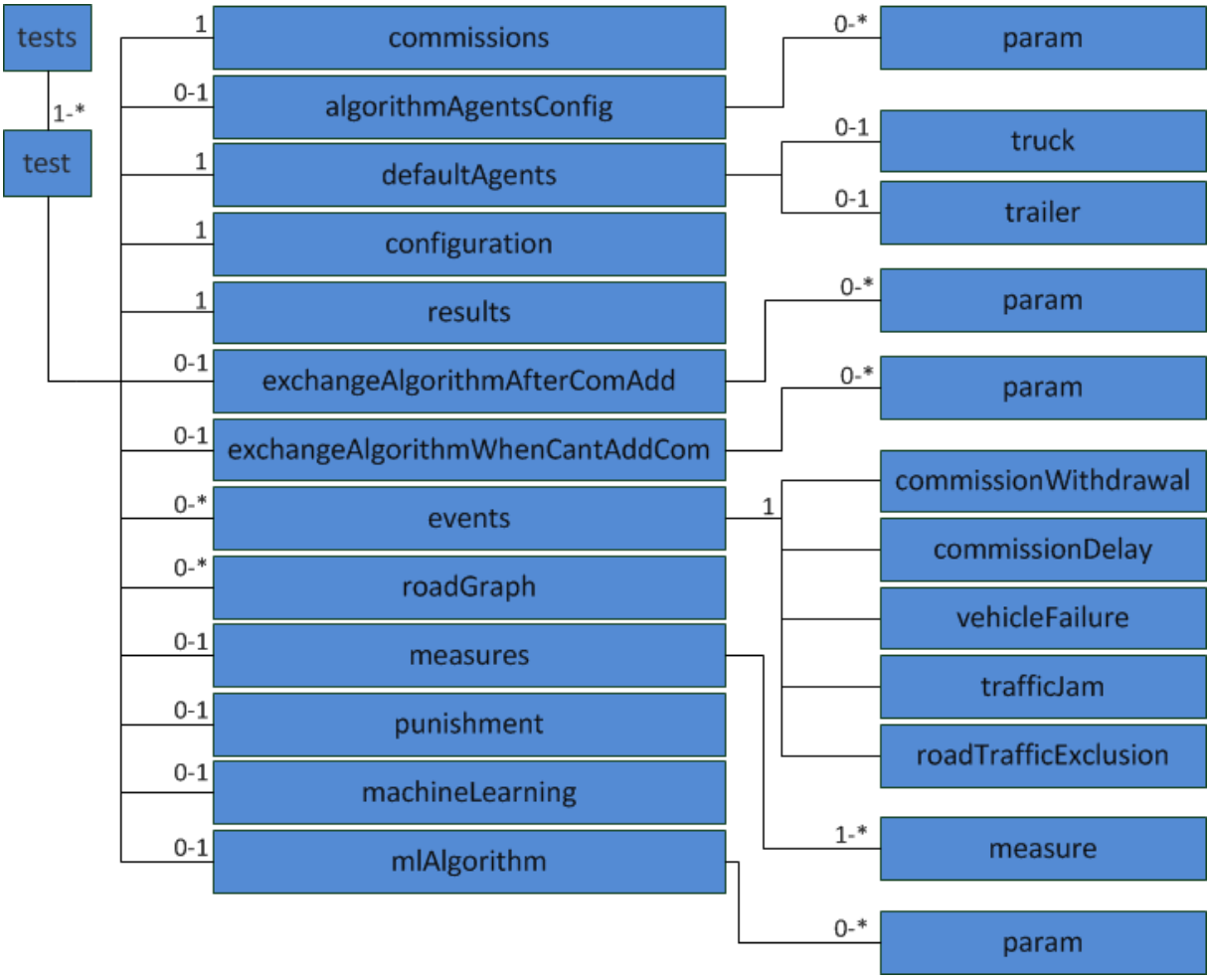
Kraków, sierpień 2012

Contents

1. Konfiguracja systemu

1.1. Instrukcja konfiguracji głównej

System konfigurowany jest bezpośrednio poprzez odpowiednio sprokurowany plik XML. Hierarchia elementów konfiguracji wraz z możliwymi ich krotnościami występowania została przedstawiona na rysunku poniżej



W poniższych tabelach zawarto skrócony opis wszystkich zawartych w grafice elementów. Typy specjalne to typy zdefiniowane w scheme'ie XSD. Typ sekwencyjny odnosi się do ilościowo dowolnej sekwencji elementów podrzędnych w danym elemencie.

1.1.1. Główne elementy konfiguracji

Plik konfiguracyjny powinien rozpoczynać się od elementu *tests*, który z kolei zawiera w sobie dowolną liczbę elementów *test*. Element ten powinien przy okazji

odwoływać się do odpowiedniej scheme'y XSD. Przykładowy kod przedstawia opisaną sytuację:

```
<?xml version="1.0" encoding="UTF-8"?>
<tests xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="xml/schemes/configuration.xsd">
    <test>
        ...
    </test>
</tests>
```

W rozdziale podano również tabele z opisem możliwych atrybutów dla każdego omawianego elementu.

tests			
Atrybuty	**Wymagane**	**Typ**	**Opis**
Zawiera dowolną liczbę elementów *test*.			

test			
Atrybuty	**Wymagane**	**Typ**	**Opis**
Definiuje konfigurację danej symulacji za pomocą wielu atrybutów podrzędnych.			
gui	Nie	Boolean	Domyślnie *„false”*. Wartość *true* oznacza inicjalizację GUI dla danej symulacji.

1.1.2. Element *commissions*

commissions			
Atrybuty	**Wymagane**	**Typ**	**Opis**
Konfiguruje główne parametry symulacji.			
dynamic	Nie	String	Domyślnie *„false”*. Określa, czy problem jest statyczny, czy dynamiczny.
packageSending	Nie	Boolean	Domyślnie *„false”*. Wybór trybu wysyłania zleceń.
choosingByCost	Nie	Boolean	Domyślnie *„true”*. Wybór jednostki do realizacji zlecenia po koszcie, lub dbając o maksymalne wypełnienie jednostek.
simmulatedTrading	Nie	Int	Domyślnie *„1”*. Określa, ile razy

			ma być uruchamiane *simulatedTrading*.
STDepth	Nie	Int	Domyślnie *„1”*. Stopień zagłębienia w algorytmie *complexSimmulatedTrading.*
firstComplexSTResultOnly	Nie	Boolean	Domyślnie *„true”*. Określa tryb działania algorytmu *complexSimmulatedTrading*. Czy ma być wybierana pierwsza znaleziona konfiguracja zleceń, czy też konfiguracja optymalna.
algorithm	Nie	String	Domyślnie *„BruteForceAlgorithm2”.* Wybór jednego z zaimplementowanych algorytmów do wstawiania zleceń w pakiecie *algorithm*.
worstCommissionByGlobalTime	Nie	Boolean	(True/False) Ustawienie, czy *worstCommission* ma być wybierane po przyroście czasu, czy dystansu.
chooseWorstCommission	Nie	Specjalny	(„time”, „wTime”, „timeWithPunishment”, „distWithPunishment”) Domyślnie *„time”*. Znaczenie jak powyżej – *worstCommissionByGlobalTime* jest pozostawione tylko dla zachowania kompatybilności poprzednich konfiguracji.
dist	Nie	Boolean	Domyślnie *„true”*. Określa, czy koszt nowego zlecenia ma być obliczany na podstawie przyrostu dystansu, czy czasu. Parametr wykorzystywany jest jedynie w momencie, kiedy Eunit ma odesłać Dystrybutorowi swoją propozycję(koszt) realizacji nowego zlecenia, żeby ten mógł wybrać najlepszą jednostkę, której przydzieli zlecenie
autoConfig	Nie	Boolean	Domyślnie *„false”*. Określa, czy konfiguracja użytego algorytmu ma być dokonana automatycznie. Ustawienie tego atrybutu na *true* wyklucza możliwość uczenia maszynowego (*exploration=”true”*).

recording	Nie	Boolean	Domyślnie *„false”*. Określa, czy mają być generowane pliki z zapisem symulacji i drogami przejechanymi przez eunity.
STTimeGap	Nie	Int	Domyślnie *„1”*. Ustawienie częstości uruchamiania *SimulatedTrading* w zależności od ilości znaczników czasowych.
STCommissionGap	Nie	Int	Domyślnie *„1”*. Ustawienie częstości uruchamiania *SimulatedTrading* w zależności od ilości przydzielonych zleceń.
confChange	Nie	Boolean	Domyślnie *„false”*. Określa, czy chcemy uruchomić dynamiczną zmianę konfiguracji w czasie symulacji.
commissionsComparator	Nie	String	Domyślnie *„Basic”*. Ustawia odpowiedni komparator dla algorytmu. Komparatory znajdują się w paczce *algorithm.comparator*. Wartość klucza powinna zawierać nazwę klasy bez sufiksu *CommissionsComparator*.

1.1.3. Element *algorithmAgentsConfig*

Służy konfiguracji agentów pomocniczych. W celu przekazania jakichś parametrów inicjalizacyjnych musimy do pliku konfiguracyjnego dodać element:

```
<algorithmAgentsConfig>
    <param name="X" value="Y"/>
    ...
    <param name="X" value="Y"/>
</algorithmAgentsConfig>
```

Potem pary *name*, *value* są przekazywane do metody *init* każdego agenta pomocniczego. Element *algorithmAgentsConfig* jest opcjonalny, ale jeśli występuje, to musi znajdować się między elementami *commissions* i *defaultAgents*.

algorithmAgentsConfig			
Atrybuty	**Wymagane**	**Typ**	**Opis**
Służy ustawieniu dodatkowych parametrów inicjalizacyjnych dla agentów pomocniczych. Powinien znajdować się pomiędzy elementami *commissions* i *defaultAgents*.			
param		Sekwencyjny	Za pomocą par *name* i *value* podajemy odpowiednie argumenty o odpowiednich wartościach do metody *init* każdego agenta pomocniczego. Przykładowo: „<param name=”X” value=”Y”/>”.

1.1.4. Elementy *defaultAgents, truck i trailer*

Element *defaultAgents* zawiera w sobie elementy *trailer* i *truck*. Przykładowo:

```
<defaultAgents>
    <truck power="600" reliability="4" comfort="4" fuelConsumption="20"/>
    <trailer mass="200" capacity="200" cargoType="1" universality="4"/>
</defaultAgents>
```

defaultAgents			
Atrybuty	**Wymagane**	**Typ**	**Opis**
Służy ustawieniu i skonfigurowaniu agentów domyślnych.			
truck	Nie	Specjalny	Konfiguracja pojazdu.
trailer	Nie	Specjalny	Konfiguracja przyczepy.

truck			
Atrybuty	**Wymagane**	**Typ**	**Opis**
Służy konfiguracji pojazdu. Dla typów *Int* można nadać dowolne wartości całkowitoliczbowe. Służą one jako parametry dla danej funkcji kosztu.			
power	Tak	Int	Moc pojazdu.
reliabilty	Tak	Int	Niezawodność pojazdu.
comfort	Tak	Int	Komfort prowadzenia pojazdu.
fuelConsumption	Tak	Int	Zużycie paliwa.

trailer			
Atrybuty	**Wymagane**	**Typ**	**Opis**
Służy konfiguracji przyczepy. Dla typów *Int* można nadać dowolne wartości całkowitoliczbowe. Służą one, jako parametry dla danej funkcji kosztu.			
mass	Tak	Int	Masa przyczepy.
capacity	Tak	Int	Ładowność.
cargoType	Tak	Int	Typ ładunku.
universality	Tak	Int	Stopień uniwersalności przyczepy.

1.1.5. Elementy *configuration* oraz *results*

Przykładowy zapis:

```
<configuration>tmp_tests/25_200</configuration>
<results>tmp_tests/results/wynik25_200</results>
```

configuration			
Atrybuty	**Wymagane**	**Typ**	**Opis**
Wskazuje na katalog z plikami konfiguracyjnymi holonów. Katalog powinien zawierać się pomiędzy <configuration> a </configuration>.			

results			
Atrybuty	**Wymagane**	**Typ**	**Opis**
Zawiera ścieżkę do pliku z wynikami wraz z nazwą pliku wynikowego. Dane te powinny zawierać się pomiędzy <results> a </results>.			

1.1.6. Elementy *exchangeAlgorithmAfterComAdd* i *exchangeAlgorithmWhenCantAddCom*

Elementy te związane są z wprowadzaniem algorytmów optymalizacji działających podobnie, jak obecne *SimmulatedTrading*.

exchangeAlgorithmAfterComAdd			
Atrybuty	**Wymagane**	**Typ**	**Opis**
Określa parametry dla algorytmów STLike. Jeśli jest używany, to wszystkie parametry dotyczące ST są ignorowane. Metoda wołana jest wtedy, co poprzednio *fullSimmulatedTrading*.			
name	Tak	String	Nazwa algorytmu(klasy).
param	Nie	Sekwencyjny	Za pomocą par *name* i *value* podajemy odpowiednie argumenty o odpowiednich wartościach dla algorytmu.

exchangeAlgorithmWhenCantAddCom			
Atrybuty	**Wymagane**	**Typ**	**Opis**
Określa parametry dla algorytmów STLike. Jeśli jest używany, to wszystkie parametry dotyczące ST są ignorowane. Metoda ta wołana jest wtedy, co poprzednio *complexSimmulatedTrading*.			
name	Tak	String	Nazwa algorytmu(klasy).
param	Nie	Sekwencyjny	Za pomocą par *name* i *value* podajemy odpowiednie argumenty o odpowiednich wartościach dla algorytmu.

1.1.7. Element *events*

events			
Atrybuty	**Wymagane**	**Typ**	**Opis**
Umożliwia określanie wydarzeń losowych(sytuacji kryzysowych) występujących w trakcie symulacji za pomocą podrzędnych elementów: *commissionWithdrawal*, *commissionDelay*, *vehicleFailure*, *trafficJam* i *roadTrafficExclusion*.			

commissionWithdrawal			
Atrybuty	**Wymagane**	**Typ**	**Opis**
Określa sytuacje kryzysową dotyczącą wycofania zlecenia z systemu.			
time	Tak	Int	
commission	Tak	Int	

commissionDelay			
Atrybuty	**Wymagane**	**Typ**	**Opis**
Określa sytuacje kryzysową dotyczącą opóźnienia zlecenia w punkcie odbioru.			
time	Tak	Int	
commission	Tak	Int	
delay	Tak	Int	

vehicleFailure			
Atrybuty	**Wymagane**	**Typ**	**Opis**
Określa sytuacje kryzysową dotyczącą awarii jednostki transportowej.			
time	Tak	Int	
vehicle	Tak	Int	
duration	Tak	Double	

trafficJam			
Atrybuty	**Wymagane**	**Typ**	**Opis**
Określa sytuacje kryzysową dotyczącą zatoru drogowego.			
time	Tak	Int	
startX	Tak	Double	
startY	Tak	Double	
endX	Tak	Double	
endY	Tak	Double	
cost	Tak	Double	

roadTrafficExclusion			
Atrybuty	**Wymagane**	**Typ**	**Opis**
Określa sytuacje kryzysową dotyczącą wyłączenia odcinka drogi z ruchu.			
time	Tak	Int	
startX	Tak	Double	
startY	Tak	Double	
endX	Tak	Double	
endY	Tak	Double	

1.1.8. Element *roadGraph*

Element powinien zostać umieszczony w konfiguracji, jeśli chcemy, aby system korzystał z grafu. Przykładowy wpis:

```
<roadGraph graphChanges="XXX" trackFinder="XXX" ST="XXX">
    Ścieżka do pliku z opisem grafu
</roadGraph>
```

roadGraph			
Atrybuty	**Wymagane**	**Typ**	**Opis**
Specyfikuje graf, z którego może korzystać system.			
graphChanges	Tak	String	Określa ścieżkę do pliku ze zaburzeniami grafu.
trackFinder	Tak	Specjalny	(Enumerator: „Astar”, „Dijkstra”, „SImmulatedAnnealing”). Określa algorytm poszukiwania ścieżki między dwoma węzłami grafu. *TrackFindery* są zaimplementowane w paczce *dtp.optimization*.
changeTime	Nie	Specjalny	(“immediately”, “afterChangeNotice”, “aftertime”) Domyślnie *„immediately”*. Określa, kiedy rozgłaszać o zaburzeniu w grafie.
notificationTime	Nie	Int	Domyślnie *„10”*. Używany tylko, gdy *changeTime=”afterTime”*. Określa po ilu timestampach ma być rozgłoszona wiadomość o zaburzeniu.
predictor	Ne	String	Domyślnie *„Standard”*. Nazwa predyktora zaburzeń w grafie(w paczce *dtp.graph*; bez końcówki *GraphLinkPredictor*).
historySize	Nie	Int	Domyślnie *„4”*. Ilość poprzednich zaburzeń grafu trzymanych w historii predyktora.
ST	Nie	Boolean	Domyślnie *„true”*. Określa, czy po zaburzeniu grafu ma być uruchamiane *SimmulatedTrading*.

1.1.9. Element *measures*

Element niezbędny, aby system wyliczał miary. Powinien znajdować się pomiędzy elementami *roadGraph* i *punishment*. *Należy* zauważyć, że po skonfigurowaniu wyliczania miar powstanie tyle dodatkowych plików wynikowych, ile podamy formatów w atrybucie *formats*. Ich nazwy będą w postaci:

<nazwa podana w tagu <results>>_measures.<rozszerzenie podane w atr formats>

measures Atrybuty	Wymagane	Typ	Opis
Służy konfiguracji wyliczania miar. W przypadku braku tej sekcji, miary nie będą wyliczane.			
formats	Tak	String	Nazwy rozszerzeń, dla których istnieje odpowiedni printer, oddzielone spacją.
timeGap	Nie	Int	Domyślnie *„1”*. Określa, co ile timestampów mają być wyliczane miary. Działa analogicznie jak *STTimeGap* tylko, że dotyczy wyliczania miar.
measure	Tak	Sekwencyjny	Każdy *measure* posiada dodatkowy boolowski atrybut *visualise*(domyślnie *„false”*), który określa, czy chcemy wizualizować daną miarę w trakcie działania systemu(true), czy też nie(false). Ponadto pomiędzy *<measure>*, a *</measure>* należy podać nazwę kalkulatora z pakietu *measure*.

1.1.10. Element *punishment*

Element ten konfiguruje i włącza miękkie okna. Jeśli elementu nie będzie w konfiguracji, to symulacja zadziała w trybie dla twardych okien czasowych.

Przykładowa konfiguracja(warto zauważyć, iż w związku z ograniczeniami języka XML co do używania znaków „<”, „>”, używane są tu: „<” zamiast „<” oraz „≥” zamiast „>=”.

```
<punishment funtion="latency;latency&lt;10?latency*latency;latency&ge;10"
holons="10" delayLimit="30"/>
```

punishment Atrybuty	Wymagane	Typ	Opis
Służy konfiguracji miękkich okien.			
function	Tak	String	Określa funkcje kary.
default	Nie	String	(Format: „nazwa1:wartość1;nazwa2:wartość2;”…) Wartości domyślne w funkcji kary.
holons	Tak	Int	Preferowana ilość holonów. Określa maksymalną liczbę jednostek, które jesteśmy w stanie zaakceptować, jako dobre rozwiązanie.
delayLimit	Nie	Double	Maksymalny dopuszczalny limit procentu spóźnienia w trasie holonu.

1.1.11. Element *machineLearning*

Element odpowiada za uruchomienie i konfigurację uczenia maszynowego w systemie. Po zakończeniu symulacji, tabela w podanym pliku zostanie uzupełniona o sekcję z jej zawartością, gdzie opisywane są wartości kolejnych jej komórek.

Scheme’a podanej tabeli znajduje się zazwyczaj w pliku *mltable.xsd*. Przykładowy plik z tabelą wygląda następująco:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<MLTable xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="xml/schemes/mltable.xsd">
    <structure>
        <globalStates k="0.5">
            <state name="S2" value="avg(MaxWaitTime) &gt;= 10"/>
            <state name="S1" value="avg(MaxWaitTime) &lt; 10"/>
        </globalStates>
        <globalActions factor="0.9" function="holonsCount">
            <action name="A2" simmulatedTrading="0"/>
            <action name="A1" simmulatedTrading="1"/>
        </globalActions>
    </structure>
</MLTable>
```

1.1.11.1. Funkcja wynagrodzenia

Jak widać przy definicji stanów musimy podać funkcję nagrody (reward). Możemy używać dowolnych operatorów matematycznych i dodatkowo parametrów które są ustawiane w *machineLearning.Helper.getParameters*, oraz miar (dla funkcji globalnej z agregatorami, dla holonów i z agregatorami i bez).

Na razie dostępnymi parametrami są:

- holonsCount – ilość holonów
- dist – sumaryczny dystans wszystkich holonów (dotyczy zleceń w ich kalendarzach)
- commissions – ilosc zleceń w kalendarzach
- costOfCommission – koszt realizacji pojedynczego zlecenia przez holon (sumaryczny koszt na, który składa się sumaryczny dystans i sumaryczna kara przy miękkich oknach, podzielony przez sumaryczną ilość zleceń)
- timeFromCreationOfLastUnit – czas od stworzenia ostatniej nowej jednostki
- holonDist, holonCommissions, holonCostOfCommission – jak wyżej tylko dla pojedynczego holonu

Dodatkowo, każdy z ww. parametrów ma swój odpowiednik z „_" na początku (np.: *dist).* Użycie w funkcji parametru z „_" oznacza jego poprzednią wartość.

Ogólnie funkcja nagrody jest funkcją złożoną definiowaną jako:

```
funkcja1;warunek1?funkcja2;warunek2?funkcja3;warunek3;…
```

Dzięki takiemu podejściu możemy nagradzać i karać, przykładowo jeśli chcemy nagradzać (reward = 100) gdy koszt realizacji zlecenia maleje, a nic nie robić (reward = 0) w przeciwnym wypadku możemy zapisać funkcje nagrody nastepująco:

```
100;costOfCommission<_costOfCommission?0;costOfCommission>-_costOfCommission
```

1.1.11.2. Wyliczanie wartości komórek

W zależności od wybranego trybu pracy (deterministyczny, lub niedeterministyczny) wartości komórek tabeli są wyliczane w różny sposób. W przypadku działania deterministycznego (deterministic=true) wzór wygnąda następująco:

$$\overline{Q}(s,a) = r + \gamma \max_{a'} \overline{Q}(s',a')$$

Natomiast gdy deterministic=false wzór wygląda następująco:

$$\overline{Q}_n(s,a) = (1-\alpha_n)\overline{Q}_{n-1}(s,a) + \alpha_n[r + \gamma \max_{a'} \overline{Q}_{n-1}(s',a')]$$

gdzie:

$$\alpha_n = \frac{1}{1 + visits_n(s,a)}$$

W powyższych wzorach przyjęto oznaczenia:

- $\overline{Q}(s,a)$ - wartość komórki w stanie s dla akcji a
- r – wartość funkcji nagrody
- visits(s,a) – ilość odwiedzin komórki (s,a)

machineLearning			
Atrybuty	**Wymagane**	**Typ**	**Opis**
Określa parametry uczenia maszynowego. Pomiędzy *<machineLearning …>*, a *</machineLearning>* powinna znaleźć się ścieżka do pliku z opisem tabeli.			
exploration	Nie	Boolean	Domyślnie *„false”*. Określa, czy tabela jest modyfikowana, czy używamy jej tylko do wyboru optymalnej konfiguracji. Ustawienie tego atrybutu na *true* wyklucza możliwość automatycznej zmiany konfiguracji (*autoConfig=”true”*).
params	Nie	String	Zawiera stałe występujące w funkcji wynagrodzenia. Kolejne stałe oddzielone są średnikami(np. *„params=”bestDist=1234;const=2””*).

1.1.12. Element *mlAlgorithm*

Przykładowy wpis przedstawiono poniżej.

```
<mlAlgorithm file="clustable.xml" algorithm="Clustering" exploration="true">
    <param name="bestDist" value="828.94"/>
</mlAlgorithm>
```

mlAlgorithm **Atrybuty**	**Wymagane**	**Typ**	**Opis**
Umożliwia przypisanie do symulacji i ustawienie konkretnego algorytmu uczenia maszynowego.			
file	Tak	String	Ścieżka do pliku z reprezentacją wiedzy wykorzystywaną przez algorytm.
exploration	Nie	Boolean	Domyślnie *„false”*. Określa, czy używamy algorytmu w trybie eksploracji(uczenia).
algorithm	Nie	String	Domyślnie *„QLearning”*. Nazwa klasy w paczce machineLearning implementującej algorytm. Przy wyborze *„Clustering”* należy wskazać ścieżkę do tablicy klastrowania, która zostanie opisana poniżej
param	Nie	Sekwencyjny	Za pomocą par *name* i *value* podajemy odpowiednie argumenty o odpowiednich wartościach dla powyższego algorytmu.

1.1.13. Przykładowa konfiguracja

Poniżej przedstawiono przykładową konfigurację systemu. Wartości podmieniono w celu uzyskania przejrzystości.

```
<?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="XXX" dynamic="XXX" dist="XXX" packageSending="XXX"
            choosingByCost="XXX" STDepth="XXX" worstCommissionByGlobalTime="XXX"
            algorithm="XXX" simmulatedTrading="XXX" STTimeGap="XXX" STCommissionGap="XXX"
            autoCofig="XXX" confChange="XXX" firstComplexSTResultOnly="XXX">
            Ścieżka do pliku z benchmarkiem
        </commissions>
        <defaultAgents>
            <truck power="Y" reliability="Y" comfort="Y" fuelConsumption="Y"/>
            <trailer mass="Y" capacity="Y" cargoType="Y" universality="Y"/>
        </defaultAgents>
        <configuration>
            Ścieżka do katalogu z z plikami konfiguracyjnymi holonów(*.properties)
        </configuration>
        <results>Ścieżka do pliku wynikowego wraz z nazwą pliku</results>
        <exchangeAlgorithmAfterComAdd name="XXX">
            <param name="XXX" value="XXX"/>
            …
            <param name="XXX" value="XXX"/>
        </exchangeAlgorithmAfterComAdd>
        <exchangeAlgorithmWhenCantAddCom name="XXX">
            <param name="XXX" value="XXX"/>
            …
            <param name="XXX" value="XXX"/>
        </ exchangeAlgorithmWhenCantAddCom>
        <roadGraph graphChanges="XXX" trackFinder="XXX" ST="XXX" changeTime="XXX"
            notificationTime="XXX" predictor="XXX" historySize="XXX">
            Ścieżka do pliku z opisem grafu
        </roadGraph>
        <measures formats = „XXX" timeGap="XXX">
            <measure visualize="XXX">miara 1</measure>
            …
            <measure visualize="XXX">miara n</measure>
        </measures>
        <punishment function="XXX" default="XXX" holons="XXX" delayLimit="XXX" />
        <machineLearning exploration="XXX">Ścieżka do pliku z definicją tablicy</machineLearning>
    </test>
</tests>
```

1.1.14. Format pliku z opisem grafu

Opis grafu musi być podany w formie xml'a. W skrócie struktura pliku jest następująca:

```
<?xml version="1.0"?>
<network xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<!---------------------- opis kolejnych punktów i wychodzących z nich połączeń ---------------------->
    <point>
        <id> jakiś numer używany jako odnośnik przy def połączeń </id>
        <name> dowolna nazwa - najczęściej reprezentująca współrzędne </name>
        <position>
            <x> współrzędna x węzła </x>
            <y> współrzędna y węzła </y>
        </position>
        <isbase> true tylko dla bazy </isbase>
        <!-------------------------- def kolejnych połączeń wychodzących z węzła ----------->
        <route>
            <id_r> id innego węzła do którego jest połączenie </id_r>
            <cost> koszt połączenia - czas przejazdu </cost>
        </route>
        <!-------------------------------------------------------------------------------->
    </point>
<!------------------------------------------------------------------------------------------------>
</network>
```

UWAGA:

Jak widać po powyższej strukturze graf jest skierowany. Oznacza to, że:

- Ścieżka między węzłami A i B może być inna od ścieżki z B do A
- Ścieżka z A do B może mieć inny czas przejazdu niż z B do A

1.1.15. Niepewne czasy przejazdu

Plik ze zmianami czasów przejazdu wygląda następująco:

```
<?xml version="1.0" encoding="UTF-8"?>
<graphChanges xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="xml/schemes/graphChanges.xsd">
<!----------------------- deklaracje zmian ------------------------------>
<change time="X"> - zmiana ma zostać wprowadzona w timestampie X
    <!--------------------- opis jakie zmiany i gdzie ------------------>
        <link sPoint="id_A" ePoint="id_B" cost="c" both="b" />
    <!------------------------------------------------------------------>
    </change>
<!------------------------------------------------------------------------>
</graphChanges>
```

gdzie:

- id_A, id_B – id węzłów grafu
- c – nowy koszt połączenia
- b – true, jeśli chcemy zmienić połączenie w obie strony na wartość c

1.1.16. Klastrowanie

Wybór *„Clustering”* jako *mlAlgorithm* wymaga wskazania poprawnie zdefiniowanej tablicy z klastrowaniem(tzw. *Clustable*). W celu pełnego zrozumienia budowy pliku z tablicą warto zapoznać się z odpowiednią scheme’ą XSD(*clustable.xsd*).

Ogólnie rzecz ujmując w pliku zdefiniowane są następujące rzeczy:

- Atrybut tagu ClusTable – learning : true – tryb uczenia, false – tryb pracy
- Globalne measurmenty używane do wyznaczania stanów globalnych
- Lokalne measurmenty używane do wyznaczania stanów lokalnych
- Globalne stany
- Lokalne stany
- Globalne akcje
- Lokalne akcje
- Globalny content – wartości komórek tabeli globalnej
- Lokalny content – wartości komórek tabeli lokalnej

Poniżej umieszczono przykładową konfiguracji opisywanego pliku.

```
<?xml version="1.0" encoding="UTF-8"?>
<ClusTable xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="xml/schemes/clustable.xsd" learning="true">
    <structure>
        <globalMeasures>
            <measure name="M1" value="avg(AverageMinDistBetweenAllCommissions)" />
            <measure name="M2" value="avg(WaitTime)" />
            <measure name="M3" value="avg(AverageNumberOfComsWithinTimeWinOfAllCommissions)" />
        </globalMeasures>
        <globalStates k="0.0">
            <state name="S1">
                <measure name="M1" value="7.89" />
                <measure name="M2" value="10" />
                <measure name="M3" value="4" />
            </state>
        </globalStates>
        <globalActions factor="0.9"
            func-tion="2*_holonsCount/holonsCount+bestDist/dist+_costOfCommission/costOfCommission">
            <action STDepth="0" choosingByCost="false" name="A2" simmulatedTrading="0" />
            <action STDepth="0" choosingByCost="true" name="A1" simmulatedTrading="0" />
            ...
        </globalActions>
        <holonMeasures>
            <measure name="M1" value="AverageMinDistBetweenAllCommissions" />
            <measure name="M2" value="WaitTime" />
            <measure name="M3" value="AverageNumberOfComsWithinTimeWinOfAllCommissions" />
        </holonMeasures>
        <holonStates k="3">
            <measure name="M1" value="7.89" />
            <measure name="M2" value="10" />
            <measure name="M3" value="4" />
        </holonStates>
        <holonActions factor="0.9"
            func-tion="2*_holonsCount/holonsCount+bestDist/dist+_costOfCommission/costOfCommission">
            <action name="A1" algorithm="a1" newCommissionCostByDist="true" simmulated-Trading="true"/>
            ....
        </holonActions>
    </structure>
```

```
<content>
    <globalTableContent>
        <state name="S10">
            <action name="A2" useCount="0" value="0.0" />
            <action name="A1" useCount="0" value="0.0" />
            ...
        </state>
        <state name="S3">
            <action name="A2" useCount="0" value="0.0" />
            <action name="A1" useCount="0" value="0.0" />
            ...
        </state>
        ...
    </globalTableContent>
    <holonTableContent>
        <state name="S1">
            <action name="A1" useCount="2" value="15" />
            ...
        </state>
    </holonTableContent>
</content>
</ClusTable>
```

1.2. Instrukcja konfiguracji elementów holonu, oraz zleceń

Pliki *drivers.properties*, *trucks.properties*, *trailers.properties*, *holonic.properties* powinny znajdować się w jednym katalogu wskazywanym przez element *configuration* w głównym pliku konfiguracyjnym.

1.2.1. Konfiguracja loggera

Plik konfiguracyjny znajduje się w *conf/Log4j.properties*. Umieszczono tam w komentarzach instrukcje jego konfiguracji.

1.2.2. Konfiguracja kierowców

Plik konfiguracyjny powinien nazywać się *drivers.properties*. W pierwszej linii znajduje się liczba kierowców i opcjonalnie domyślna funkcja kosztu.

Przykładowo:

20	0.01*dist*(4-comfort)+(dist/100)*fuel*((mass+load)/power)

1.2.3. Konfiguracja ciągników

Plik konfiguracyjny powinien nazywać się *trucks.properties*. W pierwszej linii znajduje się liczba ciągników T i opcjonalnie domyślna funkcja kosztu. W kolejnych T liniach:

<moc> <niezawodność: 1-4> <wygoda: 1-4> <zużycie paliwa: 0-100> <typ zaczepu> <opcjonalna funkcja kosztu(tylko, jeśli inna od domyślnej)>

Przykładowo:

```
5       0.01*dist*(4-comfort)+(dist/100)*fuel*((mass+load)/power)
400     1       2       20      1
120     2       2       30      1
200     3       2       40      1
500     4       2       50      1
150     1       1       60      1
```

1.2.4. Konfiguracja naczep

Plik konfiguracyjnych powinien nazywać się *trailers.properties*. W pierwszej linii znajduje się liczba naczep N i opcjonalna domyślna funkcja kosztu. Pozostałe N linii:

<masa własna> <pojemność> <typ ładunku> <uniwersalność: 1-4> <typ zaczepu> <opcjonalna funkcja kosztu(tylko, jeśli inna od domyślnej)>

Przykładowo:

```
5       0.01*dist*(4-comfort)+(dist/100)*fuel*((mass+load)/power)
300     300     1       1       1
100     300     2       1       1
200     200     3       1       1
500     400     1       1       1
150     400     2       1       1
```

1.2.5. Konfiguracja benchmarka

Format zgodny ze specyfikacją umieszczoną pod adresem http://www.sintef.no/Projectweb/TOP/Problems/PDPTW/Li--Lim-benchmark/Documentation/.

Format pliku z benchmarkiem wygląda następująco:

<ilość pojazdów> <ładowność> <prędkość>

<numer klienta> <współrzędna x> <współrzędna y> <wymagana ładowność> <najwcześniejszy czas dostarczenia> <najpóźniejszy czas dostarczenia> <czas obsługi> <odbiór> <dostawa>

Jeśli odbiór wynosi 0, to znaczy, że produkt musi zostać dostarczony do klienta o numerze <dostawa>. Jeśli dostawa wynosi 0, to znaczy, że produkt musi zostać odebrany od klienta o numerze <odbiór>.

Druga linia w pliku mówi o bazie i całkowitym przewidzianym czasie. Niezerowymi wartościami są: <współrzędna x>, <współrzędna y> i <najpóźniejszy czas dostarczenia>. Zlecenia numerowane są od 1. Dwie pierwsze linie pliku służą jedynie konfiguracji.

Przykładowo:

25	200	1						
0	40	50	0	0	1236	0	0	0
1	45	68	-10	912	967	90	11	0
2	45	70	-20	825	870	90	6	0
3	42	66	10	65	146	90	0	75

1.2.6. Konfiguracja funkcji kosztu

Funkcję kosztu w wyżej wymienionych plikach konfiguracyjnych podajemy jako string. Dopuszczalne są dowolne operacje arytmetyczne. Można używać dowolnych stałych liczbowych. Dodatkowo w funkcji można wykorzystywać następujące parametry agentów i zleceń:

a) TRUCK
   - power
   - reliability
   - comfort

b) TRAILER
   - mass
   - capacity
   - universality

c) DISTANCE
   - dist

d) COMMISSION
   - load
   - pickUpServiceTime
   - deliveryServiceTime

Przykładowe funkcje kosztu przedstawiono powyżej. Należy pamiętać, że poszczególne składowe są „case sensitive”.

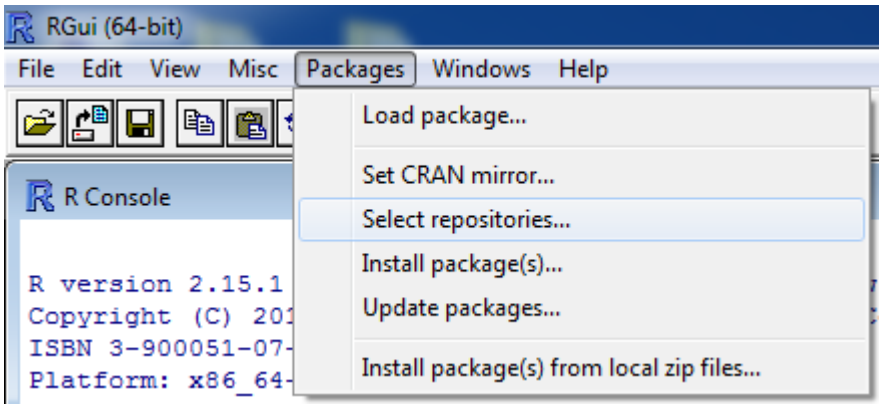
1.3. Instrukcja instalacji

1.3.1. Przygotowanie środowiska pracy

Do pracy większej części systemu wymagana jest tylko wirtualna maszyna Javy, jednakże uczenie maszynowe wykorzystujące klastrowanie wymaga instalacja platformy R do poprawnego działania.

W celu instalacji platformy R należy(instrukcja pod system Windows):

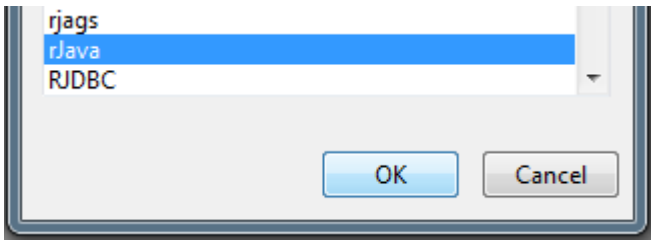
a) Pobrać aktualną wersję platformy R ze strony http://r.meteo.uni.wroc.pl/ -> wybrać „Download R for Windows” -> wybrać „base” -> wybrać link do pobierania.
b) Zainstalować pakiet R -> uruchomić instalator i przejść przez wszystkie jego kroki.
c) Zainstalować plugin rJava:
    a. Uruchomić R w wersji adekwatnej do posiadanego systemu (R – wersja 32 bitowa, R x64 – wersja 64 bitowa).
    b. W menu wybrać *Packages* -> *Select repositories…*, a następnie zaznaczyć w wyświetlonym oknie wszystkie repozytoria.



    c. W menu wybrać *Packages* -> *Install package(s)* (widoczne na rysunku powyżej pod *Select repositories*).

d. W wyświetlonym oknie wybrać CRAN-Mirror (np. „Poland (Wrocław)”).

e. W kolejnym wyświetlonym oknie wybrać *rJava* i zainstalować.



d) W sposób analogiczny do powyższego zainstalować pakiety *fpc* oraz *clue*.

e) Skonfigurować ścieżki systemowe:

a. R_HOME – ścieżka do katalogu, gdzie zainstalowano R, np. D:\Programy\R\R-2.12.2

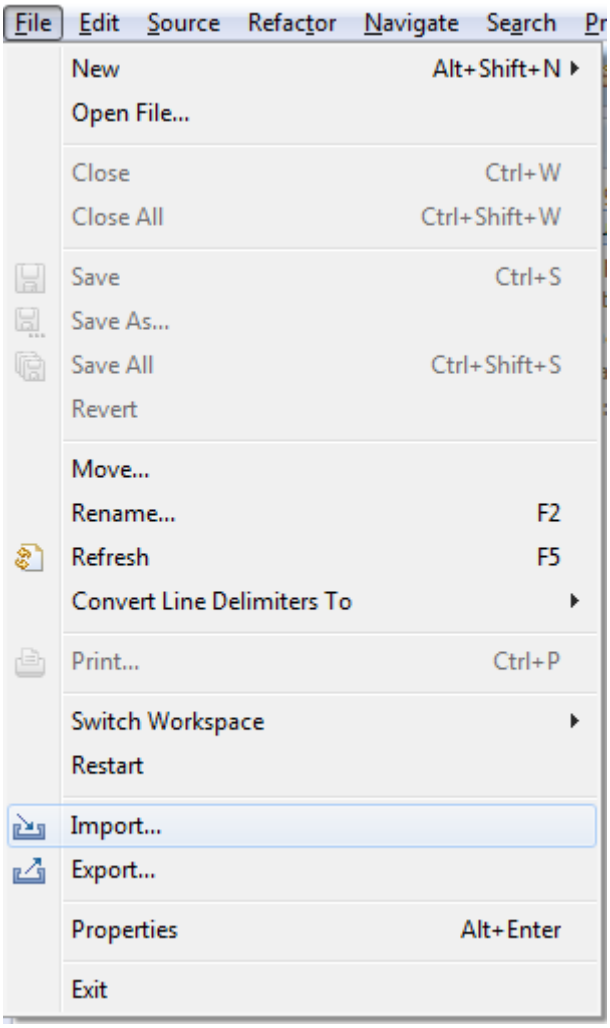
b. LD_LIBRARY_PATH – dodać tutaj %R_HOME%\library\rJava\jri

c. PATH – dodać tutaj %R_HOME%\library\rJava\jri;%R_HOME%\bin;%R_HOME%\bin\x64 (w przypadku systemów 64 bitowych);%R_HOME%\bin\i386(w przypadku systemów 32 bitowych).

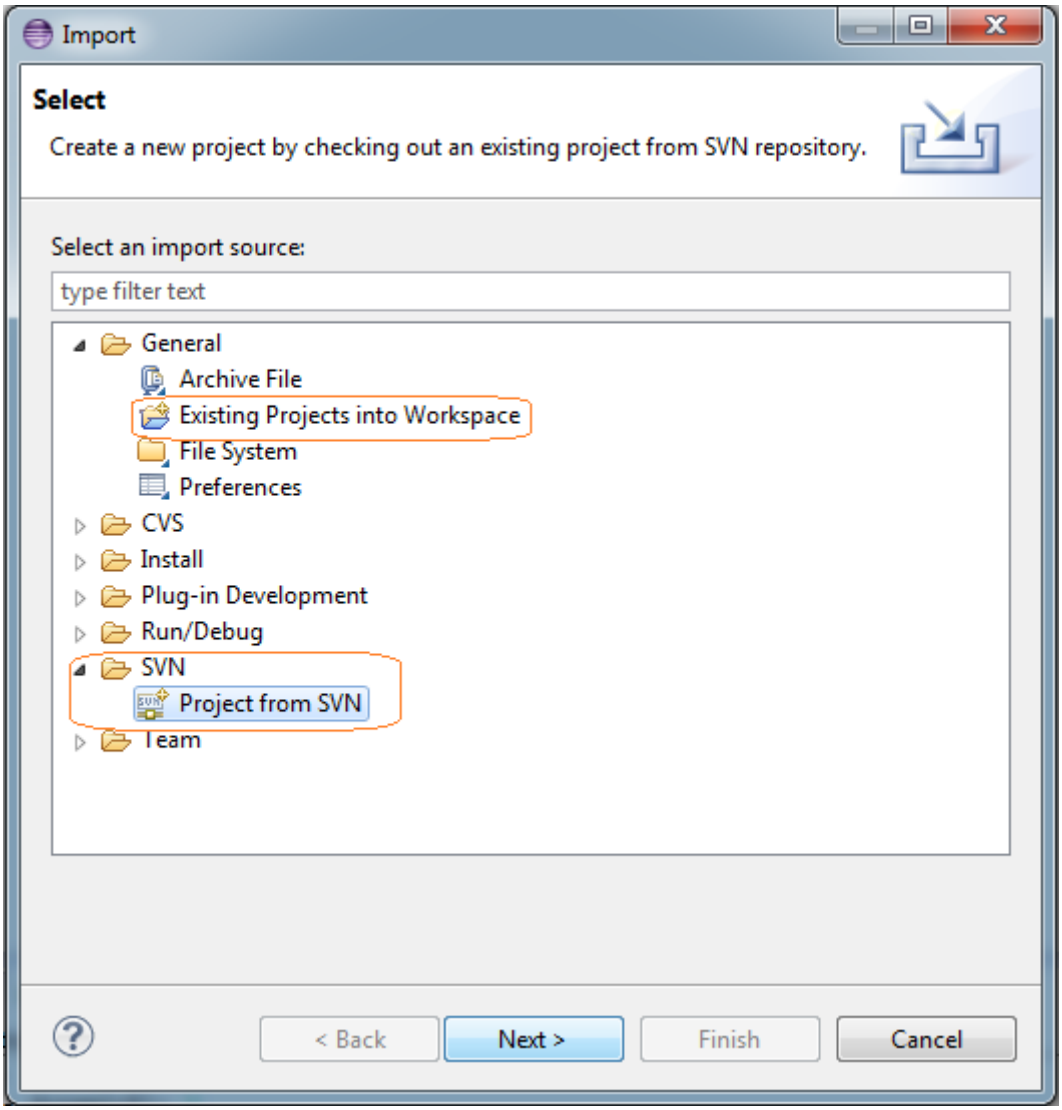
System posiada wszystkie niezbędne do współpracy z platformą R biblioteki, to też ich instalacja jest zbędna.

1.3.2. Instalacja systemu w środowisku Eclipse

Projekt importujemy do Eclipse'a klasycznie używając File -> Import.



Następnie wybieramy źródło projektu. Jest to zależne od osobistej preferencji, jednakże zazwyczaj wybierane są *General -> Existing Project sinto Workspace* lub *SVN -> Project from SVN* (wtyczka Subversive dla Eclipse'a).



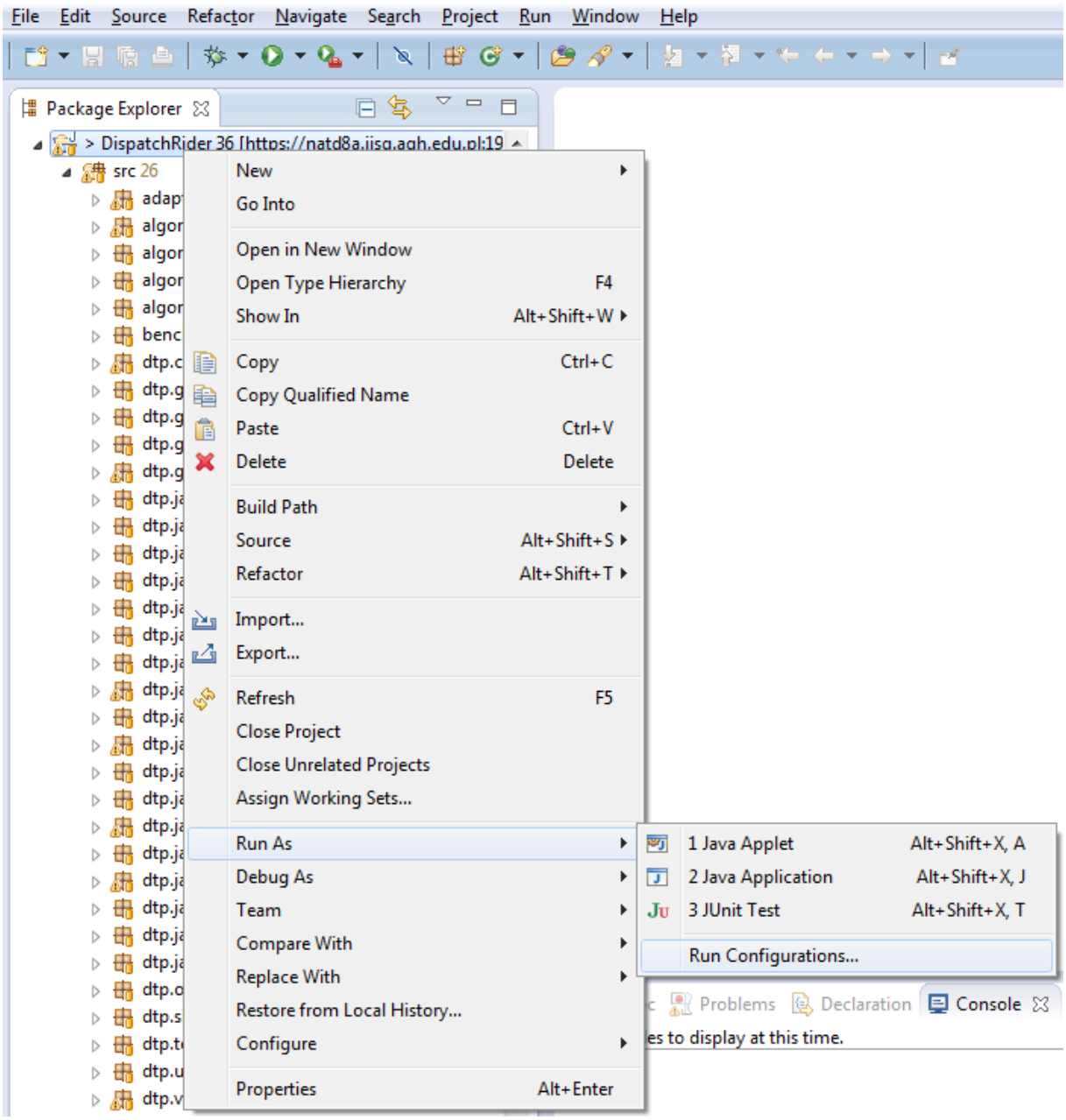
Ponadto należy odpowiednio skonfigurować jego uruchamianie w tymże środowisku, co zostało opisane w następnym podrozdziale.

1.4. Instrukcja uruchomienia

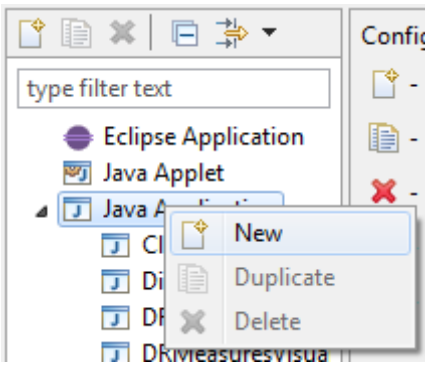
W przypadku problemów związanych z uruchomieniem aplikacji poza środowiskiem Eclipse należy wygenerować *DTP.jar* za pomocą oprogramowania Apache Ant(całe oprogramowanie opisano na http://ant.apache.org/). Plik *build.xml* został już utworzony i znajduje się w katalogu z projektem.

1.4.1. Środowisko Eclipse

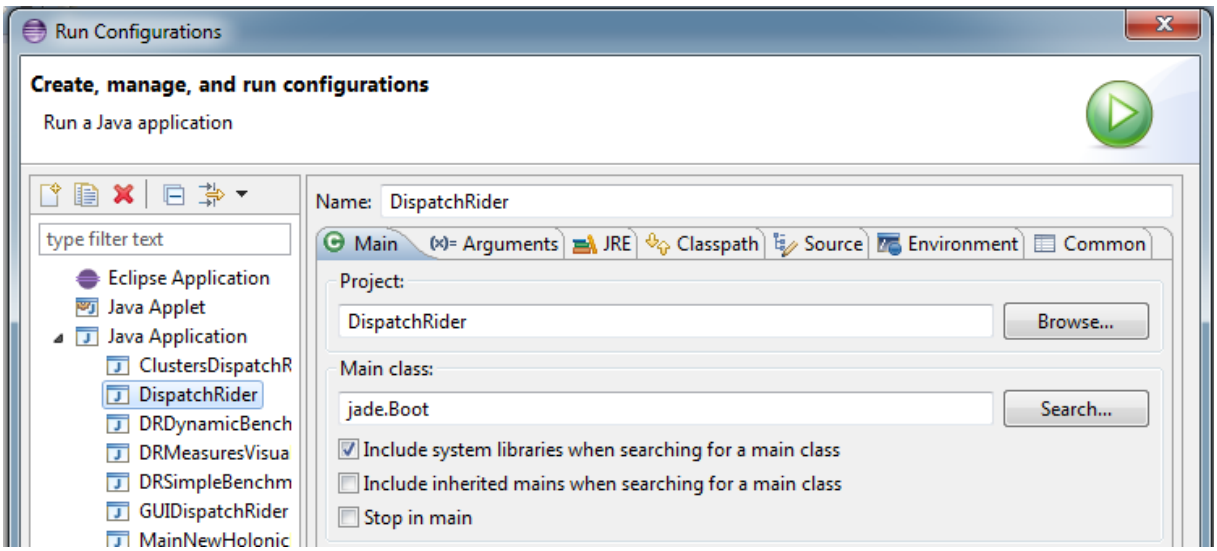
a) Klikamy prawy przyciskiem myszy na projekcie i wybieramy *Run As* -> *Run Configurations*.



b) Wybieramy *Java Application* -> *New*.



c) Ustawiamy dowolną nazwę. Jako *main class* należy podać *jade.Boot*, oraz zaznaczyć opcję *„Include system libraries when searching for a main class”*.



d) Ustawiamy odpowiednie argumenty startowe(zakładka arguments na rysunku powyżej):

```
-nomtp -gui TestAgent:dtp.jade.test.TestAgent
InfoAgent:dtp.jade.info.InfoAgent
DistributorAgent:dtp.jade.distributor.DistributorAgent
CrisisManagerAgent:dtp.jade.crisismanager.CrisisManagerAgent pause
```

lub dla GUI (nierozwijane):

```
-nomtp -gui GUIAgent:dtp.jade.gui.GUIAgentImpl
InfoAgent:dtp.jade.info.InfoAgent
DistributorAgent:dtp.jade.distributor.DistributorAgent
CrisisManagerAgent:dtp.jade.crisismanager.CrisisManagerAgent pause
```

e) Uruchamiamy aplikację. Poprawnie załadowana aplikacja powinna wyświetlić okno wyboru pliku konfiguracyjnego. Po jego wyborze rozpocznie się symulacja.

1.4.2. Uruchamianie poza środowiskiem Eclipse

a) Uruchamiamy program poprzez plik *DispatchRider.bat* lub *DispatchRiderGUI.bat*(nierozwijane).

1.4.3. Benchmark generator

Uruchamiany poprzez plik *benchmarkGenerator.bat*. Służy do wygenerowania benchmarku na podstawie wskazanych danych. W przypadku danych z systemu KrakSim

należy je uprzednio przekonwertować za pomocą skryptów opisanych w punktach 1.4.7 i 1.4.8.

Po uruchomieniu należy podać następujące parametry:

a) Plik z opisem grafu – w tym miejscu podajemy plik z grafem zgodny z formatem dla DispatchRidera
b) Ilość zleceń – musimy podać ile zleceń ma mieć nasz benchmark. Podana liczba zleceń dotyczy par pickup-delivery, dlatego jeśli podamy wartość 1, to w benchmarku będziemy mieć 2 zlecenia, jedno pickup i jedno delivery. Liczba zleceń może wynosić co najwyżej tyle, ile mamy węzłów w grafie
c) Ilość holonów – parametr określa, iloma holonami chcemy obsłuŜyć zlecenia. Generator zapewnia, że na pewno będzie dało się obsłużyć wszystkie zlecenia przez tą ilość holonów (może się jednak zdarzyć, że będzie istnieć rozwiązanie, w którym będzie można użyć mniej holonów). Liczba holonów może być co najwyżej taka jak podana ilość zleceń (par)
d) Minimalna długość okna czasowego
e) Maksymalna długość okna czasowego
f) Masa przewożonych towarów – masa każdego zlecenia będzie równa podanej wartości
g) Opóźnienie w realizacji – maksymalny czas, po którym holon musi wyjechać z bazy, żeby zdążyć obsłużyć zlecenia
h) Service time
i) Gdzie zapisać wynik – ścieżka do pliku, gdzie chcemy zapisać wygenerowany benchmark

Możliwe jest również uruchomienie aplikacji w środowisku Eclipse. W tym celu:

a) Należy utworzyć nowe *Run Configurations*(analogicznie do 1.4.1.).
b) Nadajemy dowolną nazwę dla konfiguracji.
c) Jako *main class* ustawiamy *benchmark.SimpleBenhmarkGenerator*
d) Zaznaczamy opcję *„Include system libraries when searching for a main class”*.
e) Nie potrzeba ustawiać żadnych dodatkowych argumentów.

1.4.4. Dynamic benchmarks generator

Uruchamiany poprzez plik *dynamicBenchmarksGenerator.bat*.

Możliwe jest również uruchomienie aplikacji w środowisku Eclipse. W tym celu:

a) Należy utworzyć nowe *Run Configurations*(analogicznie do 1.4.1.).
b) Nadajemy dowolną nazwę dla konfiguracji.
c) Jako *main class* ustawiamy *translator.Translator*
d) Zaznaczamy opcję *„Include system libraries when searching for a main class”*.
e) Nie potrzeba ustawiać żadnych dodatkowych argumentów.

1.4.5. Graph changes generator

Uruchamiany poprzez plik *graphChangesGenerator.bat*. Generuje plik z zaburzeniami(opóźnieniami) w grafie.

Po uruchomieniu należy:

a) Wybrać plik z opisem grafu – wybieramy odpowiedniego xml'a z opisem grafu, dla którego chcemy wygenerować plik z opóźnieniami
b) Wybrać plik z opiem zleceń, dla którego został wygenerowany graf
c) Podać ilość wirtualnych pojazdów – ustawia wartość zmiennej veicleNr

(W tym miejscu zostanie wyświetlony średni czas przejazdu w grafie i odchylenie standardowe tego czasu.)

d) Podać minimalny wzrost czasu przejazdu (o ile jednostek) – ustawiamy parameter min-Speed
e) Podać maksymalny wzrost czasuorzejazdu (o ile jednostek) – ustawiamy parameter maxSpeed
f) Podać, co ile timestampów zapisywać zmiany grafu – ustawia zmienną updateFreq
g) Wybrać, gdzie zapisać wynik

Możliwe jest również uruchomienie aplikacji w środowisku Eclipse. W tym celu:

a) Należy utworzyć nowe *Run Configurations*(analogicznie do 1.4.1.).
b) Nadajemy dowolną nazwę dla konfiguracji.
c) Jako *main class* ustawiamy *dtp.graph.GraphChangesGenerator*
d) Zaznaczamy opcję *„Include system libraries when searching for a main class"*.
e) Nie potrzeba ustawiać żadnych dodatkowych argumentów.

1.4.6. Graph generator

Uruchamiany poprzez plik *graphGenerator.bat*. Służy do wygenerowania grafu. Generacja trwa tak długo (tyle razy), aż powstanie graf spójny. Pod koniec generacji wyświetlany jest podgląd grafu, a wynik zapisywany jest do pliku *graph.xml*(jest to ustawione w kodzie).

Dostępne są dwa tryby generacji:

- *generateWithRandom* – gdzie połączenia między punktami są całkowicie losowe. Parametrem jest tutaj ilość połączeń wychodząca z pojedynczego węzła grafu.
- *generateWithNeighbours* – tutaj graf też jest generowany losowo. Jako parametry podajemy:
    - ilość sąsiadów – określa, z iloma sąsiadami(określanymi na podstawie odległości od węzła) może wystąpić połączenie

- ilość połączeń – określa, ile połączeń można wykonać z wybranymi wcześniej sąsiadami

Możliwe jest również uruchomienie aplikacji w środowisku Eclipse. W tym celu:

a) Należy utworzyć nowe *Run Configurations*(analogicznie do 1.4.1.).
b) Nadajemy dowolną nazwę dla konfiguracji.
c) Jako *main class* ustawiamy *dtp.graph.GraphGeneratorTest*
d) Zaznaczamy opcję *„Include system libraries when searching for a main class”*.
e) Nie potrzeba ustawiać żadnych dodatkowych argumentów.

1.4.7. Konwerter zaburzeń grafu z KrakSim do DispatchRider

Uruchamiany poprzez plik *KrakSim2DispatchRider_graphChanges.bat*. Format zmian prędkości w KrakSim'ie nie jest formatem xml. W związku z tym na początku następuje naprawa tego formatu (dodanie elementu głównego, oraz domknięcie niektórych elementów). Może się zdarzyć, że może występować wartość NaN. W tym wypadku zmiana jest pomijana – w grafie nic się nie zmienia. Oprócz tego w pliku zmian KrakSim’a podawane są zmiany prędkości, natomiast w naszym systemie operujemy na czasach przejazdu. Konieczne więc było dokonanie odpowiednich przeliczeń wartości. Przyjęto, że początkowy czas przejazdu w oryginalnym grafie jest równy odległości (czyli, że wyjściowa prędkość wynosi 1). Przy takim założeniu wyliczanie czasu przejazdu jest realizowane za pomocą zależności: oryginalny_czas_przejazdu/nowa_prędkość.

Po uruchomieniu translatora musimy wybrać: plik z opisem grafu KrakSim, plik z opisem zmian KrakSim, podać plik gdzie mają być zapisane skonwertowane zmiany.

Możliwe jest również uruchomienie aplikacji w środowisku Eclipse. W tym celu:

a) Należy utworzyć nowe *Run Configurations*(analogicznie do 1.4.1.).
b) Nadajemy dowolną nazwę dla konfiguracji.
c) Jako *main class* ustawiamy *dtp.graph.translator.TranslateGraphChanges*
d) Zaznaczamy opcję *„Include system libraries when searching for a main class”*.
e) Nie potrzeba ustawiać żadnych dodatkowych argumentów.

1.4.8. Konwerter grafu z KrakSim do DispatchRider

Uruchamiany poprzez plik *KrakSim2DispatchRider_graph.bat*. Konwertuje graf z formatu systemu KrakSim do formatu DispatchRidera. Graf systemu KrakSim jest dość rozbudowany, jeśli chodzi o zawarte w nim dane. Większość z nich (jak np. informacje o skrzyżowaniach, czy pasach ruchu) są dla nas nieistotne. W związku z tym przetwarzane są jedynie dane dotyczące współrzędnych punktów, oraz czasów przejazdów między nimi. Jeśli chodzi o użycie, to na początku musimy wskazać plik z opisem grafu KrakSim, a następnie plik, do którego chcemy zapisać skonwertowany graf.

Możliwe jest również uruchomienie aplikacji w środowisku Eclipse. W tym celu:

a) Należy utworzyć nowe *Run Configurations*(analogicznie do 1.4.1.).
b) Nadajemy dowolną nazwę dla konfiguracji.
c) Jako *main class* ustawiamy *dtp.graph.translator.TranslateGraph*
d) Zaznaczamy opcję *„Include system libraries when searching for a main class”*.
e) Nie potrzeba ustawiać żadnych dodatkowych argumentów.

1.4.9. Wizualizacja miar

Uruchamiana poprzez plik *measureVisualization.bat*. Służy do wizualizacji zapisanych po symulacji miar. Po uruchomieniu należy wskazać zapisany plik z miarami.

Możliwe jest również uruchomienie aplikacji w środowisku Eclipse. W tym celu:

a) Należy utworzyć nowe *Run Configurations*(analogicznie do 1.4.1.).
b) Nadajemy dowolną nazwę dla konfiguracji.
c) Jako *main class* ustawiamy *measure.visualization.MeasuresVisualizationRunner*
d) Zaznaczamy opcję *„Include system libraries when searching for a main class”*.
e) Nie potrzeba ustawiać żadnych dodatkowych argumentów.

1.5. Pliki wyjściowe systemu

W zależności od ustawionych parametrów system generuje pliki o nazwach równych wartościom podanym dla znacznika <result></result>.

Kolejne pliki mają rozszerzenia:

- .xls – zawiera dane o holonach, czasie, przejechanym dystansie, wait time’ie, czasie przejazdu, wartości funkcji kary oraz zrealizowanych zleceniach (ilość) wykonywanych zleceń
- _road.txt – kolejne linie zawierają identyfikatory zleceń realizowanych przez holona
- .xml – zawiera szczegółowe dane o poszczególnych krokach symulacji. Składa się on z następujących elementów:
  - <simTime time="0"> - określa timestamp z którego pochodzą dane
  - <holon creationTime="0" id="2" locationX="40.0" locationY="50.0"> - Informacje o pojedynczym holonie (czas stworzenia, id, oraz bierzące położenie)
  - <parts connector="1"> - dane dotyczące składowych holonu

        - <commissions> - zlecenia holonu – każde z nich jest opisane przez czas dojazdu holonu do punktu gdzie jest realizowane zlecenia, czas odjazdu, oraz nr zlecenia w benchmarku
        - <baseReturns> - zawiera informacje o czasie powrotu holonów do bazy
    - _measures... – zawiera informacje dotyczące wyliczanych miar.
    - Pliki dla machine learningu

1.5.1. Uczenie maszynowe

Pliki generowane są jedynie przy użyciu ML w trybie ksploracji. W tym przypadku generowane są dwa pliki:

- Plik z nową tabelką (dodatkowo stary plik z tabelką jest zastępowany nową wartością, w celu ułatwienia przeprowadzania eksploracji – można skonfigurować wiele testów, które odwołują się do tej samej tabelki (nazwa), przy czy każdy kolejny test będzie korzystał z zawartości zmienionej przez poprzednie testy)
- Log zawierający informacje o wartości funkcji nagrody w przypadku pojawienia się nowego pojazdu i gdy nastąpiła jakaś wymiana zleceń

Na razie plik z logami zawiera informacje o zmianie ilości pojazdów i wymianie zleceń.

To, czy należy dodać wartość do logu jest określane poprzez wartość:

- holonsCount>_holonsCount dla ilości pojazdów
- sum(NumberOFReceivedCommissions)>0 – w przypadku wymian

Dane do logowania są zbierane i zarządzane przez machineLearnig.MLLogger