

Zaawansowane Techniki Integracji Systemów

Platforma wymiany dóbr między
podmiotami realizującymi proces
produkcji



AGH

Krzysztof Mycek

Jakub Krawętkowski

Matuesz Rudnicki

Jarosław Szczęśniak

Cel Projektu

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

Realizacja

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.

W kolejnych rozdziałach przedstawiony zostanie opis stworzonego systemu.

2. 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.

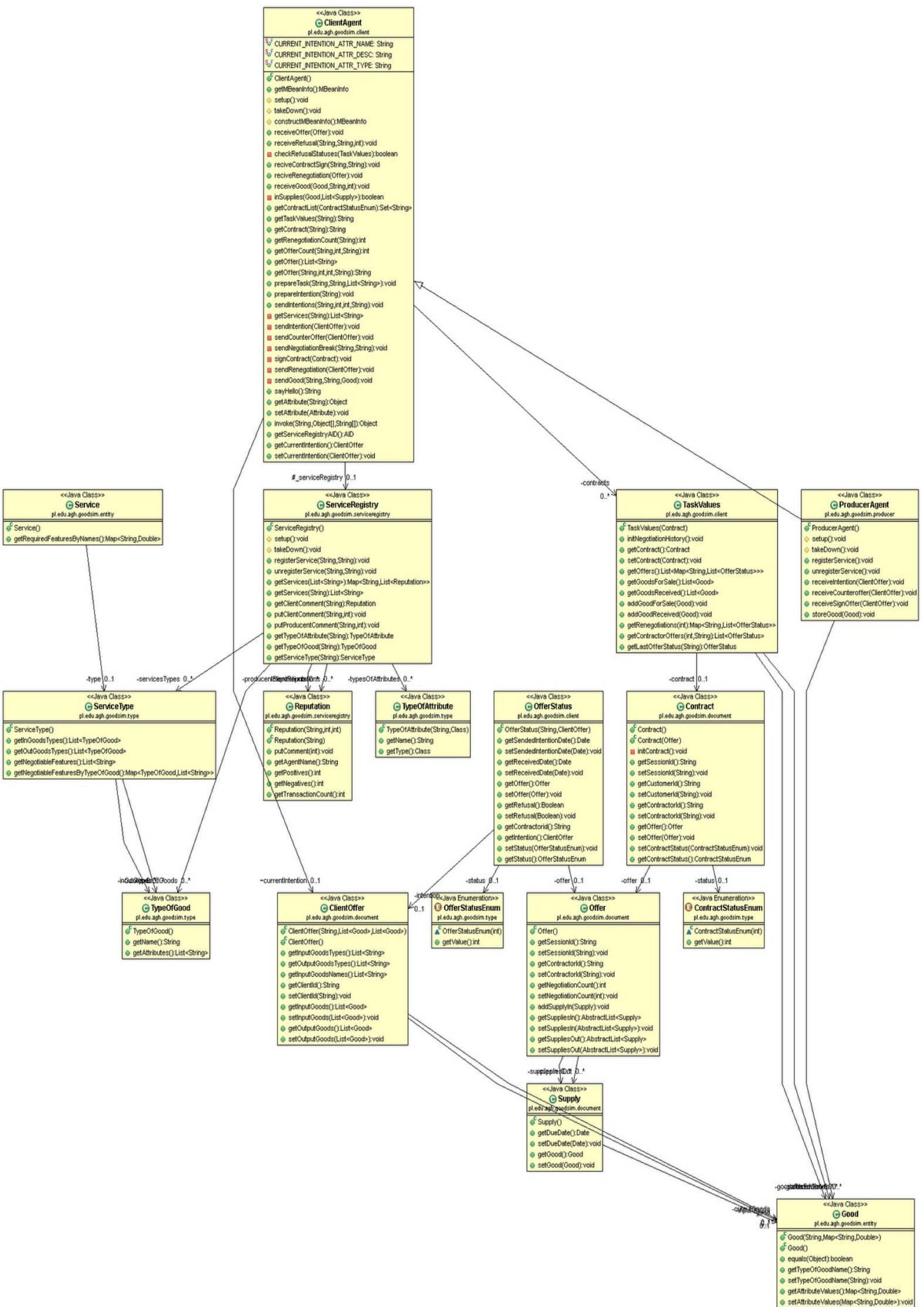
Renegocjacja - zmiana dat lub wartości cech dóbr ustalonych w kontrakcie za porozumieniem obu stron. Może odbyć się z inicjatywy klienta lub producenta.

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.

3. Implementacja

Na kolejnej stronie znajduje się diagram klas zaimplementowanego systemu.



W związku z koniecznością dużego zmniejszenia diagramu na potrzeby dokumentacji, poniżej znajduje się tekstowe wylistywanie zaimplementowanych klas i funkcji, wraz z ich opisem. Nazwy metod dobrane zostały w taki sposób, aby same się opisywały.

Good – klasa reprezentująca dobro, dostępne metody:

String getTypeOfGoodName(); - zwraca nazwę typu dobra

void setTypeOfGoodName(String typeOfGoodName); - ustawia nazwę typu dobra

Map<String, Double> getAttributeValues(); - zwraca wartości atrybutów/cech dobra

void setAttributeValues(Map<String, Double> attributeValues); - ustawia wartości atrybutów/cech dobra.

TypeOfGood – reprezentuje typ dobra, dostępne metody:

String getName(); - zwraca nazwę typu dobra

List<String> getAttributes(); - zwraca listę atrybutów typu dobra

TypeOfAttribute – mapuje nazwę atrybutu na jego typ:

String getName();

Class getType();

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

```

MBeanInfo getMBeanInfo();

void receiveOffer(Offer offer);

void receiveRefusal(String sessionId, String contractorID, int offerCount);

void receiveContractSign(String sessionId, String contractorID);

void receiveRenegotiation(Offer offer);

void receiveGood(Good good, String sessionId, int container);

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);

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

void registerService();

void unregisterService();

void receiveIntention(ClientOffer offer);

void receiveCounteroffer(ClientOffer counterOffer);

void receiveSignOffer(ClientOffer signOffer);

void storeGood(Good good);

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);

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();

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);

OfferStatusEnum – status oferty:

PREPARING_INTENTION(0x0), // stan początkowy (czekamy aż przez JMX
// zostanie wypełniona intencja)
INTENTIN_READY(0x1), // stan po wprowadzeniu treści intencji
// przez JMX, możliwa wysłania do producenta
WAITING_FOR_OFFER(0x2), // stan po wysłaniu intencji do producenta
// jak przyjdzie refusal to przejść do
// 0x9
OFFER_RECEIVED(0x3), // stan po odebraniu oferty od producenta
// z tego stanu można przejść do:
// 0x4, 0x6 lub 0x9
PREPAING_COUNTOFFER(0x4), // czekamy aż przez JMX zostanie
// wypełniona kontroferata
COUNTOFFER_READY(0x5), // stan po wprowadzeniu treści
// kontroferaty przez JMX, możliwa wysłania i
// wtedy przejść do 0x2
SIGING(0x6), // stan po wysłaniu checi podpisania, jak
// zostanie odebrane refusal to przejść
// do 0x9
SIGNED(0x7), // stan po odebraniu potwierdzenia
// podpisania
CANCEL(0x9);

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);

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);

ContractStatusEnum – status kontraktu:

getContractStatus();

NEGOTIATION(0x1),
AWAITING(0x2),
REALISATION(0x4),
SETTLEMENT(0x8),
BEFORE_NEGOTIATION(0x10),
CLOSED(0x20),
WAIT_FOR_RESPONSE(0x40);

Supply:

Date getDueDate();

void setDueDate(Date dueDate);

Good getGood();

void setGood(Good good);

Service – usługa:

Map<String, Double> getRequiredFeaturesByNames();

ServiceType – typ usługi:

```
List<TypeOfGood> getInGoodsTypes();
```

```
List<TypeOfGood> getOutGoodsTypes();
```

```
List<String> getNegotiableFeatures();
```

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

Reputation – przechowuje reputacje klienta / producenta:

```
void putComment(int points);
```

```
String getAgentName();
```

```
int getPositives();
```

```
int getNegatives();
```

```
int getTransactionCount();
```

ServiceRegistry – rejestr usług:

```
void registerService(String serviceName, String agentName);
```

```
void unregisterService(String serviceName, 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);
```

```
// Global Mappings
```

```
TypeOfAttribute getAttribute(String attributeName);
```

```
TypeOfGood getGood(String goodName);
```

```
ServiceType getServiceType(String serviceName);
```

MethodEnvelope - zawiera funkcje pomocnicze do serializacji i deserializacji obiektów do i z postaci XML:

```
void addArgument(Object obj);
```

```
void setFunctionName(String functionName);
```

```
String getFunctionName();
```

```
@SuppressWarnings("unchecked")
```

```
<T> T getArgument(int index);
```

```
String getArgumentClassName(int index);
```

```
String toXML();
```

JademxAgentTesting i JMXTest - klasy testowe, służą do weryfikacji działania systemu

4. Ważniejsze elementy systemu:

Poniżej znajduje się opis odpowiedzialności wybranych, ważniejszych klas systemu:

- (ClientAgent) Klienci:
 - Odpytywanie rejestru usług
 - Tworzenie dokumentu intencji
 - Odbieranie ofert
 - Historia negocjacji/renegocjacji
 - Dostarczanie dóbr
 - Wystawianie oceny producentowi (pozytywna, negatywna)
- (ProducerAgent) Producenci:
 - Wypełnienie rejestru usług
 - Magazyn dóbr
 - Odpowiedzi na oferty
 - Produkcja dóbr
 - Tworzenie dokumentów intencji dla producentów podwykonawców
- (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

5. 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.

6. Zasada działania

Aby wyjaśnić w jaki sposób poszczególne obiekty systemu wchodzą w interakcje między sobą, a także w jakim kontekście wywoływane są liczne zaimplementowane funkcje, poniżej zostały zamieszczone diagramy życia dla kluczowych elementów systemu: Rejestru Usług oraz komunikacji Klient - Konsument.

Rejestr usług - Rejestracja, wyrejestrowanie usługi:

Services Registry	Klient		Producent		Komentarz
	Agent	JMX	Agent	JMX	
					Uruchomienie agenta producenta
			<i>registerService</i>		Agent wysyła do SR chęć zarejestrowania swojej usługi
registerService					Usługa rejestrowana w SR
					Zamykanie agenta producenta
			<i>unregisterService</i>		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		<i>prepareTask</i>			Wybranie w konsoli JMX klienta co chcemy otrzymać i wygenerowanie owego sesionID i mainSessionID
			<i>prepareTask</i>				Inicjalizacja obiektu TaskValues
		waiting for services	<i>getServices</i>				Pobranie listy usług realizujących zamawiane dobro wraz z reputacją
getServices							
	negotiation	preparing intention					Przygotowano szablony dokumentów intencji, trzeba wypełnić intencje ręcznie przez konsolę JMX
				<i>prepareIntention</i>			Wprowadzenie w konsoli JMX wartości cech intencji
		intention ready					Stan: intencje zostały wypełnione, gotowe do rozesłania
				<i>sendIntention</i>			Zlecenie w konsoli JMX wysłania intencji
			<i>sendIntention</i>		<i>receiveIntention</i>		Wysłanie intencji i odebranie ich przez producenta
		waiting for offer					Czekanie aż producent odpowie ofertą
					<i>getClientReputation</i>		Pobranie od rejestru usług reputacji klienta
getClientReputation						<i>prepareOffer</i>	Stan: ofertę trzeba wypełnić ręcznie przez JMX
						<i>sendOffer/sendRefusal</i>	Zlecenie wysłania oferty/zlecenie wysłania odmowy wykonania usługi
		offer received	<i>receiveOffer/receiveRefusal</i>		<i>sendOffer/sendRefusal</i>		Wysłanie jw. i odebranie tego przez agenta klienta
		preparing countoffer					Stan: klient otrzymał ofertę i przez JMX ma zdecydować co z nią zrobić

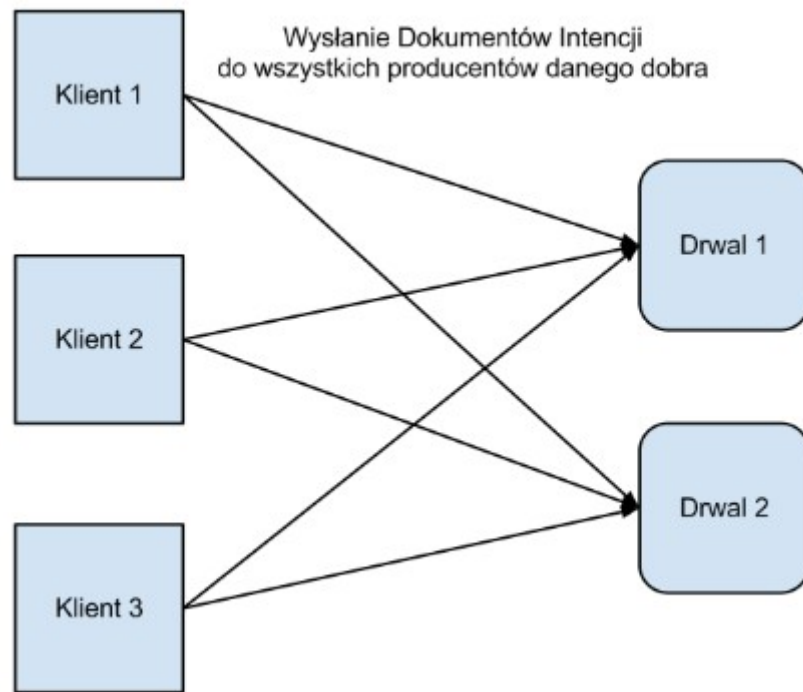
				<i>prepareCounteroffer</i>			W tym przypadku przez JMX jest wprowadzana kontroferta
		counteroffer ready					Stan: kontroferta gotowa do wysłania
				<i>sendCounteroffer</i>			Zlecenie przez JMX wysłania kontroferty
			<i>sendCounteroffer</i>		receiveCounteroffer		Wysłanie kontroferty i odebranie jej przez agenta producenta
		waiting for offer					Oczekiwanie na nową ofertę od producenta
						<i>prepareOffer</i>	Stan: ofertę trzeba wypełnić ręcznie przez JMX
						<i>sendOffer</i>	Zlecenie wysłania oferty/zlecenie wysłania odmowy wykonania usługi
		offer received	receiveOffer		<i>sendOffer</i>		Wysłanie oferty i odebranie jej przez klienta
		preparing countoffer					Stan: klient otrzymał ofertę i przez JMX ma zdecydować co z nią zrobić
				<i>signOffer/negotiation Break</i>			Tym razem przez JMX zlecono podpisanie kontraktu/rezygnację z dalszych negocjacji
		siging offer	<i>signOffer/negotiationBreak</i>		receiveSignOffer/receiveBreakNegotiation		Wysłanie zlecenia jw. i odebranie go przez producenta
						<i>signOffer/rejectOffer</i>	Decyzja przez JMX czy producent ma zdecydować się podpisać czy zrezygnować
	Waiting	signed	receiveSignedOffer/receiveReject Offer		<i>sendSignedOffer/sendReject Offer</i>		Wysłanie jw. i odebranie przez agenta. Jeżeli odmówiono podpisania kontraktu mimo
		normal		<i>storeGood</i>			Przekazanie agentowi dobra przez konsolę JMX
			storeGood				Przekazanie agentowi dobra można dokonać w każdym momencie kontraktu
				<i>transferGood</i>			Zlecenie przekazania dobra innemu agentowi

			<i>transferGood</i>		<i>storeGood</i>		Przekazanie dobra innemu agentowi (zmiana właściciela)
			<i>receiveStoreGoodACK</i>		<i>storeGoodACK</i>		Przesłanie potwierdzenia odebrania zasobu i że jest on zgodny z oczekiwaniami
				<i>sendRenegotiation</i>			Zlecenie przez JMX przeprowadzenie renegotjacji z inicjatywy klienta
		init renegotiation	<i>sendRenegotiation</i>		<i>receiveRenegotiation</i>		Wysłanie do producenta kontroferty inicjującej renegotjacje
		waiting for offer				<i>prepareOffer</i>	Oczekiwanie na ofertę i przeprowadzenie akcji negocjacji wg takich samych zasad jak podczas negocjacji.
						<i>sendOffer</i>	
		offer received	<i>receiveOffer</i>		<i>sendOffer</i>		Równolegle przeprowadza się negocjacje nowego kontraktu z innymi producentami ale to już jest osobny kontrakt (nie ten).
		preparing countoffer					
				<i>prepareCounteroffer</i>			
		counteroffer ready					Przerwanie negocjacji skutkuje wystawieniem negatywnego komentarza klientowi.
				<i>sendCounteroffer</i>			
			<i>sendCounteroffer</i>		<i>receiveCounteroffer</i>		
		waiting for offer				<i>prepareOffer</i>	Zakończenie negocjacji renegotjacji pozytywnie może ale nie musi skutkować wystawieniem słabszego komentarza pozytywnego (zależy od korzyści nowego kontraktu).
						<i>sendOffer</i>	
		offer received	<i>receiveOffer</i>		<i>sendOffer</i>		Renegocjacja z inicjatywy producenta.
		preparing countoffer		<i>signOffer/negotiation Break</i>			
			<i>signOffer/negotiationBreak</i>		<i>receiveSignOffer/receiveBreakNegotiation</i>		
		signed				<i>signOffer/rejectOffer</i>	Przebiega niemal identycznie jak
			<i>receiveSignedOffer</i>		<i>sendSignedOffer</i>		
		normal				<i>sendRenegotiation</i>	
		init renegotiation	<i>receiveRenegotiation</i>		<i>sendRenegotiation</i>		
		preparing countoffer					
				<i>prepareCounteroffer</i>			

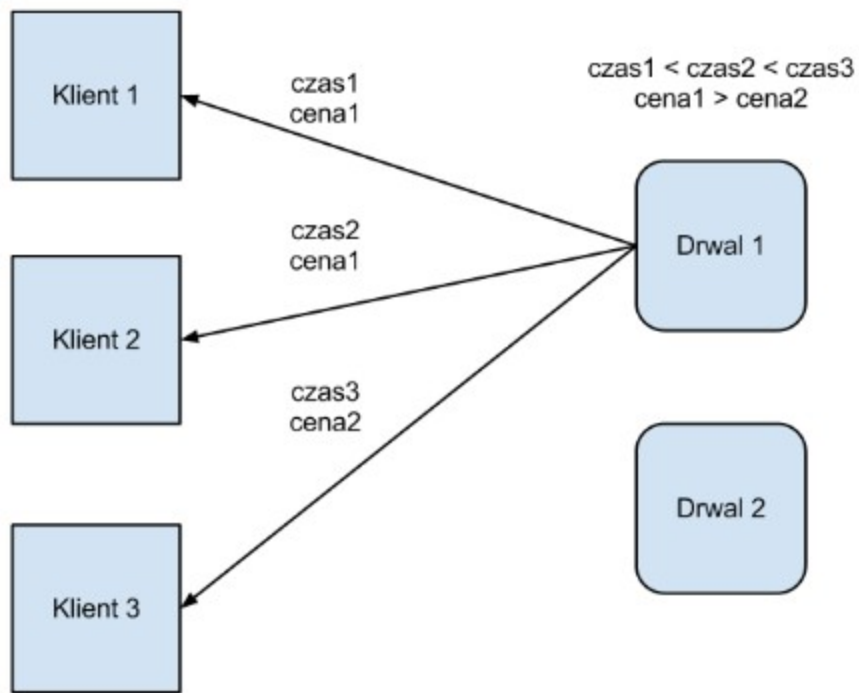
		counteroffer ready					
				<i>sendCounteroffer</i>			
			<i>sendCounteroffer</i>		receiveCounteroffer		
		waiting for offer					
						<i>prepareOffer</i>	
						<i>sendOffer</i>	
		offer received	receiveOffer		<i>sendOffer</i>		
		preparing countoffer					
				<i>singOffer/negotiation Break</i>			
		siging offer	<i>signOffer/negotiationBreak</i>		receiveSignOffer/receiveBreakNegotiation		
						<i>singOffer/rejectOffer</i>	
		signed	receiveSignedOffer		<i>sendSignedOffer</i>		
		normal					
						<i>runWorkProcess</i>	Zlecenie rozpoczęcia prac nad dobrem wyjściowym
	Working	normal					Prace nad dobrem wyjściowym rozpoczęto
... to samo co w fazie oczekiwania (Waiting) ...							
						<i>finalWorkProcess</i>	W konsoli JMX informacja, że produkt gotowy do odbioru
	Payoff	normal				<i>storeGood</i>	
					storeGood		
						<i>transferGood</i>	
			storeGood		<i>transferGood</i>		
			<i>storeGoodACK</i>		receiveStoreGoodACK		
				<i>storeGood</i>			
			storeGood				
				<i>transferGood</i>			
			<i>transferGood</i>		storeGood		
			receiveStoreGoodACK		<i>storeGoodACK</i>		
				<i>putComment</i>			Oba podmioty wystawiają sobie po

putProducentComment			putComment				komentarzu. Komentarz można wystawić tylko w fazie finalizacji
						putComment	
putClientComment					putComment		
	finalized						Wszystkie procesy związane z kontraktem zostały zakończone

Jedną z ważniejszych cech systemu, jest wsparcie dla jednoczesnych negocjacji z wieloma producentami. Działa to w następujący sposób:

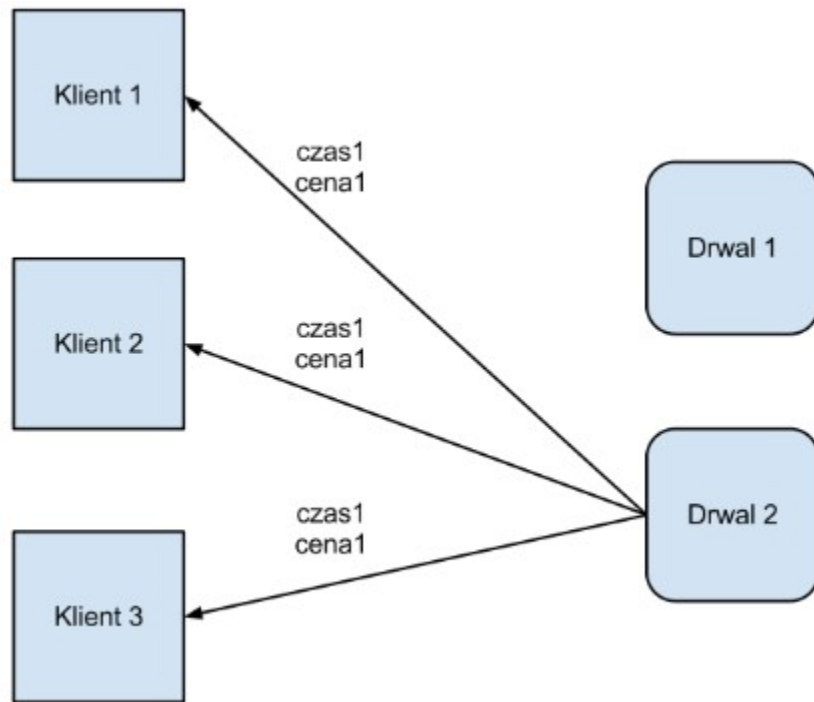


Na etapie negocjacji klienci wysyłają Dokumenty Intencji do wszystkich producentów dobra danego typu. Określają w nich typy i ilości dóbr, które dostarczają oraz jakiego typu dobra oraz cech oczekują w zamian.



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

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

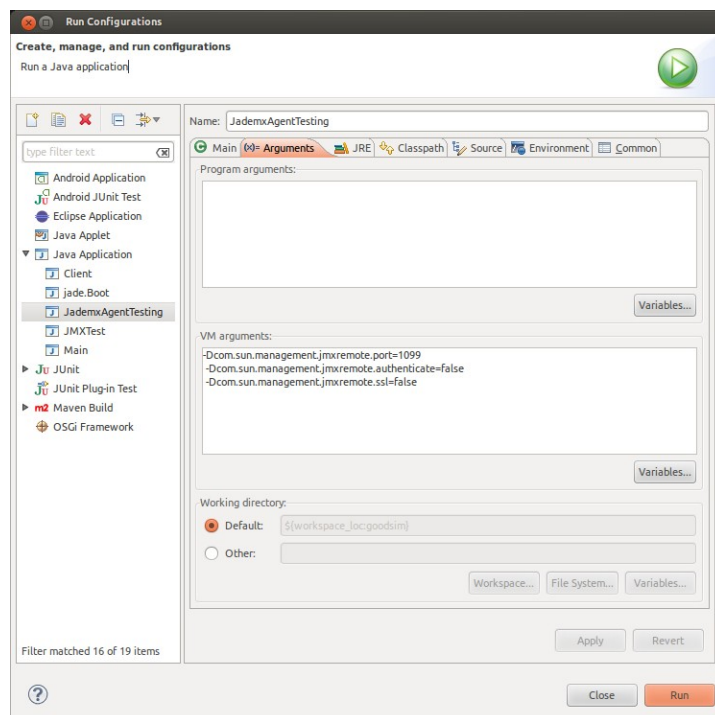
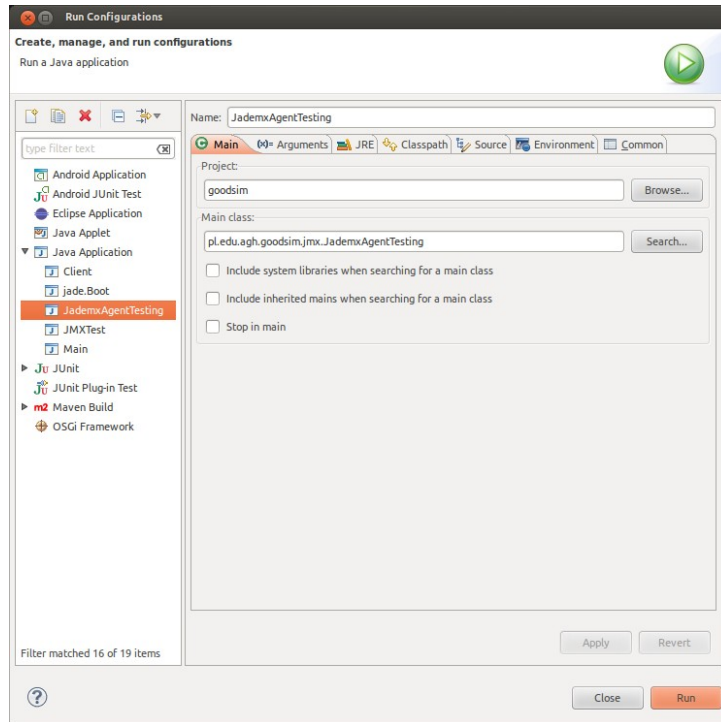


Niektórzy producenci nie rezerwują czasu dla poszczególnych klientów, co może powodować problemy w przypadku przyjęcia oferty przez wszystkich klientów.

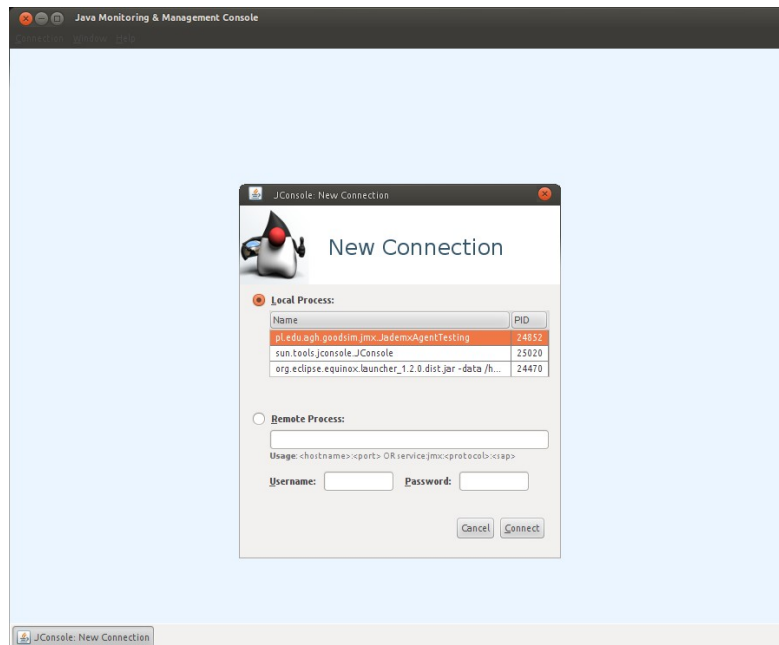
Jeżeli producent nie będzie mógł wywiązać się z realizacji zamówienia, konieczne może być renegotjowanie kontraktu. Może to prowadzić do obniżenia reputacji producenta w przypadku niepowodzenia.

7. Demonstracja

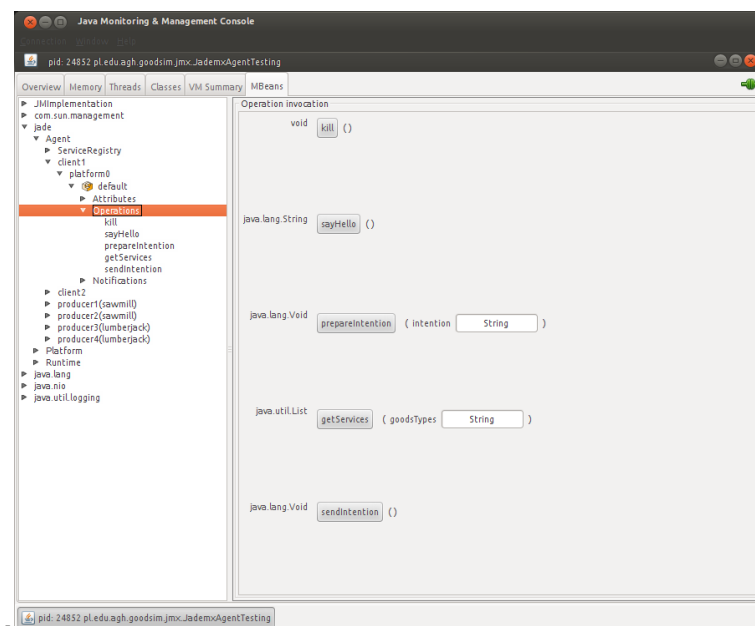
W celu uruchomienia przykładu z poziomu eclipse, należy użyć następującej konfiguracji.



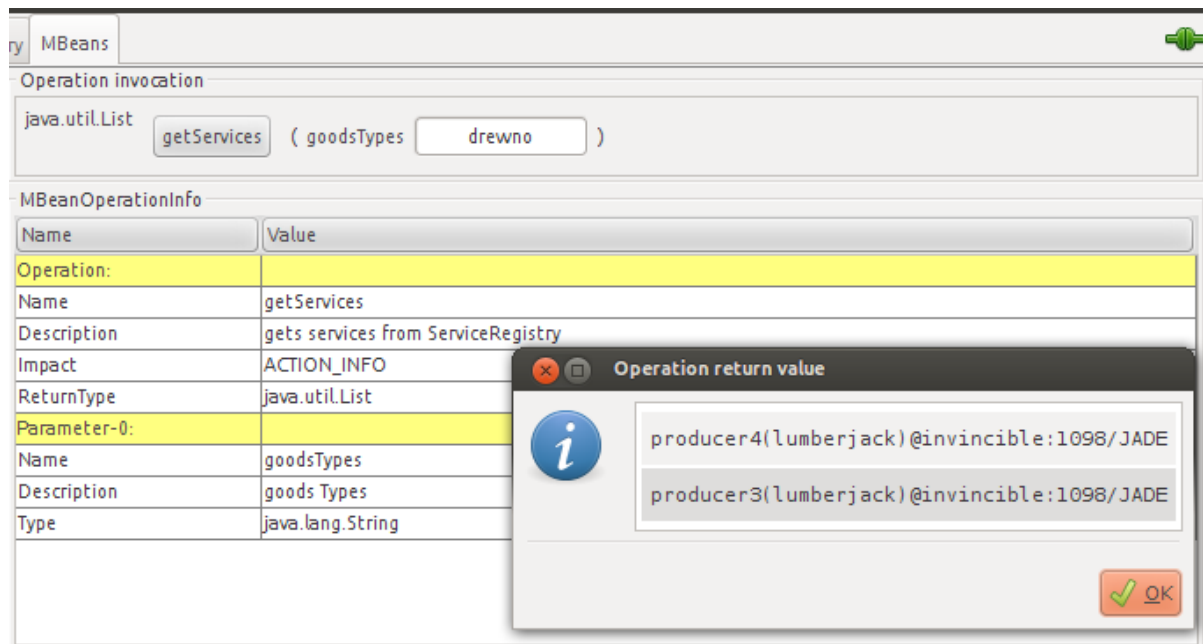
Po uruchomieniu aplikacji, możemy się do niej podłączyć poprzez **jconsole**. W uruchomionym oknie, będziemy mieli dostęp zarówno do platformy agentowej jak i rezydujących na niej agentów.



Wybierając odpowiedniego agenta, mamy możliwość wywołania na nim metody przy pomocy JMX

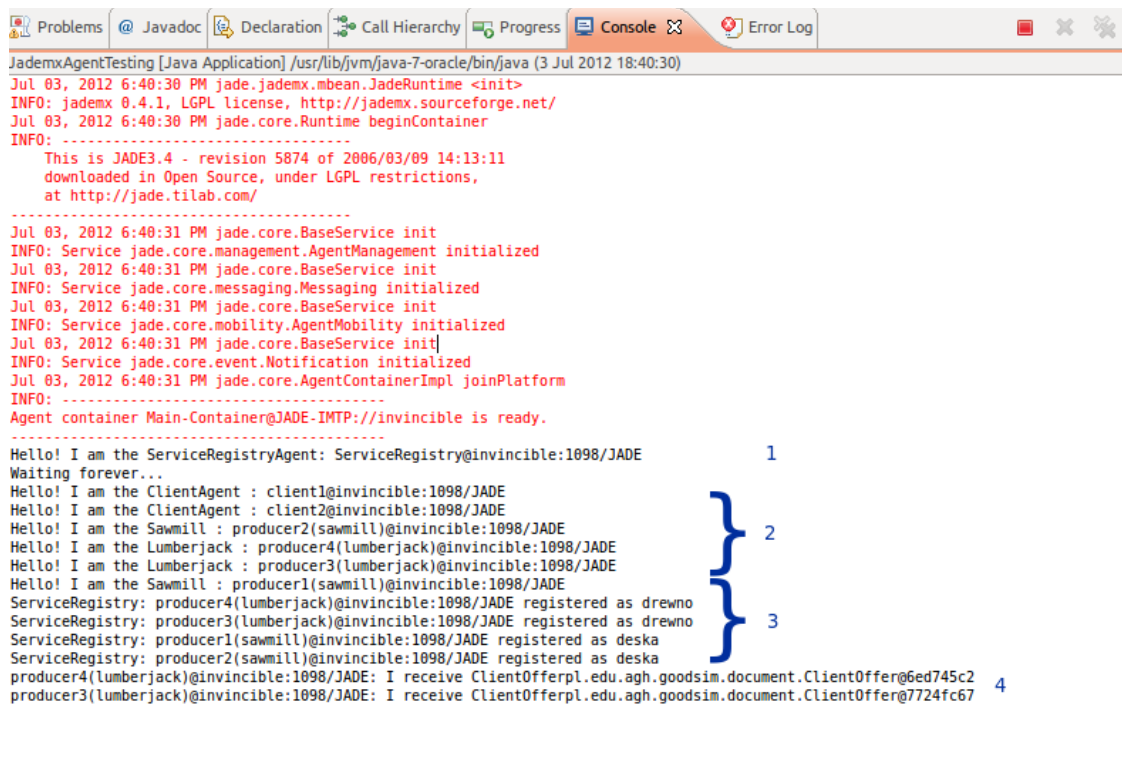


Przykładowe wywołanie metody JMX getServices z parametrem "drewno".



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

Poniżej log z przykładowego uruchomienia platformy wymiany dóbr, wraz z opisem poszczególnych kroków.



```
JademxAgentTesting [Java Application] /usr/lib/jvm/java-7-oracle/bin/java (3 Jul 2012 18:40:30)
Jul 03, 2012 6:40:30 PM jade.jademx.mbean.JadeRuntime <init>
INFO: jademx 0.4.1, LGPL license, http://jademx.sourceforge.net/
Jul 03, 2012 6:40:30 PM jade.core.Runtime beginContainer
INFO: -----
This is JADE3.4 - revision 5874 of 2006/03/09 14:13:11
downloaded in Open Source, under LGPL restrictions,
at http://jade.tilab.com/
-----
Jul 03, 2012 6:40:31 PM jade.core.BaseService init
INFO: Service jade.core.management.AgentManagement initialized
Jul 03, 2012 6:40:31 PM jade.core.BaseService init
INFO: Service jade.core.messaging.Messaging initialized
Jul 03, 2012 6:40:31 PM jade.core.BaseService init
INFO: Service jade.core.mobility.AgentMobility initialized
Jul 03, 2012 6:40:31 PM jade.core.BaseService init
INFO: Service jade.core.event.Notification initialized
Jul 03, 2012 6:40:31 PM jade.core.AgentContainerImpl joinPlatform
INFO: -----
Agent container Main-Container@JADE-IMTP://invincible is ready.
-----
Hello! I am the ServiceRegistryAgent: ServiceRegistry@invincible:1098/JADE 1
Waiting forever...
Hello! I am the ClientAgent : client1@invincible:1098/JADE
Hello! I am the ClientAgent : client2@invincible:1098/JADE
Hello! I am the Sawmill : producer2(sawmill)@invincible:1098/JADE
Hello! I am the Lumberjack : producer4(lumberjack)@invincible:1098/JADE
Hello! I am the Lumberjack : producer3(lumberjack)@invincible:1098/JADE
Hello! I am the Sawmill : producer1(sawmill)@invincible:1098/JADE
ServiceRegistry: producer4(lumberjack)@invincible:1098/JADE registered as drewno
ServiceRegistry: producer3(lumberjack)@invincible:1098/JADE registered as drewno
ServiceRegistry: producer1(sawmill)@invincible:1098/JADE registered as deska
ServiceRegistry: producer2(sawmill)@invincible:1098/JADE registered as deska
producer4(lumberjack)@invincible:1098/JADE: I receive ClientOfferpl.edu.agh.goodsim.document.ClientOffer@6ed745c2 4
producer3(lumberjack)@invincible:1098/JADE: I receive ClientOfferpl.edu.agh.goodsim.document.ClientOffer@7724fc67
```

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.

8. Podsumowanie i propozycje na kontynuację projektu

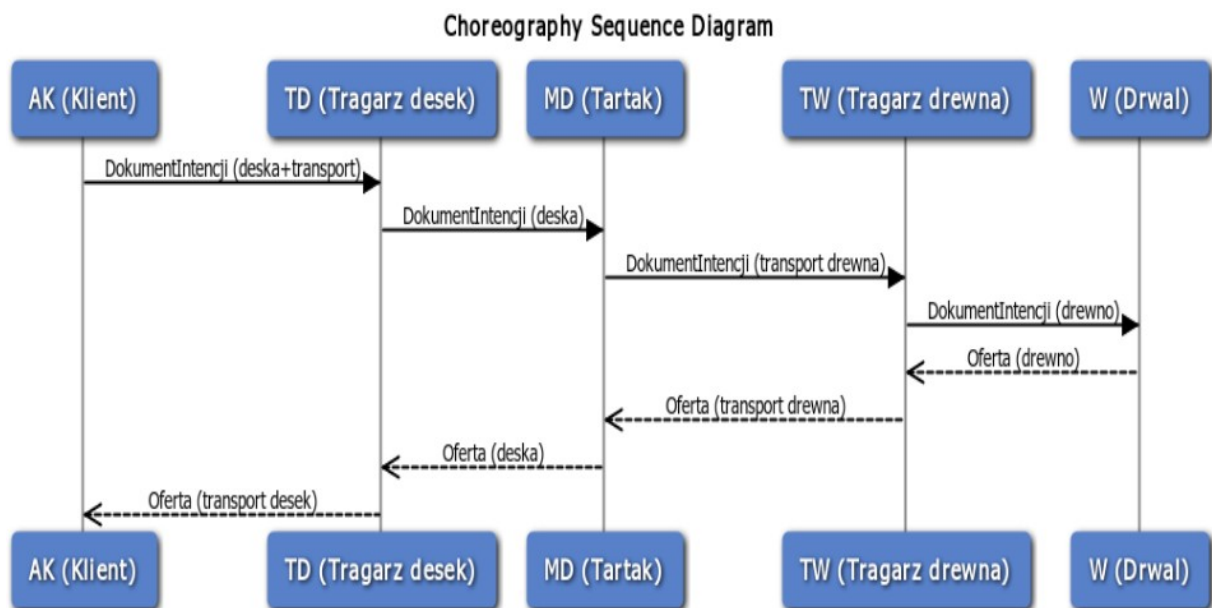
Zrealizowany projekt dostarcza solidny fundament platformy wymiany dóbr, wraz z pełną integracją z technologiami JADE i JMX.

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:



W przypadku algorytmu orkiestracji sytuacja wyglądałaby w następujący sposób:

