

Systemy zdecentralizowane i agentowe - projekt.

Temat: Środowisko do badania różnych rodzajów aukcji

z wykorzystaniem technik agentowych

Autorzy: Tomarz Chmielarz i Andrzej Ryl

[1. Cel projektu 2](#_Toc448057790)

[2. Zarys architektury 2](#_Toc448057791)

[3. Agenci w systemie 4](#_Toc448057792)

[3.1 AuctionHouse 5](#_Toc448057793)

[3.2 Auction 6](#_Toc448057794)

[3.3 Buyer 7](#_Toc448057795)

[3.4 DataAnalyzer 8](#_Toc448057796)

[4. Typy aukcji 8](#_Toc448057797)

[4.1 Aukcja angielska 8](#_Toc448057798)

[4.2 Aukcja japońska 8](#_Toc448057799)

[4.3 Aukcja holenderska 9](#_Toc448057800)

[4.4 Aukcja typu 'Sealed-bid' 9](#_Toc448057801)

[4.5 Aukcja eliminacyjna 9](#_Toc448057802)

[5. Strategie kupców 9](#_Toc448057803)

[5.1 DefaultStrategy 9](#_Toc448057804)

# 1. Cel projektu

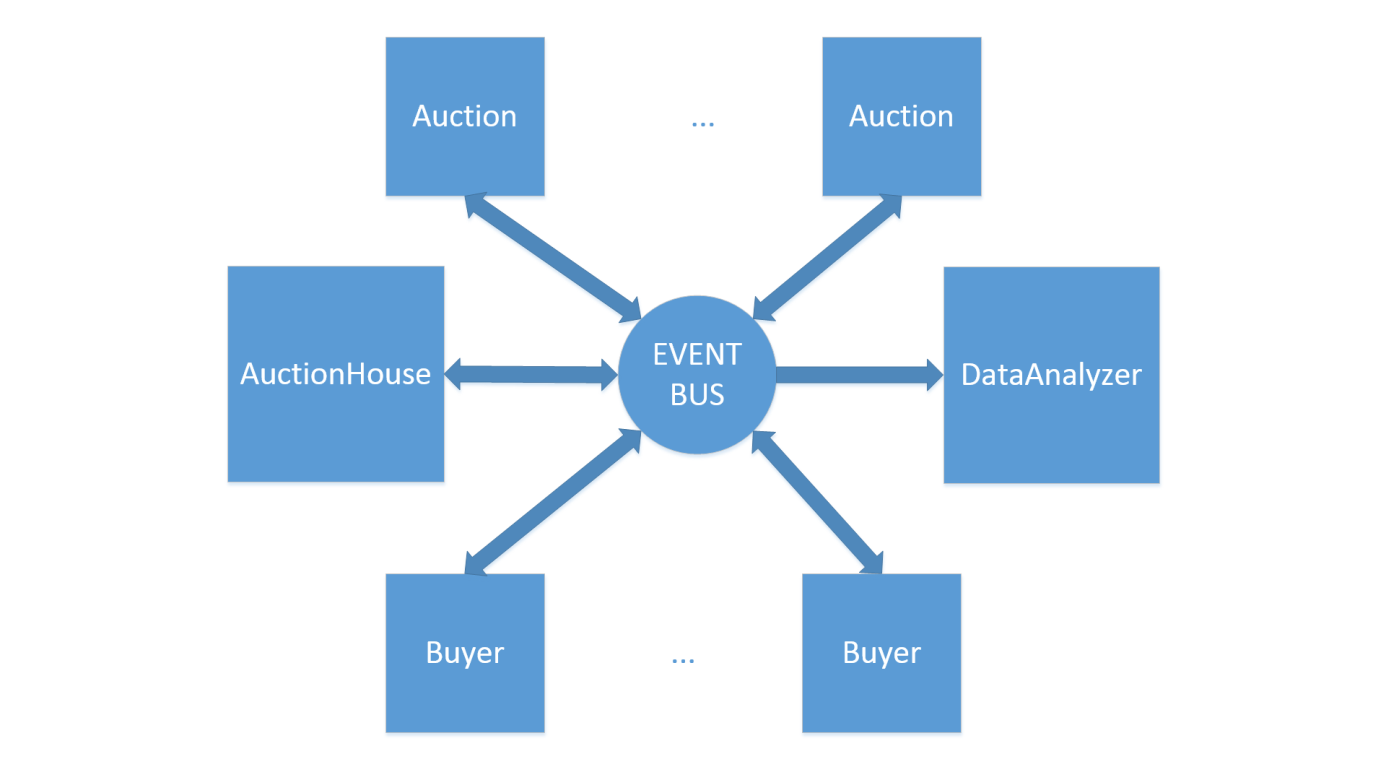
Nasz projekt zakłada stworzenie systemu, który będzie w stanie przetestować różne modele aukcji z pomocą technik agentowych. Do tego celu mamy zamiar użyć agentów reprezentujących kupców o różnych strategiach grania na aukcjach.

W ostatecznej wersji nasz system będzie w stanie porównać skuteczność takich aukcji jak m.in. aukcja angielska, czy aukcja japońska na podstawie zachowań generowanych przez różne strategie kupców. Dzięki temu będziemy w stanie określić, który typ aukcji jest najskuteczniejszy (przynosi największe zyski) dla zdefiniowanych wcześniej warunków (cen początkowych, cen maksymalnych oraz charakterystyki kupujących).

# 2. Zarys architektury

Nasz system opiera się na wzorcu projektowym Publish/Subscribe. Dzięki zastosowaniu tego wzorca, mamy możliwość łatwej komunikacji pomiędzy wszystkimi agentami. Każdy z takich agentów może 'zapisać się' na zdarzenia pewnego typu (np. kupujący zapisują się na zdarzenia typu 'Aukcja nr xxx się rozpoczęła') oraz emitować swoje własne zdarzenia, na które zapisują się inni (np. aukcja emituje zdarzenie mówiące o tym, że się rozpoczęła lub zakończyła).

Ogólną architekturę prezentuje poniższy diagram.



W samym centrum naszego systemu znajduje się komponent EventBus, który służy nam do efektywnej komunikacji między wszystkimi agentami.

Każdy z trzech typów agentów (Auction, Buyer oraz AuctionHouse) emituje swoje własne zdarzenia i 'zapisuje się' na wydarzenia, które go interesują. Dodatkowy agent w systemie (DataAnalyzer) jedynie nasłuchuje wszelkich zdarzeń w systemie i na ich podstawie porównuje skuteczność różnych typów aukcji. Dokładny opis wszystkich wydarzeń zostanie podany w następnych rozdziałach.

Poniżej znajduje się aktualny schemat UML całego systemu.

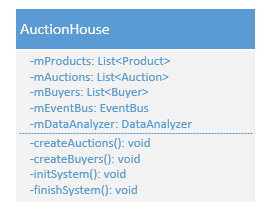


# 3. Agenci w systemie

Nasz system posiada obecnie 4 agentów. Są to:

* AuctionHouse - agent odpowiadający za utworzenie wszystkich komponentów oraz uruchomienie i zakończenie działania całego systemu
* Auction - agent reprezentujący poszczególne aukcje w systemie
* Buyer - agent reprezentujący kupców biorących udział w aukcjach
* DataAnalyzer - agent odpowiedzialny za zebranie wszystkich danych podczas działania systemu, przeanalizowanie ich i na ich podstawie wygenerowanie porównania poszczególnych typów aukcji

## 3.1 AuctionHouse



**Emituje zdarzenia typu:**

* SystemStartedEvent
* SystemFinishedEvent

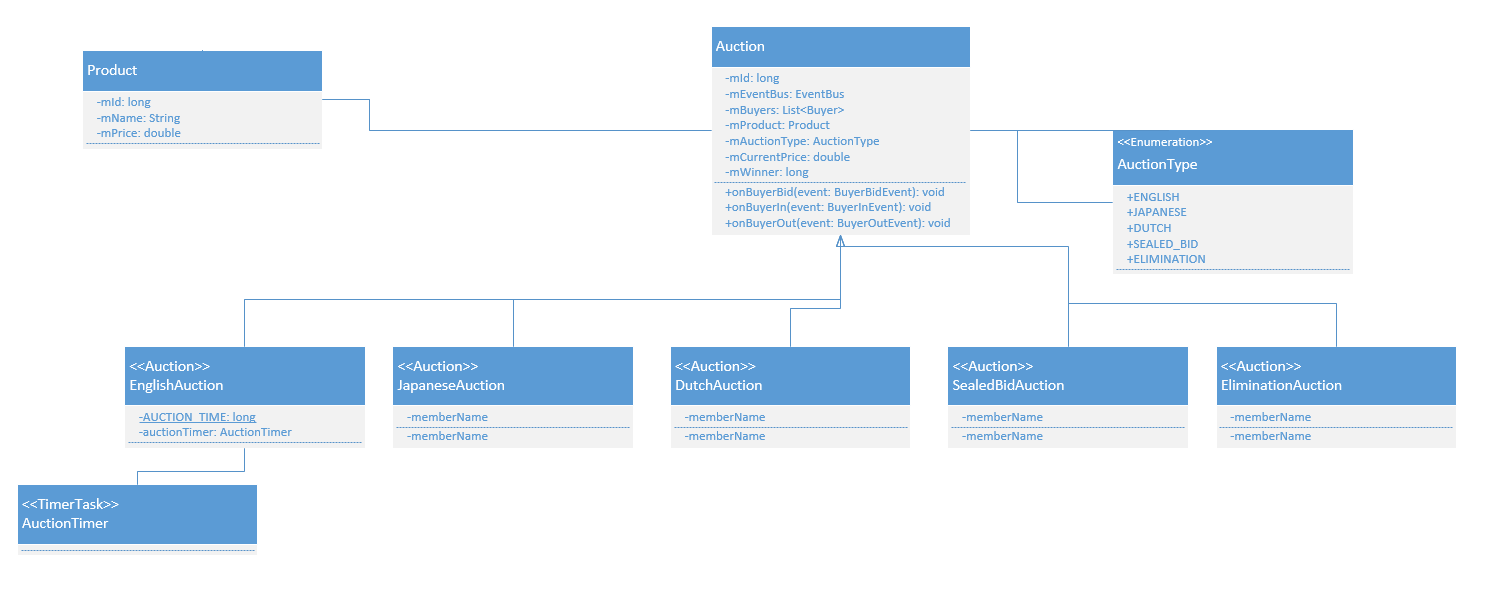
**Nasłuchuje zdarzeń typu:**

* AuctionStartedEvent
* AuctionFinishedEvent

Ten agent odpowiedzialny jest za zainicjowanie oraz zakończenie działania całego systemu. Na początku tworzy on listę aukcji (różnych typów) oraz listę kupujących (korzystających z różnych strategii kupowania) i przyporządkowuje tym aukcjom produkty ze swojej listy (produkty posiadają jedynie nazwę oraz swoją cenę rynkową).

Następnie agent ten inicjuje agenta DataAnalyzer i emituje zdarzenie typu 'SystemStartedEvent', które uruchamia wszystkie aukcje. Po wykonaniu tej czynności agent nasłuchuje zdarzeń typu 'AuctionStartedEvent' (aukcja sygnalizuje, że się rozpoczęła) oraz 'AuctionFinishedEvent' (aukcja sygnalizuje, że się zakończyła). W momencie, w którym wszystkie aukcje się zakończą, emituje on zdarzenie typu 'SystemFinishedEvent' sygnalizujące agentowi DataAnalyzer, że należy przeanalizować wszystkie zebrane dane i wygenerować podsumowanie.

## 3.2 Auction



**Emituje zdarzenia typu:**

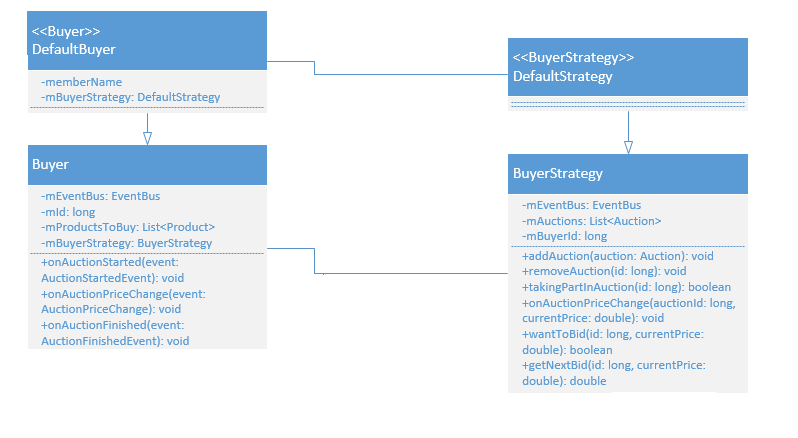
* AuctionStartedEvent
* AuctionFinishedEvent
* AuctionPriceChangeEvent

**Nasłuchuje zdarzeń typu:**

* SystemStartedEvent
* SystemFinishedEvent
* BuyerBidEvent
* BuyerInEvent
* BuyerOutEvent

Ten agent reprezentuje aukcję w systemie. Jest to klasa abstrakcyjna, która umożliwia nam tworzenie różnych typów aukcji korzystających z identycznego interfejsu. Główny model aukcji emituje zdarzenia opisujące jego stan: 'AuctionStartedEvent' (aukcja się rozpoczęła), 'AuctionFinishedEvent' (aukcja się zakończyła) oraz 'AuctionPriceChangeEvent' (zmieniła się cena produktu sprzedawanego na tej aukcji). Wszelkie reakcje na emitowane przez innych agentów zdarzenia są natomiast implementowane w poszczególnych typach aukcji, gdyż zachowania te różnią się w zależności od typu. Poszczególne zachowania zostaną opisane w następnych rozdziałach.

## 3.3 Buyer



**Emituje zdarzenia typu:**

* BuyerBidEvent
* BuyerInEvent
* BuyerOutEvent

**Nasłuchuje zdarzeń typu:**

* AuctionStartedEvent
* AuctionFinishedEvent
* AuctionPriceChangeEvent
* SystemStartedEvent
* SystemFinishedEvent

Ten aktor reprezentuje kupujących biorących udział w aukcjach. Każdy z kupujących korzysta ze swojej własnej strategii i na jej podstawie decyduje, czy przebijać dalej, czy wycofać się aukcji. Podstawowa implementacja kupującego emituje zdarzenia typu 'BuyerInEvent' (kupujący wchodzi do gry), 'BuyerOutEvent' (kupujący wycofuje się z aukcji) oraz 'BuyerBidEvent' (kupujący przebija) na podstawie swojej strategii. Poszczególne implementacje strategii będą opisane w późniejszych rozdziałach.

## 3.4 DataAnalyzer

**Ten aktor nasłuchuje wszystkich wydarzeń i nie wysyła żadnych.**

Jego implementacja jest wciąż niekompletna z powodu braku implementacji wszystkich typów aukcji oraz wszystkich typów strategii kupujących.

# 4. Typy aukcji

Każdy z typów aukcji rozszerza podstawowego agenta, jakim jest Auction. Podstawowa implementacja obsługuje cztery zdarzenia :

* AuctionStartedEvent - to zdarzenie jest emitowane podczas rozpoczynania aukcji
* BuyerBidEvent - jeśli cena przekazana w tym zdarzeniu jest wyższa od aktualnej, ta zmienia się, a agent emituje zdarzenie typu 'AuctionPriceChangedEvent'
* BuyerInEvent - jeśli ten kupujący jeszcze nie bierze udziału w naszej aukcji, to agent go dodaje
* BuyerOutEvent - jeśli kupujący bierze udział w naszej aukcji, to agent go usuwa

Poszczególne typy jedna wprowadzają szereg zmian w celu implementacji reguł, które rządzą poszczególnymi aukcjami.

## 4.1 Aukcja angielska

**Zasady**

* sprzedawca daje cenę początkową
* kupcy starają się przebić cenę
* aukcja kończy się po sprecyzowanym wcześniej czasie
* zwycięzca MUSI kupić produkt po tej cenie, jaką podał

**Zmiany w podstawowej implementacji**

* podczas rozpoczynania aukcji uruchamiany jest timer (wykorzystywana jest do tego klasa AuctionTimer)
* podczas rozpoczynania aukcji generowana jest cena produktu (losowa wartość mniejsza od ceny rynkowej)
* każda wyższa propozycja ceny powoduje wyznaczenie nowego tymczasowego zwycięscy
* po zakończeniu czasu emitowane jest zdarzenie typu 'AuctionFinishedEvent'. W tym samym czasie tymczasowy zwycięzca zostaje ostatecznym zwycięzcą

## 4.2 Aukcja japońska

**Zasady**

* sprzedawca daje cenę początkową
* kupujący mówią czy są "in", czy nie, czyli czy są skłonni kupić produkt po tej cenie, czy nie
* następnie sprzedawca zwiększa cenę i znowu pyta
* jeśli kupiec odpada, to nie może wejść z powrotem
* aukcja kończy się, gdy zostaje jeden kupiec i on MUSI kupić

## 4.3 Aukcja holenderska

**Zasady**

* sprzedawca daje bardzo wysoką cenę
* potem stosunkowo ją zmniejsza
* jak kupiec usłyszy zadowalającą go cenę, to się zgłasza
* ten, który pierwszy się zgłosi, MUSI kupić produkt

## 4.4 Aukcja typu 'Sealed-bid'

**Zasady**

* każdy z kupców daje tzw. 'sealed-bid' czyli swoją cenę, ALE potajemnie
* żaden z kupców nie zna propozycji pozostałych kupców
* kupiec z najwyższą ofertą MUSI kupić produkt ale po cenie jaką sam podał lub jaką podał kupiec, który był na drugim miejscu

## 4.5 Aukcja eliminacyjna

**Zasady**

* rozpoczynamy aukcję typu sealed-bid
* kupiec z najniższą ofertą odpada, a jego cena staję się jawną ceną minimalną drugiej rundy aukcji typu sealed-bid
* kontynuujemy tak, aż zostanie jeden kupiec i on wtedy płaci cenę minimalną ostatniej rundy

# 5. Strategie kupców

Każdy z kupców w naszym systemie działa według swojej strategii. Istnieje ogólny model reprezentowany przez klasę 'BuyerStrategy', który jest rozszerzany przez poszczególne implementacje. Podstawowy model na podstawie implementacji metod 'getNextBid' oraz 'wantsToBid' emituje zdarzenia typu 'BuyerBidEvent' (przebicie ceny), 'BuyerInEvent' (kupujący bierze udział w aukcji) oraz 'BuyerOutEvent' (kupujący wycofuje się z aukcji).

## 5.1 DefaultStrategy

Ta strategia jest strategią najprostszą i została stworzona głównie po to, aby testować pozostałe komponenty. Kupujący przebija cenę zupełnie losowo (losowo wybierana jest zarówno kwota jak i fakt samej chęci przebicia).

**Implementacja wantsToBid**

Kupujący losowo wybiera, czy chce przebijać, czy nie.

**Implementacja getNextBid**

Kupujący losowo wybiera kwotę z przedziału (1.2 \* obecna cena; 1.3\*obecna cena).