Akademia Górniczo-Hutnicza im. Stanisława Staszica
w Krakowie

Zaawansowane techniki integracji systemów

Platforma wymiany dóbr pomiędzy podmiotami realizującymi proces produkcji

Krzysztof Mycek Mateusz Rudnicki
Jakub Krawętkowski Jarosław Szczęśniak

11 lipca 2012

Spis treści

1 Cel projektu

Celem projektu jest stworzenie platformy wymiany dóbr między producentami i konsumentami, umożliwiającej między innymi negocjacje i renegocjacje kontraktów. Docelowo przebiegiem wymiany dóbr mają zarządzać algorytmy orkiestracji i choreografii.

2 Realizacja

2.1 Wstęp

W ramach projektu stworzyliśmy platformę systemu wymiany dóbr. W obrębie systemu udało nam się zaimplementować kluczowe klasy i funkcjonalności, a także dokonać integracji technologii odpowiedzialnych za sterowanie i wspomaganie procesu wymiany dóbr – JADE i JMX. Stworzona platforma stanowi doskonałą bazę do rozszerzania i dodania algorytmów choreografii i orkiestracji których nie udało nam się zaimplementować w wyniku ograniczeń czasowych.

2.2 Narzędzia

JADE - platforma agentowa wpełni napisana w języku JAVA upraszczająca tworzenie systemów wieloagentowych. Jedna z najbardziej popularnych platform Open Source. Posiada duże zaplecze i wsparcie techniczne [1].

JMX - technologia umożliwiająca zarządzanie oraz monitorowanie aplikacji i usług sieciowych [2].

JADEMX - Środowisko łączące obie technologie, dzięki któremu agenci JADE mogą udostępniać swoje atrybuty i funkcje w sposób zgodny z JMX [3].

Eclipse - wielojęzykowe, zintegrowane środowisko programistyczne [6].

XStream - biblioteka umożliwiająca serializację obietów w języku JAVA [5].

2.3 Model

Na model naszego systemu składają następujące byty rzeczywiste:

Dobro –przedmiot wymiany między producentem a konsumentem.

Przykład:

- drewno (grubość: 20, długość: 120, rodzaj drewna: buk, położenie [5; 18])
- deska (grubość: 2, długość: 100, szerokość: 10, rodzaj drewna: buk, położenie: [10;30])

Typ dobra - określa zestaw cech opisujących dobro.

Przykład:

- drewno (cechy: grubość, długość, rodzaj drewna, położenie)
- deska (cechy: grubość, długość, szerokość, rodzaj drewna, położenie)

Producent – produkuje dobro określonego typu i o określonych cechach.

Przykład – Drwal:

- Wymaga: nic
- Produkuje: drewno 1 szt.
- lista cech produkowanego dobra do negocjacji: grubość, długość, rodzaj drewna.

Konsument - konsumuje dobro, które wcześniej zamówił u producenta.

Przykład – Tartak:

- wymaga: drewno 1 szt.,
- produkuje: deska n szt.,
- lista cech produkowanego dobra do negocjacji: grubość, długość, szerokość, rodzaj drewna, liczba sztuk

Kontrakt - zawierany między producentem a odbiorcą dobra. Przed jego zawarciem odbywa się negocjacja jego warunków (czyli wartości cech dóbr i dat ich dostarczenia)

Negocjacja - podlegają jej wartości cech dóbr oraz daty ich dostaw lub gotowości do odbioru. Konsument prowadzi negocjacje z wieloma producentami jednocześnie. Szansę na wygranie negocjacji ma ten producent, który wystawi najlepszą ofertę.

Renegocjacja - zmiana dat lub wartości cech dóbr ustalonych w kontrakcie za porozumieniem obu stron. Może odbyć się z inicjatywy klienta lub producenta.

Reputacja - każdy konsument po pomyślnej realizacji kontraktu wystawia producentowi ocenę pozytywną. W przypadku niewywiązania się producenta ze zobowiązania, konsument wystawia producentowi ocenę negatywną. Reputacja jest wykorzystywana jako składowa funkcji użyteczności podczas fazy negocjacji.

Funkcja użyteczności – jedna z kluczowych cech systemu, odpowiada za wybór producenta, z którym klient chce podpisać kontrakt, na podstawie oferty i reputacji producenta.

2.4 Aktorzy systemu

W poprzednim rozdziale zostały przedstawione obiekty, z których składa się system. W tym rozdziale przedstawimy kto i w jaki sposób z nich korzysta.

Klient w naszym systemie jest to jednostka, która odpytuje rejestr usług w celu pobrania listy Producentów, którzy spełniają jego wymagania co do usług, które oferują. Odpowiada tworzenie i wysyłanie zleceń do centralnego agenta Rejestru Usług lub bezpośrednio do odpowiednich Producentów celem rozpoczęcia negocjacji do uzyskania wymaganego dobra. Klienci przechowują również historię wszystkich etapów negocjacji oraz wystawiają oceny Producentom, z którymi negocjowali, które są przechowywane w Rejestrze Usług.

- (ClientAgent) Klienci:
  - Odpytywanie rejestru usług
  - Tworzenie dokumentu intencji
  - Odbieranie ofert
  - Historia negocjacji/renegocjacji
  - Dostarczanie dóbr
  - Wystawianie oceny producentowi (pozytywna, negatywna)

Producent jest to tak na prawdę specyficzny rodzaj Klienta, który może również produkować dobra. Posiada te same cechy co Klient, poza tym wypełniają rejestr usług, produkują i magazynują dobra.

- (ProducerAgent) Producenci:
  - Wypełnienie rejestru usług
  - Magazyn dóbr
  - Odpowiedzi na oferty
  - Produkcja dóbr
  - Tworzenie dokumentów intencji dla producentów podwykonawców

Rejestr Usług jest to agent przechowujący dostępne obiekty globalne (typy dóbr i atrybutów), listę zarejestrowanych producentów oraz typy usług jakie oni oferują, a także wartość wskaźników reputacji producentów. Funkcją rejestru usług jest rejestracja, przechowywanie i wyrejestrowanie usług, zwracanie listy zarejestrowanych producentów określonego dobra oraz obliczanie wartości reputacji producentów na podstawie otrzymanych ocen.

- (ServiceRegistry) Rejestr usług:
  - rejestracja usługi
  - wyrejestrowanie usługi
  - zwracanie listy zarejestrowanych producentów dla określonego dobra
  - obliczanie wartości reputacji producentów
  - przechowuje obiekty globalne (typy dóbr i atrybutów)
  - przechowuje listę zarejestrowanych producentów:
  - typ usługi jaki oferują
  - przechowuje wartość wskaźników reputacji producentów

2.5 Dynamika systemu

Klienci (a także w pewnych przypadkach Producenci) odpytują Rejestr Usług w celu pobrania listy Producentów, którzy pełnią odpowiednie usługi na podstawie określonego dobra. Realizacja usługi polega na podpisaniu kontraktu i sfinalizowaniu go przez Producenta lub szereg Producentów, tworząc finalne, wymagane przez Klienta dobro.

Realizacja kontraktu odbywa się w kilku fazach:

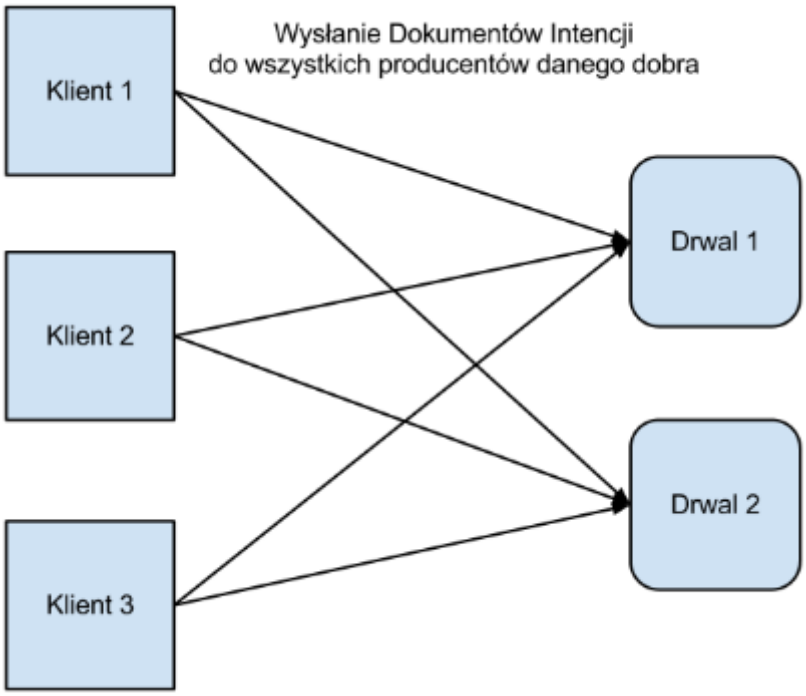
1. Faza negocjacji – kończy się zawarciem kontraktu (lub rezygnacją),
2. Faza oczekiwania – trwa od momentu zawarcia kontraktu do momentu dostarczenia do producenta pierwszego dobra wejściowego (czyli zanim ruszy produkcja).
3. Faza realizacji – trwa od rozpoczęcia produkcji dobra wyjściowego do jego zakończenia i umieszczenia w magazynie.
4. Faza rozliczenia – trwa od momentu umieszczenia dobra wyjściowego w magazynie do odebrania go przez zamawiającego.

W fazie oczekiwania i fazie realizacji może odbyć się renegocjacja kontraktu (patrz: Renegocjacje). W fazie oczekiwania możliwe jest zerwanie kontraktu bez ponoszenia dodatkowych kosztów (prócz utraty punktów reputacji). W fazie realizacji zerwanie kontraktu skutkuje niemożliwością odzyskania dóbr skonsumowanych do produkcji.

Negocjacjom podlegają wartości cech dóbr oraz daty ich dostaw lub gotowości do odbioru. Konsument prowadzi negocjacje z wieloma producentami jednocześnie. Szansę na wygranie negocjacji ma ten producent, który wystawi najlepszą ofertę. Wartość oferty wyznaczana jest za pomocą funkcji użyteczności oferty, która jest indywidualną funkcją, dla każdego konsumenta. Funkcja użyteczności uwzględnia wartości cech dóbr wejściowych, wartości cech dóbr wyjściowych, daty ich dostarczenia oraz reputację producenta, który złożył ofertę.

Renegocjacja jest to zmiana dat lub wartości cech dóbr ustalonych w kontrakcie za porozumieniem obu stron. Renegocjacja może się odbyć z inicjatywy klienta (gdy klient chce zmienić zamówienie) lub producenta (jeżeli pojawiły się problemy z produkcją). Podczas gdy konsument renegocjuje kontrakt z jednym producentem, równolegle może przeprowadzać negocjacje z innymi producentami (z konkurencją). Jeżeli wartość oferty konkurencji okaże się większa od wartości nowej oferty obecnego producenta powiększonej o wartość kosztów zerwania kontraktu, to w toku renegocjacji kontrakt może zostać zerwany i podpisany z nowym producentem.

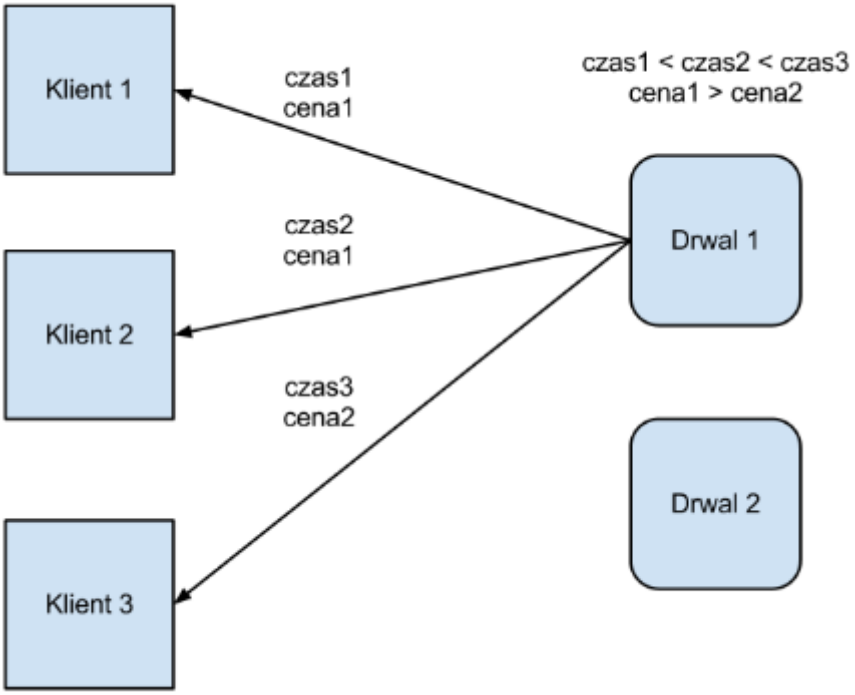
Jedną z ważniejszych cech systemu, jest wsparcie dla jednoczesnych negocjacji z wieloma producentami. Na etapie negocjacji klienci wysyłają Dokumenty Intencji do wszystkich producentów dobra danego typu (Rysunek 1).



Rysunek 1: Wysłanie dokumentów intencji do wszystkich producentów danego dobra

Określają w nich typy i ilości dóbr, które dostarczają oraz jakiego typu dobra oraz cech oczekują w zamian (Rysunek 2).

Należy pamiętać, że producenci rezerwują pewien czas na realizację zamówienia dla poszczególnych klientów, stąd też oferowany czas realizacji dla kolejnych klientów

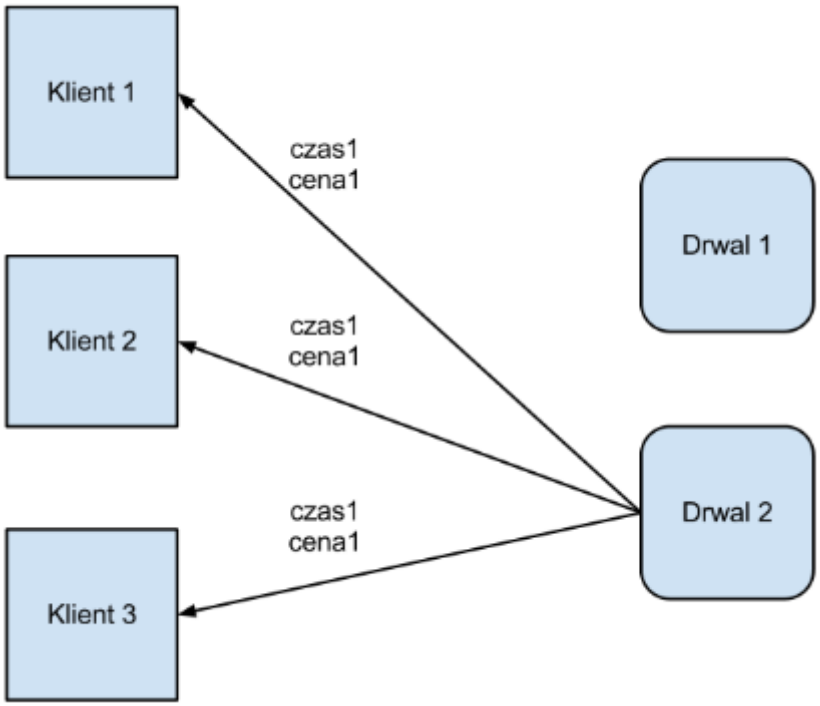


Rysunek 2: W odpowiedzi Producenci wysyłają cechy oferowanych produktów, cenę oraz czas realizacji zamówienia.

może się zwiększać.

Niektórzy producenci nie rezerwują czasu dla poszczególnych klientów, co może powodować problemy w przypadku przyjęcia oferty przez wszystkich klientów (Rysunek 3).

Jeżeli producent nie będzie mógł wywiązać się z realizacji zamówienia, konieczne może być renegocjowanie kontraktu. Może to prowadzić do obniżenia reputacji producenta w przypadku niepowodzenia.



Rysunek 3: Wszyscy klienci przyjmują tą samą ofertę.

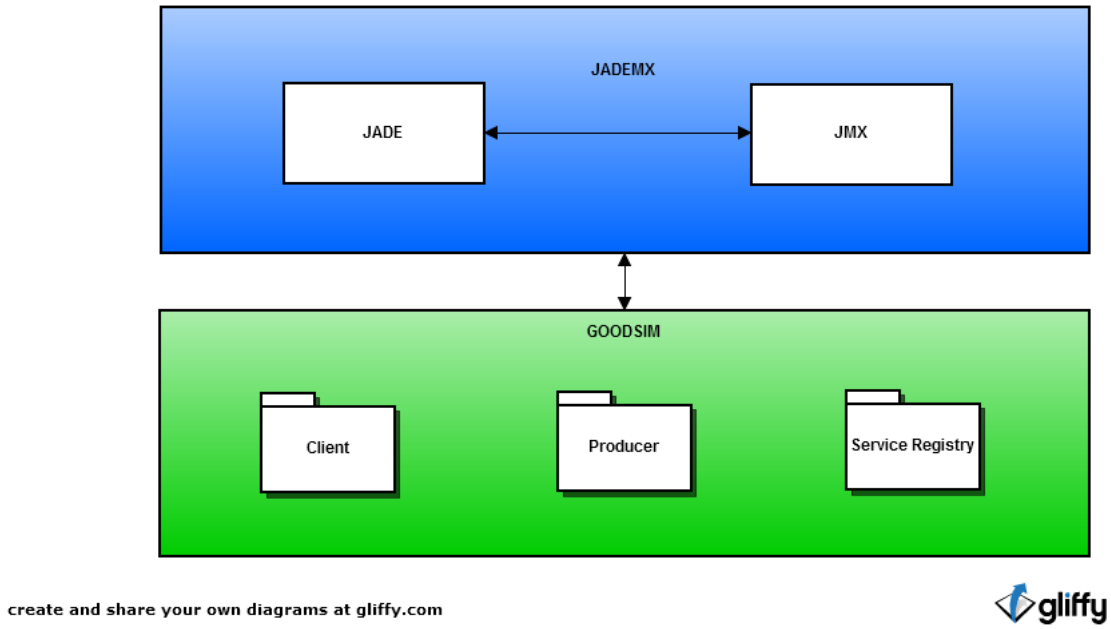
2.6 Implementacja

Projekt został w całości napisany w języku JAVA w środowisku Eclipse i został podzielony na odpowiednie logiczne pakiety, które zostaną opisane w kolejnych podrozdziałach.

Poniżej przedstawiamy schemat architektury systemu (Rysunek 4) oraz pełny diagram klas przedstawiający strukturę i zależności projektu (Rysunek 5).

W związku z koniecznością dużego zmniejszenia diagramu na potrzeby dokumentacji, przy każdym podrozdziale znajduje się tekstowe wylistowanie zaimplementowanych klas i funkcji wraz z ich opisem.

Nazwy metod dobrane zostały w taki sposób, aby same się opisywały.



Rysunek 4: Schemat architektury systemu

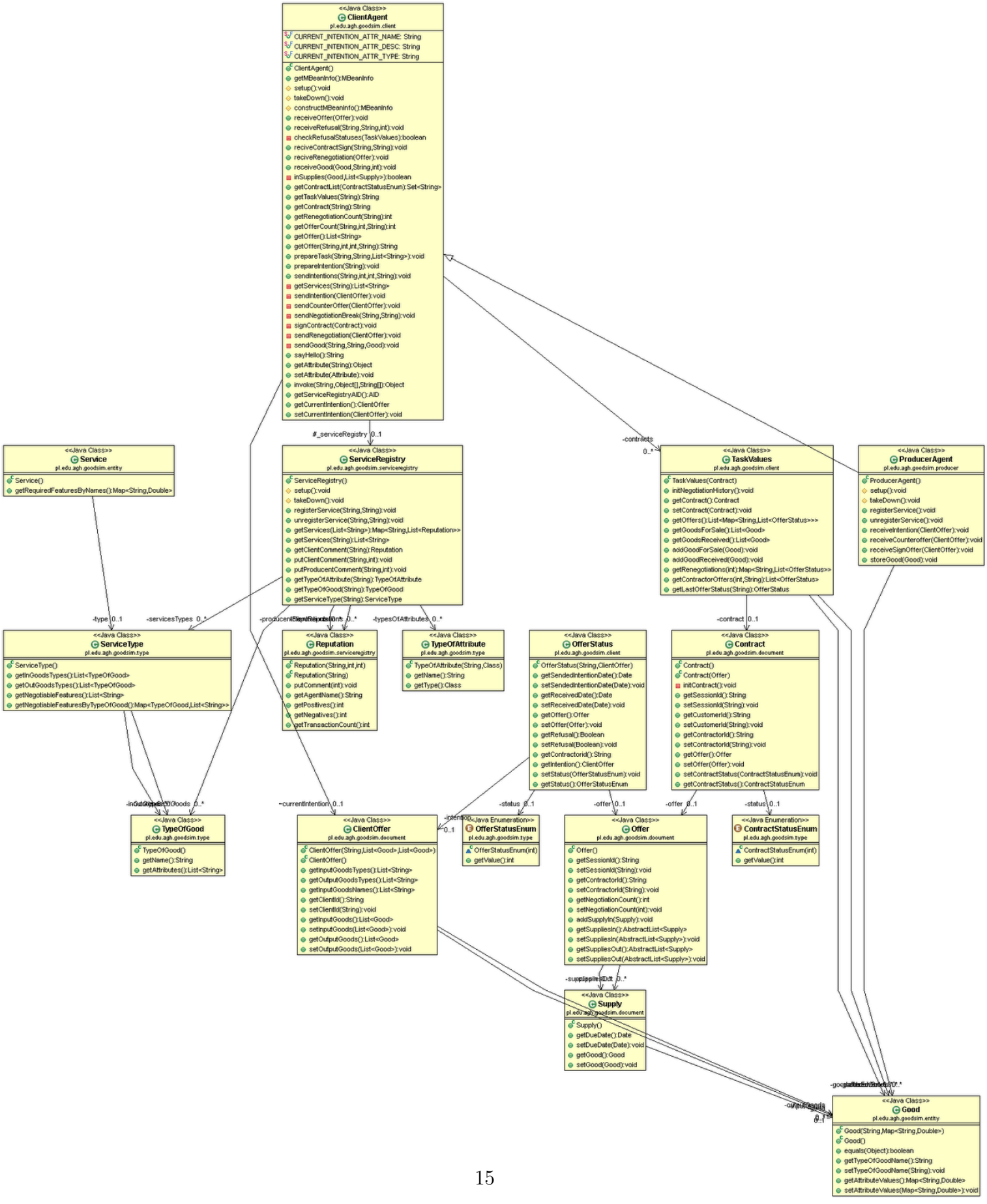
2.6.1 Klient

Pakiet Klienta zawiera przede wszystkim główną klasę Agenta, która implementuje odpowiedni interfejs i metody, aby mogła być użyta w środowisku JADE, a także właściwe funkcje JMX, które po uruchomieniu projektu mogą być wywoływane z konsoli JMX. Oprócz tego składa się również z klas wykorzystywanych przez Klienta do przechowywania kontraktów i historii negocjacji.

ClientAgent – agent klienta, konsument. Szerszy opis funkcji wraz z kontekstem ich wywołań znajduje się w dalszej częsci dokumentacji, na diagramie przebiegu negocjacji.

MBeanInfo getMBeanInfo();
void receiveOffer(Offer offer);
void receiveRefusal(String sessionId, String contractorID, int offerCount);
void reciveContractSign(String sessionID, String contractorID);
void reciveRenegotiation(Offer offer);
void receiveGood(Good good, String sessionID, int container);

Rysunek 5: Diagram klas



Set<String> getContractList(ContractStatusEnum status);
String getTaskValues(String sessionID);
String getContract(String sessionID);
int getRenegotiationCount(String sessionID);
int getOfferCount(String sessionID, int renegotiation, String contractorID);
List<String> getOffer();
String getOffer(String sessionID, int renegotiation, int countOffer, String contractorID);
void prepareTask(String sessionId, String mainSessionId, List<String> goodsTypes);
void prepareIntention(String intention);
void sendIntentions(String sessionId, int renegotiation, int offerNumber, String contractorId);
String sayHello();
Object getAttribute(String attribute) throws AttributeNotFoundException,
MBeanException, ReflectionException;
void setAttribute(Attribute attribute) throws AttributeNotFoundException,
InvalidAttributeValueException, MBeanException, ReflectionException;
Object invoke(String actionName, Object params[], String signature[]) throws MBeanException,
ReflectionException;
AID getServiceRegistryAID();
ClientOffer getCurrentIntention();
void setCurrentIntention(ClientOffer currentIntention);

OfferStatus – Stan oferty:

Date getSendedIntentionDate();
void setSendedIntentionDate(Date sendedIntentionDate);
Date getReceivedDate();
void setReceivedDate(Date receivedDate);
Offer getOffer();
void setOffer(Offer offer);
Boolean getRefusal();
void setRefusal(Boolean refusal);
String getContractorid();
ClientOffer getIntention();
void setStatus(OfferStatusEnum status);
OfferStatusEnum getStatus();

TaskValues – przechowuje cały kontekst dotyczący pojedynczego kontraktu – oferty, negocjacje, renegocjacje, producentów itd.

void initNegotiationHistory();

Contract getContract();
void setContract(Contract contract);
List<Map<String, List<OfferStatus>>> getOffers();
List<Good> getGoodsForSale();
List<Good> getGoodsReceived();
void addGoodForSale(Good goodForSale);
void addGoodReceived(Good goodReceived);
Map<String, List<OfferStatus>> getRenegotiations(int renegotiation);
List<OfferStatus> getContractorOffers(int renegotiation, String contractorID);
OfferStatus getLastOfferStatus(String contractorID);

2.6.2 Producent

Ze względu na to, że w niektórych przypadkach Klient może stać się Producentem stworzyliśmy klasę, która dziedziczy po klasie Klienta i rozszerza go o odpowiednie dla Producenta metody.

ProducerAgent – agent producent. Posiada on wszystkie metody należące do agenta konsumentu, oraz poniższe:

void registerService();
void unregisterService();
void receiveIntention(ClientOffer offer);
void receiveCounteroffer(ClientOffer counterOffer);
void receiveSignOffer(ClientOffer signOffer);
void storeGood(Good good);

2.6.3 Rejestr Usług

Rejestr usług składa się z klasy reprezentującej i implementującej agenta JADE oraz klasy, która opakowuje element reputacji w systemie.

ServiceRegistry - rejestr usług:

void registerService(String serviceTypeName, String agentName);
void unregisterService(String serviceTypeName, String agentName);
Map<String, List<Reputation>> getServices(List<String> goodsTypes);

List<String> getServices(String goodTypeName);
Reputation getClientComment(String clientAgentName);
void putClientComment(String clientAgentName, int points);
void putProducentComment(String producentAgentName, int points);
TypeOfAttribute getTypeOfAttribute(String typeOfAttributeName);
TypeOfGood getTypeOfGood(String typeOfGoodName);
ServiceType getServiceType(String serviceTypeName);

Reputation - przechowuje reputacje klienta / producenta:

void putComment(int points);
String getAgentName();
int getPositives();
int getNegatives();
int getTransactionCount();

2.6.4 Encje

Pakiet **entity** składa się z kluczowych encji obiektów, na których agenci operują w systemie.

Good – klasa reprezentująca dobro, dostępne metody:

String getTypeOfGoodName(); – zwraca nazwe typu dobra
void setTypeOfGoodName(String typeOfGoodName); – ustawia nazwe typu dobra
Map<String, Double> getAttributeValues(); – zwraca wartosci atrybutow/cech dobra
void setAttributeValues(Map<String, Double> attributeValues); – ustawia wartosci atrybutow/cech dobra

Service - usługa:

Map<String, Double> getRequiredFeaturesByNames();

2.6.5 Dokumenty

Klasy implementujące dokument intencji, ofertę oraz kontrakt.

ClientOffer – oferta klienta:

List<String> getInputGoodsTypes();
List<String> getOutputGoodsTypes();

List<String> getInputGoodsNames();

String getClientId();

void setClientId(String clientId);

List<Good> getInputGoods();

void setInputGoods(List<Good> inputGoods);

List<Good> getOutputGoods();

void setOutputGoods(List<Good> outputGoods);

Offer – oferta:

String getSessionId();

void setSessionId(String sessionId);

String getContractorId();

void setContractorId(String contractorId);

int getNegotiationCount();

void setNegotiationCount(int negotiationCount);

void addSupplyIn(Supply supplyIn);

AbstractList<Supply> getSuppliesIn();

void setSuppliesIn(AbstractList<Supply> suppliesIn);

AbstractList<Supply> getSuppliesOut();

void setSuppliesOut(AbstractList<Supply> suppliesOut);

Contract – kontrakt:

String getSessionId();

void setSessionId(String sessionId);

String getCustomerId();

void setCustomerId(String customerId);

String getContractorId();

void setContractorId(String contractorId);

Offer getOffer();

void setOffer(Offer offer);

void setContractStatus(ContractStatusEnum status);

Supply - klasa opakowująca dobro wraz z datą jego wytworzenia:

Date getDueDate();

void setDueDate(Date dueDate);

Good getGood();

void setGood(Good good);

2.6.6 Typy

Pakiet zawiera zaimplementowane wszystkie typy usług, dóbr, serwisów, a także statusy negocjacji.

TypeOfGood – reprezentuje typ dobra, dostępne metody:

String getName(); – zwraca nazwe typu dobra

List<String> getAttributes(); – zwraca liste atrybutow typu dobra

TypeOfAttribute – mapuje nazwę atrybutu na jego typ:

String getName();

Class getType();

OfferStatusEnum – status oferty:

```
PREPARING_INTENTION(0x0), // stan poczatkowy (czekamy az przez JMX
                          // zostanie wypelniona intencja)
INTENTIN_READY(0x1), // stan po wprowadzeniu tresci intencji
                     // przez JMX, mozna wyslac do producenta
WAITING_FOR_OFFER(0x2), // stan po wyslaniu intencji do producenta
                        // jak przyjdzie refusal to przejscie do
                        // 0x9
OFFER_RECEIVED(0x3), // stan po odebraniu oferty od producenta
                     // z tego stanu mozna przejsc do:
                     // 0x4, 0x6 lub 0x9
PREPAING_COUNTOFFER(0x4), // czekamy az przez JMX zostanie
                          // wypelniona kontroferta
COUNTOFFER_READY(0x5), // stan po wprowadzeniu tresci
                       // kontroferty przez JMX, mozna wyslac i
                       // wtedy przejscie do 0x2
SIGING(0x6), // stan po wyslaniu checi podpisania, jak
             // zostanie odebrane refusal to przejscie
             // do 0x9
SIGNED(0x7), // stan po odebraniu potwierdzenia
             // podpisania
CANCEL(0x9);
```

ContractStatusEnum - status kontraktu:

getContractStatus();

NEGOTIATION(0x1),

AWAITING(0x2),

REALISATION(0x4),

SETTLEMENT(0x8),

BEFORE_NEGOTIATION(0x10),

CLOSED(0x20),

WAIT_FOR_RESPONSE(0x40);

ServiceType - typ usługi:

List<TypeOfGood> getInGoodsTypes();

List<TypeOfGood> getOutGoodsTypes();

List<String> getNegotiableFeatures();

Map<TypeOfGood, List<String>> getNegotiableFeaturesByTypeOfGood();

2.7 Komunikacja i sterowanie - JADE i JMX

Chcąc sprawdzić działanie całego systemu i/lub modyfikować go w trakcie działania postanowiliśmy zapewnić komunikację z agentami przy pomocy technologii JMX.

System jest domyślnie pisany, aby działać na platformie JADE, która wspomaga budowę systemów wieloagentowych. Niestety przy jej wykorzystaniu nie ma możliwości samodzielnej rejestracji agentów w MBeanServerze, przez co technologia JMX ich „nie widzi".

Aby połączyć oba frameworki zintegrowaliśmy nasz system do współpracy nie tylko z JADE, ale także z JADEMX, który jest środowiskiem, dzięki któremu agenci mogą udostępniać swoje atrybuty i funkcje w sposób zgodny z JMX.

Dzięki JADEMX możemy:

- skonfigurować ilość i atrybuty agentów w pliku XML,
- uruchomić JadeMXServer, dzięki któremu możemy rejestrować Agentów,
- uruchomić JadeFactory, które uruchamia nam agenty skonfigurowane w XML i rejestruje je w JadeMXServer
- uruchomić ostatecznie JadeRuntime - platformę JADE, w której obecne będą nasi agenci.

Ostatecznie mamy system, którego poprawność działania możemy sprawdzać w każdym momencie, jak również możemy wpływać na zachowanie agentów, w celu testowania specyficznych przypadków.

2.8 Cykle życia kontraktu

Poniżej przedstawiamy tabele, które szczegółowo przedstawiają pełny cykl życia kontraktu i jego realizacji wraz z komentarzem do każdego etapu.

Poruszając się od góry do dołu widzimy, który agent jest odpowiedzialny za odpowiedni stan oraz kolejność w jakiej agenci wywołują swoje funkcje.

Rejestr usług - Rejestracja, wyrejestrowanie usługi:

Services Registry	Klient		Producent		Komentarz
	Agent	JMX	Agent	JMX	
					Uruchomienie agenta producenta
			registerService		Agent wysyła do SR chęć zarejestrowania swojej usługi
registerService					Usługa rejestrowana w SR
					Zamykanie agenta producenta
			unregisterService		Agent wysyła do SR chęć wyrejestrowania swojej usługi
unregisterService					Usługa zostaje wyrejestrowana

Klient – Konsument - przebieg realizacji kontraktu przy pełnej obsłudze z konsol JMX:

Services Registry	Stan kontraktu		Klient		Producent		Komentarz
	Główny	Oferty	Agent	JMX	Agent	JMX	
	init	init object		*prepareTask*			Wybranie w konsoli JMX klienta co chcemy otrzymać i wygenerowanie owego sesionID i mainSessionID
			prepareTask				Inicjalizacja obiektu TaskValues
		waiting for services	*getServices*				Pobranie listy usług realizujących zamawiane dobro wraz z reputacją
getServices							
	negotiation	preparing intention					Przygotowano szablony dokumentów intencji, trzeba wypełnić intencje ręcznie przez konsolę JMX
				prepareIntention			Wprowadzenie w konsoli JMX wartości cech intencji
		intention ready					Stan: intencje zostały wypełnione, gotowe do rozesłania
				sendIntention			Zlecenie w konsoli JMX wysłania intencji
			sendIntention		receiveIntention		Wysłanie intencji i odebranie ich przez producenta
		waiting for offer					Czekanie aż producent odpowie ofertą
getClientReputation					*getClientReputation*		Pobranie od rejestru usług reputacji klienta
						prepareOffer	Stan: ofertę trzeba wypełnić ręcznie przez JMX
						sendOffer/sendRefusal	Zlecenie wysłania oferty/zlecenie wysłania odmowy wykonania usługi
		offer received	receiveOffer/receiveRefusal		*sendOffer/sendRefusal*		Wysłanie jw. i odebranie tego przez agenta klienta
		preparing countoffer					Stan: klient otrzymał ofertę i przez JMX ma zdecydować co z nią zrobić
				prepareCounteroffer			W tym przypadku przez JMX jest wprowadzana kontroferta
		counteroffer ready					Stan: kontroferta gotowa do wysłania
				sendCounteroffer			Zlecenie przez JMX wysłania kontroferty
			sendCounteroffer		receiveCounteroffer		Wysłanie kontroferty i odebranie jej przez agenta producenta

		waiting for offer					Oczekiwanie na nową ofertę od producenta
						prepareOffer	Stan: ofertę trzeba wypełnić ręcznie przez JMX
						sendOffer	Zlecenie wysłania oferty/zlecenie wysłania odmowy wykonania usługi
		offer received	receiveOffer		*sendOffer*		Wysłanie oferty i odebranie jej przez klienta
		preparing countoffer					Stan: klient otrzymał ofertę i przez JMX ma zdecydować co z nią zrobić
				singOffer/negotiationBreak			Tym razem przez JMX zlecono podpisanie kontraktu/rezygnację z dalszych negocjacji
		siging offer	*signOffer/negotiationBreak*		receiveSignOffer/receiveBreakNegotiation		Wysłanie zlecenia jw. i odebranie go przez producenta
						singOffer/rejectOffer	Decyzja przez JMX czy producent ma zdecydować się podpisać czy zrezygnować
	Waiting	signed	receiveSignedOffer/receiveRejectOffer		*sendSignedOffer/sendRejectOffer*		Wysłanie jw. i odebranie przez agenta. Jeżeli odmówiono podpisania kontraktu mimo
		normal		*storeGood*			Przekazanie agentowi dobra przez konsole JMX
			storeGood				Przekazanie agentowi dobra można dokonać w każdym momencie kontraktu
				transferGood			Zlecenie przekazania dobra innemu agentowi
			transferGood		storeGood		Przekazanie dobra innemu agentowi (zmiana właściciela)
			receiveStoreGoodACK		*storeGoodACK*		Przesłanie potwierdzenia odebrania zasobu i że jest on zgodny z oczekiwaniami
				sendRenegotiation			Zlecenie przez JMX przeprowadzenie renegocjacji z inicjatywy klienta
		init renegotiation	*sendRenegotiation*		receiveRenegotiation		Wysłanie do producenta kontroferty inicjującej renegocjacje
		waiting for offer					Oczekiwanie na ofertę i przeprowadzenie akcji negocjacji wg takich samych zasad jak podczas
						prepareOffer	
						sendOffer	

		offer received	receiveOffer		sendOffer		
		preparing countoffer					
				prepareCounteroffer			
		counteroffer ready					
				sendCounteroffer			
			sendCounteroffer		receiveCounteroffer		
		waiting for offer					
						prepareOffer	
						sendOffer	negocjacji.
		offer received	receiveOffer		sendOffer		
		preparing countoffer					Równolegle przeprowadza się negocjacje nowego kontraktu z innymi producentami ale to już jest osobny kontrakt (nie ten).
				singOffer/negotiationBreak			
		siging offer	signOffer/negotiationBreak		receiveSignOffer/receiveBreakNegotiation		
						singOffer/rejectOffer	Przerwanie negocjacji skutkuje wystawieniem negatywnego komentarza klientowi.
		signed	receiveSignedOffer		sendSignedOffer		
		normal					
						sendRenegotiation	Renegocjacja z inicjatywy producenta.
		init renegotiation	receiveRenegotiation		sendRenegotiation		Przebiega niemal identycznie jak poprzednia.
		preparing countoffer					
				prepareCounteroffer			
		counteroffer ready					Różnica polega na tym, że tu klient ma prawo wystawić negatywny komentarz w przypadku gdy będzie musiał zerwać kontrakt z tym producentem i podpisać z innym.
				sendCounteroffer			
			sendCounteroffer		receiveCounteroffer		
		waiting for offer					
						prepareOffer	
						sendOffer	
		offer received	receiveOffer		sendOffer		
		preparing countoffer					
				singOffer/negotiationBreak			
		siging offer	signOffer/negotiationBreak		receiveSignOffer/receiveBreakNegotiation		
						singOffer/rejectOffer	
		signed	receiveSignedOffer		sendSignedOffer		

normal
runWorkProcess
Zlecenie rozpoczęcia prac nad dobrem wyjściowym
Working
normal
Prace nad dobrem wyjściowym rozpoczęto
... to samo co w fazie oczekiwania (Waiting) ...
finalWorkProcess
W konsoli JMX informacja, że produkt gotowy do odbioru
Payoff
normal
storeGood
storeGood
Wprowadzanie w JMX parametrów dobra (store do producenta) i przekazanie go klientowi (transfer do klienta)
transferGood
transferGood
storeGood
storeGoodACK
receiveStoreGoodACK
storeGood
storeGood
transferGood
transferGood
receiveStoreGoodACK
Przekazanie przez klienta zapłaty
storeGood
storeGoodACK
putComment
putComment
putProducentComment
Oba podmioty wystawiają sobie po komentarzu. Komentarz można wystawić tylko w fazie finalizacji
putComment
putClientComment
putComment
finalized
Wszystkie procesy związane z kontraktem zostały zakończone

2.9 Instalacja i uruchomienie

Do projektu dołączone są skrypty dla platformy Windows (install.bat, runAgent.bat) oraz Linux (install.sh, runAgent.sh).

install.bat / install.sh

Skrypt ten odpowiedzialny jest za zbudowanie projektu przy użyciu narzędzia Maven2, uwzględniając brak publicznego repozytorium dla archetypów jade oraz jademx. Pliki biblioteki są kopiowane do lokalnego repo, a następnie dołączane poprzez maven do projektu w celu zbudowania kompletnego archiwum .jar wraz ze wszystkimi zależnościami aplikacji.

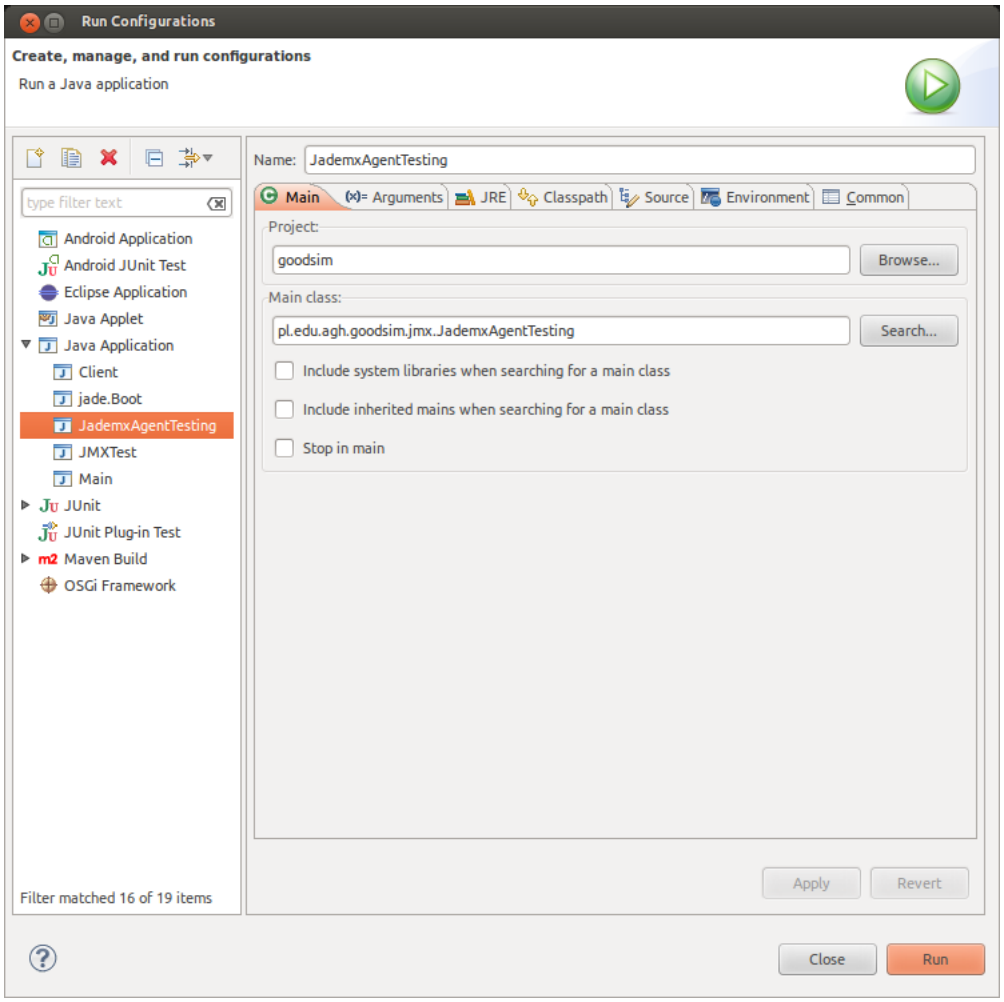
runAgent.bat / runAgent.sh

Skrypt ten jest odpowiedzialny za uruchomienie podstawowej klasy platformy. Tylko poprzez takie uruchomienie, mamy możliwość uruchomienia platformy Jade wraz z rejestracją agentów w JadeMXServerze, który udostępnia metody JMX agentów.

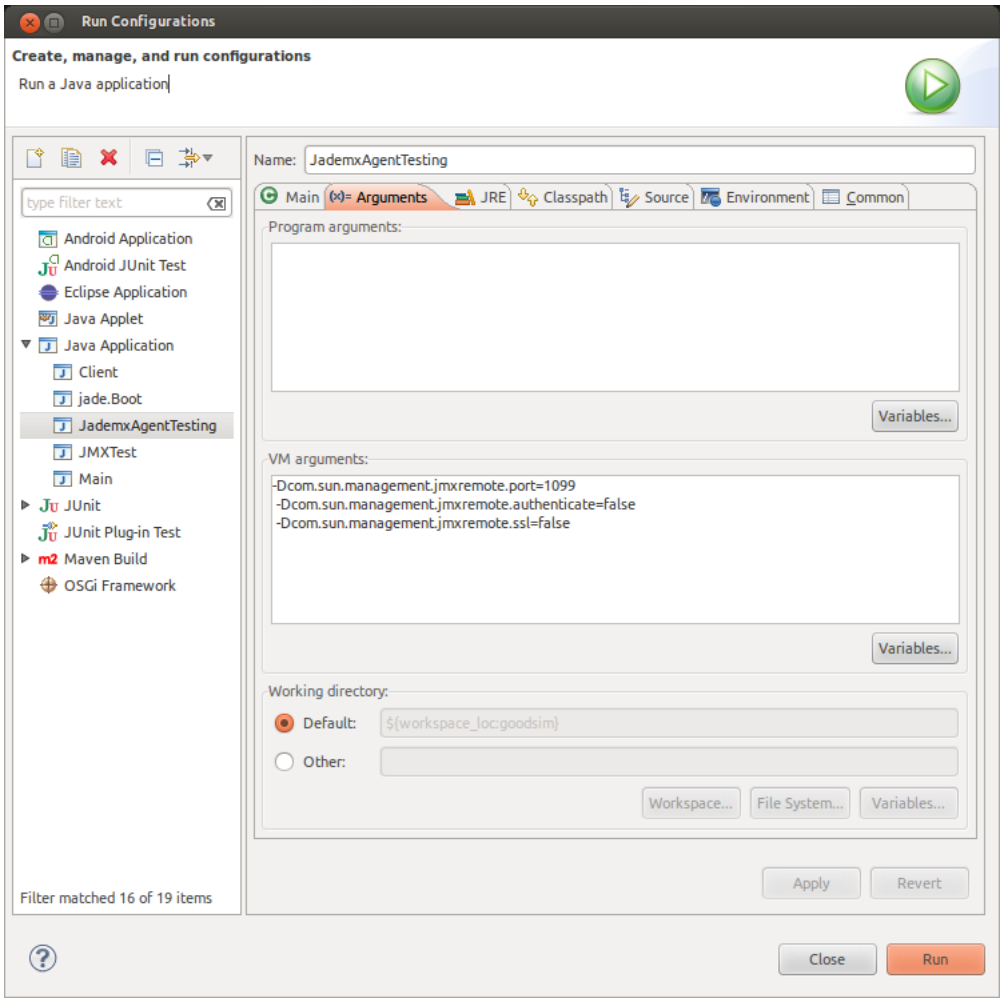
W kolejnym rozdziale opisane zostały kroki uruchomienia platformy wymiany dóbr z poziomu Eclipse.

2.10 Demonstracja

W celu uruchomienia przykładu z poziomu eclipse, należy użyć konfiguracji przedstawionej na rysunku 6 i 7.

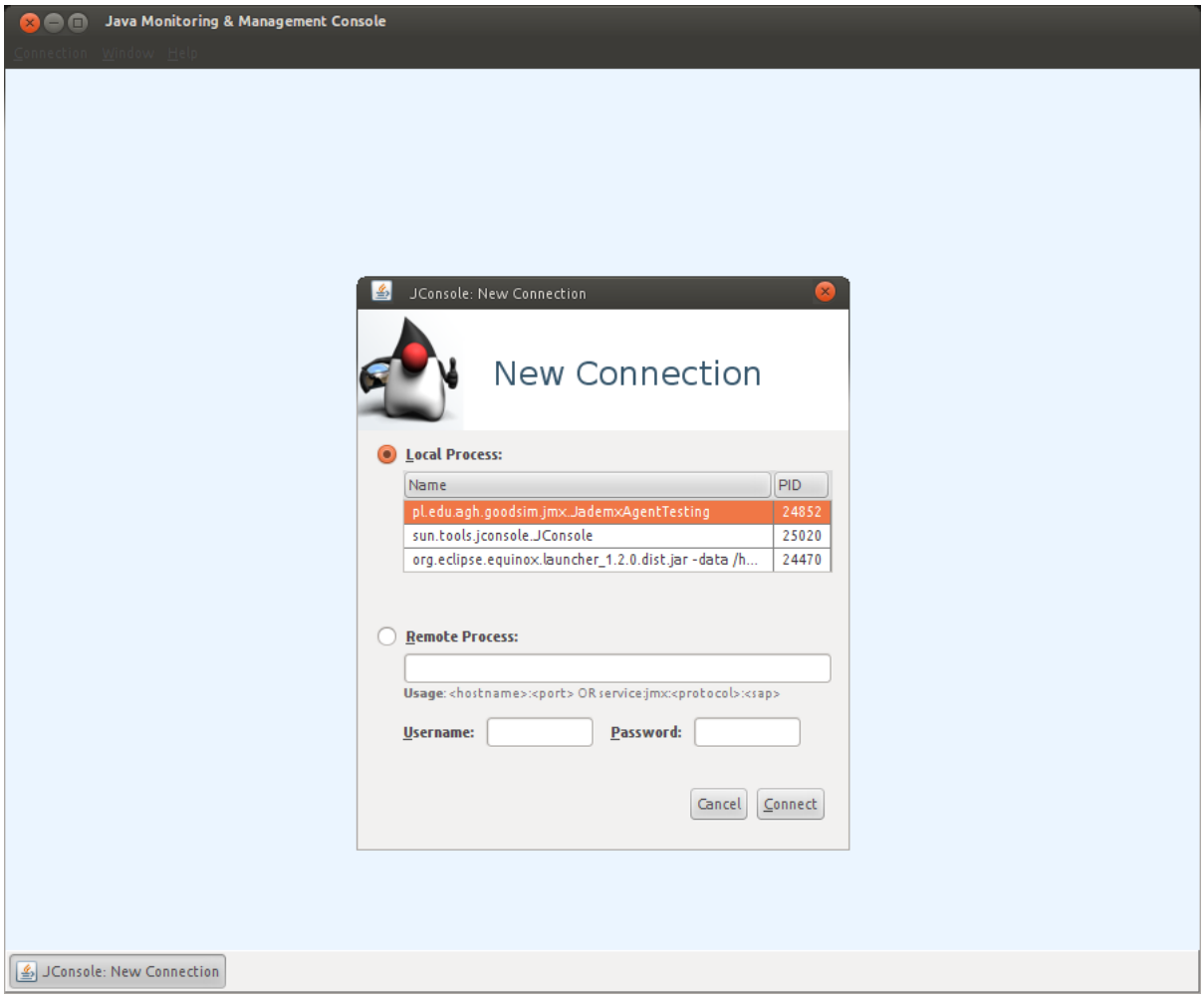


Rysunek 6: Konfiguracja z poziomu eclipse - główne opcje



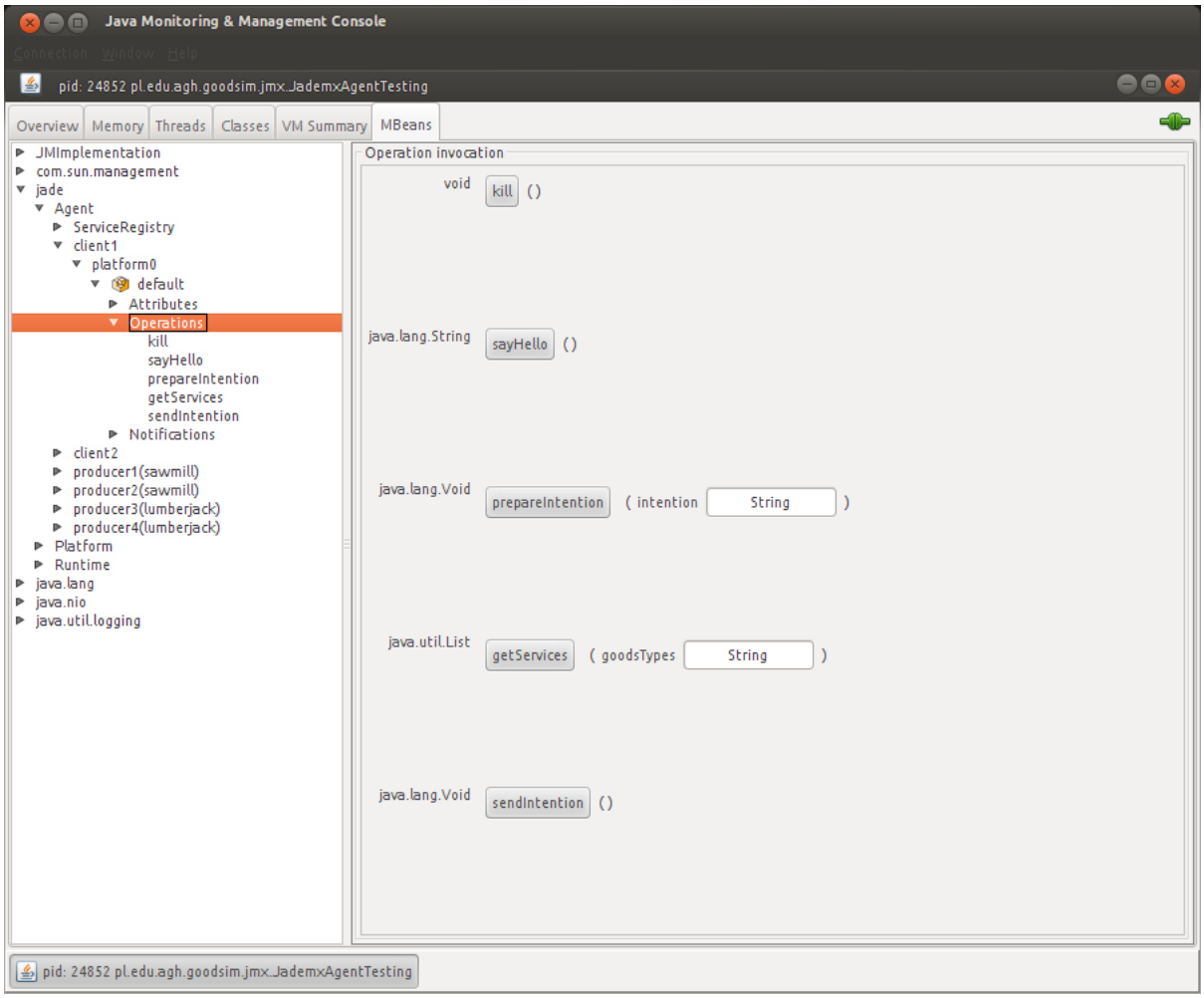
Rysunek 7: Konfiguracja z poziomu eclipse - parametry

Po uruchomieniu aplikacji, możemy się do niej podłączyć poprzez jconsole (Rysunek 8). W uruchomionym oknie, będziemy mieli dostęp zarówno do platformy agentowej jak i rezydujących na niej agentów.



Rysunek 8: Konsola JMX

Wybierając odpowiedniego agenta, mamy możliwość wywołania na nim metody przy pomocy JMX (Rysunek 9).



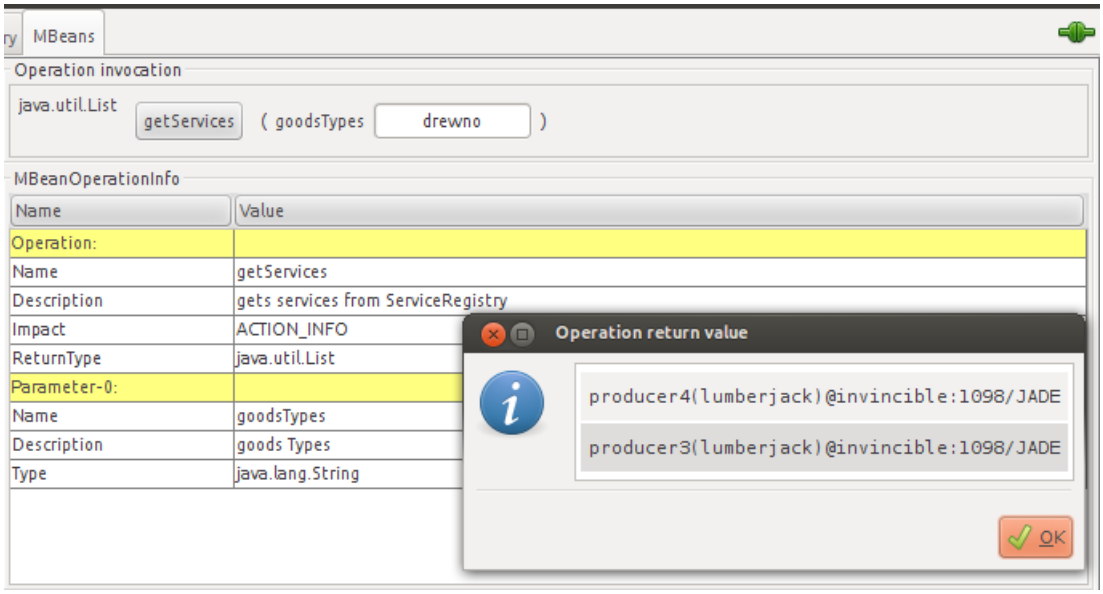
Rysunek 9: Przykładowe wywołanie metody JMX getServices z parametrem „drewno”.

Poniżej log z przykładowego uruchomienia platformy wymiany dóbr, wraz z opisem poszczególnych kroków (Rysunek 11).

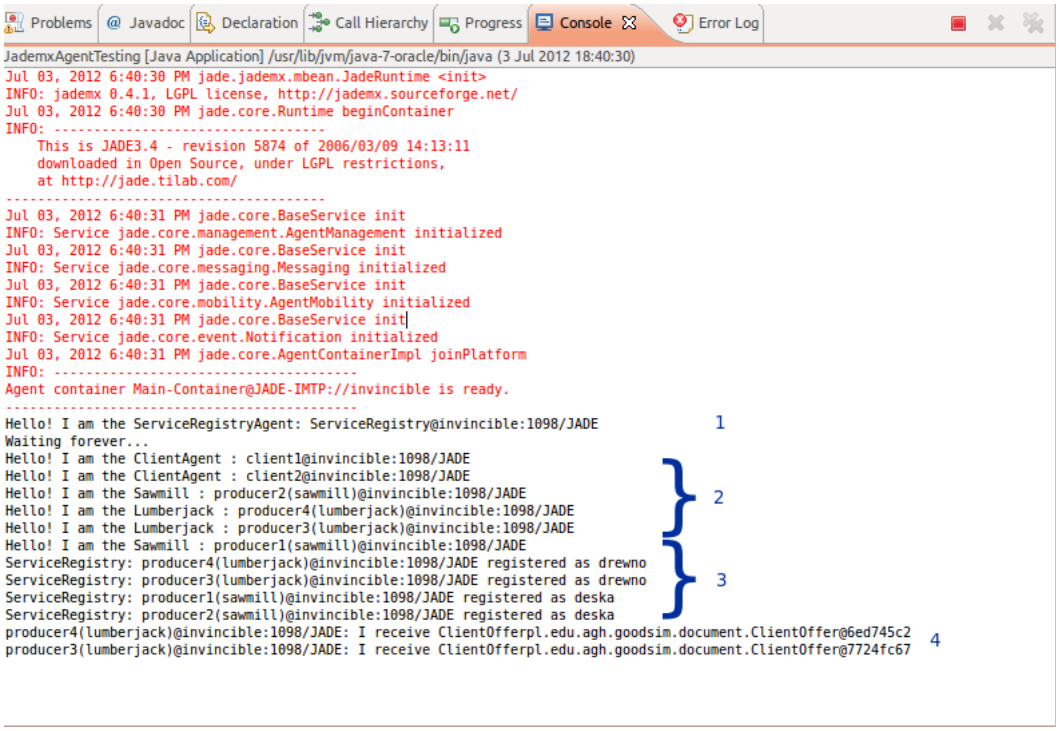
1. Uruchomienie ServiceRegistry
2. Uruchomienie 2 agentów klientów oraz 4 agentów producentów (2x drwal oraz 2x tartak)
3. Rejestracja producentów w ServiceRegistry
4. Komunikat kontrolny od producentów, wywołany w odpowiedzi na odebrany od klienta dokument intencji (ClientOffer)

Punkty 1-3 wywoływane są z poziomu aplikacji JADE.

Punkt 4 jest odpowiedzią na stworzony w kliencie, a następnie rozesłany do producentów dokument intencji. Obie te czynności wykonane są przy użyciu metod JMX, wywołanych z poziomu JConsole.



Rysunek 10: Widzimy odpowiedź serviceAgent w postaci listy agentów producentów oferujących typ dobra drewno.



Rysunek 11: Wycinek logu z uruchomionej aplikacji.

2.11 Podsumowanie i propozycje na kontynuację projektu

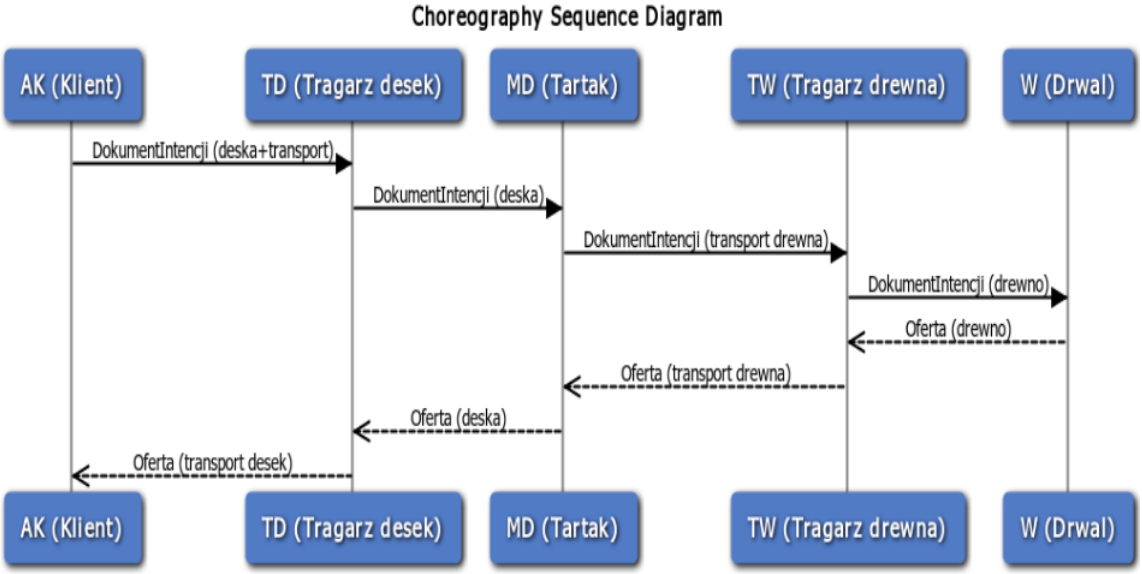
Zrealizowany projekt dostarcza solidny fundament platformy wymiany dóbr, wraz z pełną integracją z technologiami JADE i JMX.

Jako kontynuację projektu, proponujemy rozbudowę istniejącego systemu o:

- pokrycie wszystkich funkcji agentów odpowiednimi funkcjami JADEMX
- implementację algorytmów choreografii i orkiestracji

Implementacja algorytmów choreografii i orkiestracji pozwoli na w pełni dynamiczny przebieg wymiany dóbr wraz z fazami negocjacji i renegocjacji, co powinno dać ciekawe do analizy rezultaty.

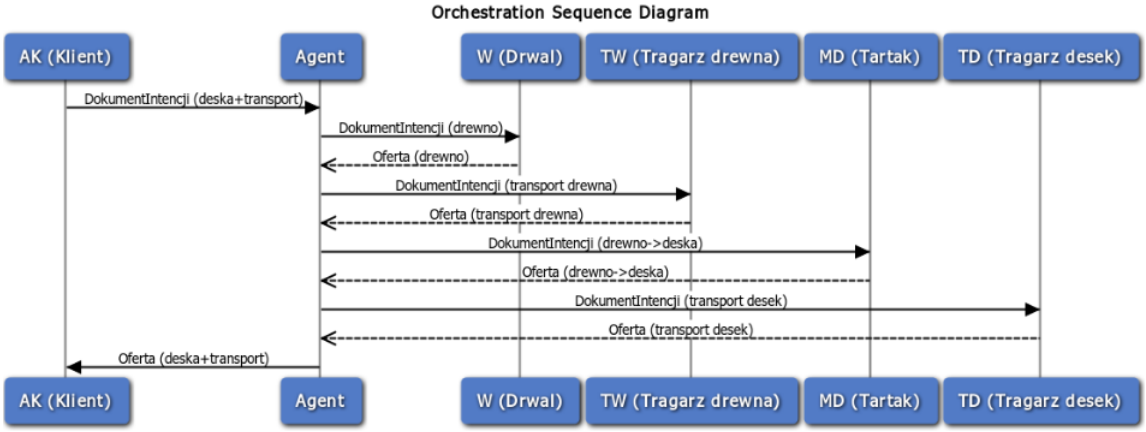
Przykładowy przebieg wymiany dóbr po implementacji algorytmu choreografii mógłby wyglądać następująco (Rysunek 12):



Rysunek 12: Diagram sekwencji algorytmu Choreografii

Algorytm choreografii jest z założenia kooperacyjny. Polega na widocznej globalnie interakcji, która zachodzi pomiędzy wszystkimi elementami systemu. Każda ze stron może być zaangażowana w proces wymiany wiadomości i żadna ze stron nie staje się jego właścicielem.

W przypadku algorytmu orkiestracji sytuacja wyglądałaby w następujący sposób (Rysunek 13):



Rysunek 13: Diagram sekwencji algorytmu Orkiestracji

Zasada działania orkiestracji jest przeciwna do algorytmu choreografii. Wyróżniamy tu jednego, centralnego Agenta, który kontroluje i pośredniczy w wymianie informacji pomiędzy elementami systemu. Orkiestracja może skutkować trwałym, wielokrokowym modelem procesu.

3 Bibliografia

Literatura

[1] JADE - Java Agent DEvelopment Framework – http://jade.tilab.com/

[2] JMX - Java Management Extensions Technology – http://www.oracle.com/technetwork/java/javase/tech/javamanagement-140525.html

[3] JADEMX - JMX access to JADE agents – http://jademx.sourceforge.net/

[4] Apache Maven Project – http://maven.apache.org/

[5] XStream – http://xstream.codehaus.org/

[6] Eclipse – http://www.eclipse.org/