Projekt WSD
Raport

System wspomagający wybór najlepszego baru

Zespół 2.0
Szymon Bezpalko
Michał Korzeniewski
Michał Piotrak
Paweł Piotrowski
Dawid Zaniewski

Organizacja projektu

Role osób w projekcie

Ze względu na wczesny etap projektu, przypisane role do konkretnych osób są dość ogólne. Przy realizacji kolejnych części projektu, zostaną one opisane w sposób bardziej szczegółowy:

- Szymon Bezpalko - specjalista ds. implementacji
- Michał Korzeniewski - lider zespołu
- Michał Piotrak - specjalista ds. architektury systemu
- Paweł Piotrowski - specjalista ds. architektury systemu
- Dawid Zaniewski - specjalista ds. implementacji

Repozytorium kodu

Kod naszego systemu będzie można znaleźć w publicznym repozytorium pod podanym adresem: https://github.com/SBe/WSD2019Z-distributed-beer-problem

Poprawki w dokumentacji

Zgodnie z zaleceniem odnośnie wyraźnego zaznaczania poprawek w dokumentacji dokonanych po ocenieniu danego etapu, postanowiliśmy oznaczać je czerwonym kolorem czcionki.

Część A

Identyfikacja i opis problemu

W ramach realizacji projektu, postanowiliśmy pochylić się nad problemem wyboru odpowiedniego lokalu do spożycia piwa dla danej osoby, biorąc pod uwagę jej indywidualne preferencje w najistotniejszych kwestiach dotyczących tego zagadnienia. Wbrew pozorom, rozważany problem nie jest wcale błahy, ponieważ wiąże się on z identyfikacją osobistych oczekiwań w stosunku do konkretnego baru, określeniem ich ważności dla nas oraz uzyskaniem niezbędnej wiedzy na temat, czy oceniany lokal zaspokaja nasze wcześniej sprecyzowane pragnienia.

W celu bardziej klarownego wytłumaczenia problematyki danej materii, posłużymy się konkretnym przykładem. Dana osoba chce napić się określonego piwa w miejscu znajdującym się niedaleko niej, które gwarantuje jej także możliwość spotkania się ze znajomymi w spokojnej atmosferze. Przypuśćmy, że posiada już jakąś wiedzę o istniejących lokalach w okolicy, jednak nadal może napotkać na dylematy dotyczące np.:

- kwestii dostępności wymarzonego piwa,
- liczby wolnych miejsc,
- hałasu w barze,
- wyboru pomiędzy lokalami, gdzie pierwszy oferuje w sprzedaży dane piwo, ale niestety nie gwarantuje zacisznej atmosfery. Za to drugi lokal nie posiada w swojej ofercie oczekiwanego piwa, ale może zaproponować coś podobnego oraz także pożądany spokój.

Jak widać, ze względu na brak wystarczającej ilości informacji, podmiot może mieć problem z podjęciem właściwej decyzji o wyborze odpowiadającego mu lokalu. Z tego względu postanowiliśmy zaprojektować system spełniający założenia architektury wieloagentowej, który wspomoże go przy tej czynności. System może się okazać pomocny dla wielu użytkowników, ze względu na narastającą popularność tzw. piw rzemieślniczych, z czym oczywiście wiąże się także rozwój lokali, serwujących tego typu napoje.

Rozwiązanie

Ogólny opis rozwiązania

W projektowanym systemie będzie można wyróżnić dwa rodzaje użytkowników - pierwsi z nich to będą potencjalni klienci lokali, drudzy będą odpowiadać konkretnym barom. Zainteresowany pójściem do baru będzie miał możliwość wyszukania w aplikacji mobilnej najlepszego dla niego miejsca do wypicia piwa poprzez określenie swoich preferencji. Ta czynność będzie polegała na wyspecyfikowaniu wartości / opcji oraz także stopnia ważności dla parametrów branych pod uwagę przy ocenie danego lokalu. Lista tych parametrów oraz przyjmowane przez nich wartości zostaną dokładnie przemyślane oraz ustalone, ale na pewno nie zabraknie tam takich czynników jak:

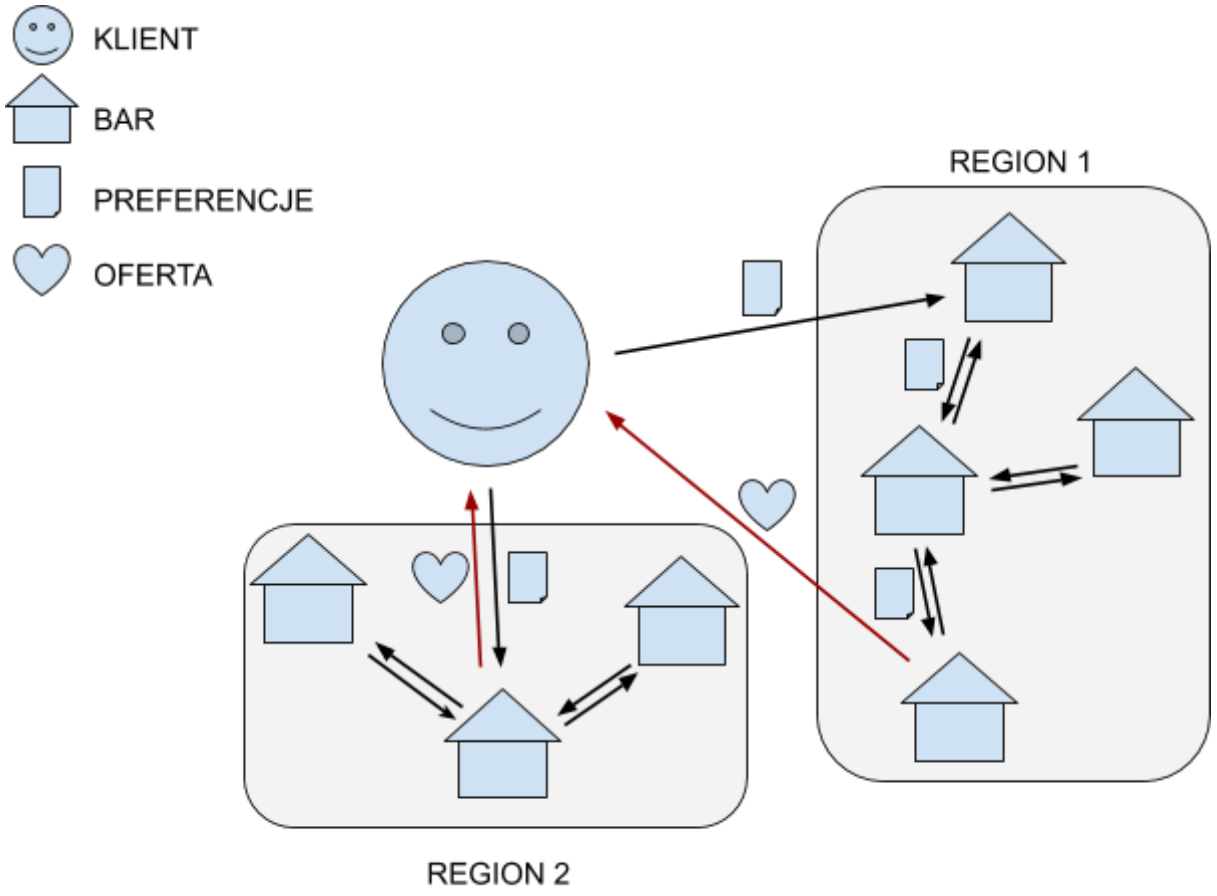
- odległość baru od aktualnej lokalizacji,
- dostępność podanego piwa,
- dostępność podanego stylu piwa,
- cena,
- hałas panujący w lokalu,
- dostępność miejsc.

Stopień ważności parametru w ocenie lokalu będzie można ustalić w skali procentowej, najprawdopodobniej poprzez odpowiednie przeciągniecie suwaka, znajdującego się przy danym czynniku.

Oprogramowanie po stronie baru będzie swoistym źródłem danych o jego ofercie, wyrażonej poprzez wymienione wcześniej parametry oceny lokalu. Aktualizację ich wartości będzie można dokonać w sposób manualny jak np. cenę danego piwa, ale nie wykluczamy tutaj również zastosowania zautomatyzowanych mechanizmów, które będą wyznaczały jakość niektórych czynników. W ten sposób bar może stać się złożonym systemem wieloagentowym, w którym dla przykładu występuje agent decybelomierz, służący za pomiar hałasu panującego w lokalu, czy agent monitoring, dostarczający obrazy z kamer, na podstawie których można wnioskować o liczbie wolnych miejsc. W realizacji tego pomysłu, funkcjonalność wspomnianych wcześniej agentów zostanie znacząco uproszczona i będzie polegała na zwracaniu jakiejś losowej wartości na żądanie pomiaru danego parametru.

Opisane wyżej oprogramowanie klienckie oraz baru należy utożsamiać z dwoma konkretnymi typami agentów, których od tej pory będziemy nazywać po prostu klientami lub barami. W następnym rozdziale zostanie wytłumaczona komunikacja między agentami, która ma na celu wyznaczenie miejsca, spełniającego w najlepszym stopniu wymagania danego klienta.

Opis komunikacji pomiędzy agentami



Powyższy rysunek ma na celu zobrazować przebieg komunikacji pomiędzy agentami w systemie. Po uruchomieniu aplikacji, sprecyzowaniu wymagań oraz określeniu maksymalnej odległości, w promieniu której powinien znajdować się bar, przez klienta zostanie rozesłany komunikat do pewnej ilości barów (minimalizacja prawdopodobieństwa wystąpienia błędu komunikacji). Następnie bary rozpoczną rozsyłanie tego komunikatu do pozostałych agentów tego samego typu oraz przeprowadzą między sobą negocjacje, w celu zidentyfikowania najbardziej dopasowanych reprezentantów. Podczas tych negocjacji będą brane pod uwagę między innymi poszczególne preferencje użytkownika czy opinie o danym barze, aby proces ten nie sprowadzał się do porównania tylko jednego czynnika. Aby zagwarantować realny czas znalezienia idealnego baru, zostanie narzucone pewne ograniczenie czasowe, po przekroczeniu którego wszystkie oferty, które nie przegrały negocjacji z innymi agentami zostaną przesłane do aplikacji klienckiej.

Część B

Projekt naszego systemu wieloagentowego postanowiliśmy zrealizować przy pomocy metodologii GAIA. O takim wyborze zdecydowała głównie przystępność metody, brak narzuconych z góry wymagań dotyczących sposobu realizacji oraz możliwość ułatwienia implementacji systemu w środowisku JADE, którego wykorzystanie zostanie rozważone w kolejnej części raportu.

Faza analizy

Identyfikacja ról

Customer - klient poszukujący baru najlepiej dopasowanego do jego preferencji:

- rola odpowiadająca za kontakt z klientem: **CustomerPreferencesManager**.
- rola czuwająca nad komunikacją z barami: **CustomerHandler**.

Bar - lokal zarejestrowany w systemie:

- rola odpowiedzialna za odbiór preferencji klienta i komunikację z innymi barami: **BarOfferManager**.
- rola zarządzająca parametrami baru: **BarParametersManager**.
- rola reprezentująca bar z aktualnie najlepszą ofertą: **BestOfferHolder**.

Bar controllers - zbiór obiektów przy pomocy których ustalane są wartości danych parametrów baru. Mogą być to specjalne mechanizmy w postaci np. sensorów znajdujących się w barze, czy po prostu osoba, określająca wartość czynnika w sposób manualny:

- rola odpowiedzialna za pomiary hałasu w barze: **LoudnessController**.
- rola czuwająca nad zasobami: **ResourcesController**.
- rola reprezentująca system do określania liczby wolnych miejsc: **SeatsController**.

Identyfikacja aktywności / protokołów dla danych ról

- **CustomerPreferencesManager** - ustala preferencje klienta.
- **CustomerHandler** - wysyła preferencje do poszczególnych barów, w celu wyszukania najlepszego i zbiera dla klienta oferty barów.
- **BarOfferManager** - odbiera listę preferencji klienta, przedstawia swoją ofertę i negocjuję z **BestOfferHolder**.
- **BarParametersManager** - ustala parametry baru na podstawie komunikacji z innymi agentami, znajdującymi się wewnątrz baru.
- **BestOfferHolder** - trzyma najlepszą ofertę (która jeszcze nie przegrała w dotychczas przeprowadzonych negocjacjach przez **BestOfferHolder**), negocjuje z innymi barami, zwraca najlepszą ofertę klientowi.
- **LoudnessController** - nadzoruje poziom hałasu w barze.
- **ResourcesController** - nadzoruje zasoby baru, czyli piwo - zarówno pod względem ilościowym, jak i ceny.
- **SeatsController** - nadzoruje liczbę wolnych miejsc w barze.

Model ról

CustomerPreferencesManager

Aktywności:

- SetPreferences - ustawia nowe wartości preferencji klienta.

Protokoły:

- GivePreferences - dostarcza preferencje klienta.

Role Schema: CustomerPreferencesManager
Description: Zadaniem tej roli jest ustalenie preferencji klienta, które posłużą potem do wybrania najlepszego dla niego baru.
Protocols and activities: SetPreferences, GivePreferences
Permissions: **changes** customerPreferences
Responsibilities: Liveness: CustomerPreferencesManager = ((SetPreferences . GivePreferences) | (GivePreferences))+ Safety: true

CustomerHandler

Aktywności:

Protokoły:

- AwaitCall - oczekuje na żądanie klienta.
- SendPreferences - wysyła preferencje klienta do barów.
- AwaitOffers - oczekuje na oferty przesyłane przez bary.
- InformCustomer - informuje klienta o dostarczonych ofertach.

Role Schema: CustomerHandler
Description: Zadaniem tej roli jest przekazanie preferencji do barów i zebranie w ustalonym czasie najlepszych ofert.
Protocols and activities: AwaitCall, SendPreferences, AwaitOffers, InformCustomer
Permissions: **reads** customerPreferences customerDetails offers
Responsibilities: Liveness: CustomerHandler = (AwaitCall . GenerateOffers)$^{\omega}$ GenerateOffers = (SendPreferences . AwaitOffers . InformCustomer) Safety: infoAvailable(customerPreferences, customerDetails, offers)

BarOfferManager

Aktywności:

- CompareOffers - porównuje zaproponowane oferty pod względem dopasowania do dostarczonych preferencji klienta.

Protokoły:

- GetBarParameters - pobiera parametry określające bar.
- AwaitNegotiations - oczekuje na rozpoczęcie negocjacji.
- GetOpponentOffer - pobiera ofertę rywalizującego z nim baru do przeprowadzenia negocjacji.
- SendCounteroffer - wysyła kontrofertę.
- AdmitDefeat - wysłanie komunikatu o braku możliwości przebicia oferty drugiego baru.

Role Schema: BarOfferManager
Description: Zadaniem tej roli jest negocjacja przejęcia tytułu najlepszej oferty (roli BestOfferHolder). Negocjacje odbywają się na podstawie preferencji klienta, oferty reprezentowanego baru oraz aktualnie najlepszej oferty.
Protocols and activities: GetBarParameters, AwaitNegotiations, GetOpponentOffer, CompareOffers, SendCounteroffer, AdmitDefeat
Permissions: **reads** customerPreferences ownOffer bestOffer
Responsibilities: Liveness: BarOfferManager = (GetBarParamaters . AwaitNegotiations . Negotiate)* Negotiate = (GetOpponentOffer . CompareOffers . (SendCounteroffer | [AdmitDefeat]))+ Safety: infoAvailable(ownOffer)

BarParametersManager

Aktywności:

Protokoły:

- GetLoudnessLevel - pobiera informację o poziomie hałasu w barze.
- GetSeatsNumber - pobiera informację o liczbie wolnych miejsc.
- GetResourcesInfo - pobiera informację o stanie zasobów.
- ProduceOffer - generuje ofertę baru na podstawie jego parametrów.

Role Schema: BarParameteresManager
Description: Zadaniem tej roli jest ustalenie parametrów baru i wystawienie na ich podstawie oferty.
Protocols and activities: GetLoudnessLevel, GetSeatsNumber, GetResourcesInfo, ProduceOffer
Permissions: **reads** loudnessLevel resourcesInfo seatsNumber **generates** offer
Responsibilities: Liveness: BarParametersManager = (GetLoudnessLevel . GetSeatsNumber . GetResourcesInfo . ProduceOffer) Safety: infoAvailable(loudnessLevel, resourcesInfo, seatsNumber)

BestOfferHolder

Aktywności:

- CompareOffers - porównuje zaproponowane oferty pod względem dopasowania do dostarczonych preferencji klienta.

Protokoły:

- ProvideBestOffer - dostarcza dotychczas najlepszą ofertę do klienta.
- SetNewBestOffer - ustanawia daną ofertę za aktualnie najlepszą.
- SendOffer - wysyła swoją ofertę do innego baru w celu rozpoczęcia negocjacji.
- AwaitCounteroffer - oczekuje na kontrofertę.
- AdmitDefeat - wysłanie komunikatu o braku możliwości przebicia oferty drugiego baru.

Role Schema: BestOfferHolder
Description: Zadaniem tej roli jest trzymanie najlepszej aktualnie oferty oraz zwrócenie jej w ustalonym czasie CustomerHandler.
Protocols and activities: ProvideBestOffer, SetNewBestOffer, SendOffer, CompareOffers, AwaitCounteroffer, AdmitDefeat
Permissions: **reads** otherOffer bestOffer # own bar offer customerDetails # for ProvideBestOffer customerPreferences
Responsibilities: Liveness: BestOfferHolder = (Negotiate+ . (SetNewBestOffer | ProvideBestOffer)) Negotiate = ((SendOffer . AwaitCounteroffer . CompareOffers)+ . [AdmitDefeat]) Safety: infoAvailable(bestOffer, customerDetails)

LoudnessController

Aktywności:

Protokoły:

- LoudnessLevelRequest - żąda dokonania pomiaru poziomu hałasu w barze.
- LoudnessLevelResponse - dostarcza informację o poziomie hałasu.

Role Schema: LoudnessController
Description: Zadaniem tej roli jest nadzorowanie poziomu hałasu w lokalu.
Protocols and activities: LoudnessLevelRequest, LoudnessLevelResponse
Permissions: **generates** loudnessLevel
Responsibilities: Liveness: LoudnessController = (LoudnessLevelRequest . LoudnessLevelResponse) Safety: true

ResourcesController

Aktywności:

Protokoły:

- ResourcesInfoRequest - żąda informacji o stanie zasobów.
- ResourcesInfoResponse - dostarcza informację o stanie zasobów.

Role Schema: ResourcesController
Description: Zadaniem tej roli jest nadzorowanie ilości poszczególnych piw dostępnych w lokalu oraz ich cen.
Protocols and activities: ResourcesInfoRequest, ResourcesInfoResponse
Permissions: **generates** resourcesInfo
Responsibilities: Liveness: ResourcesController = (ResourcesInfoRequest . ResourcesInfoResponse) Safety: true

SeatsController

Aktywności:

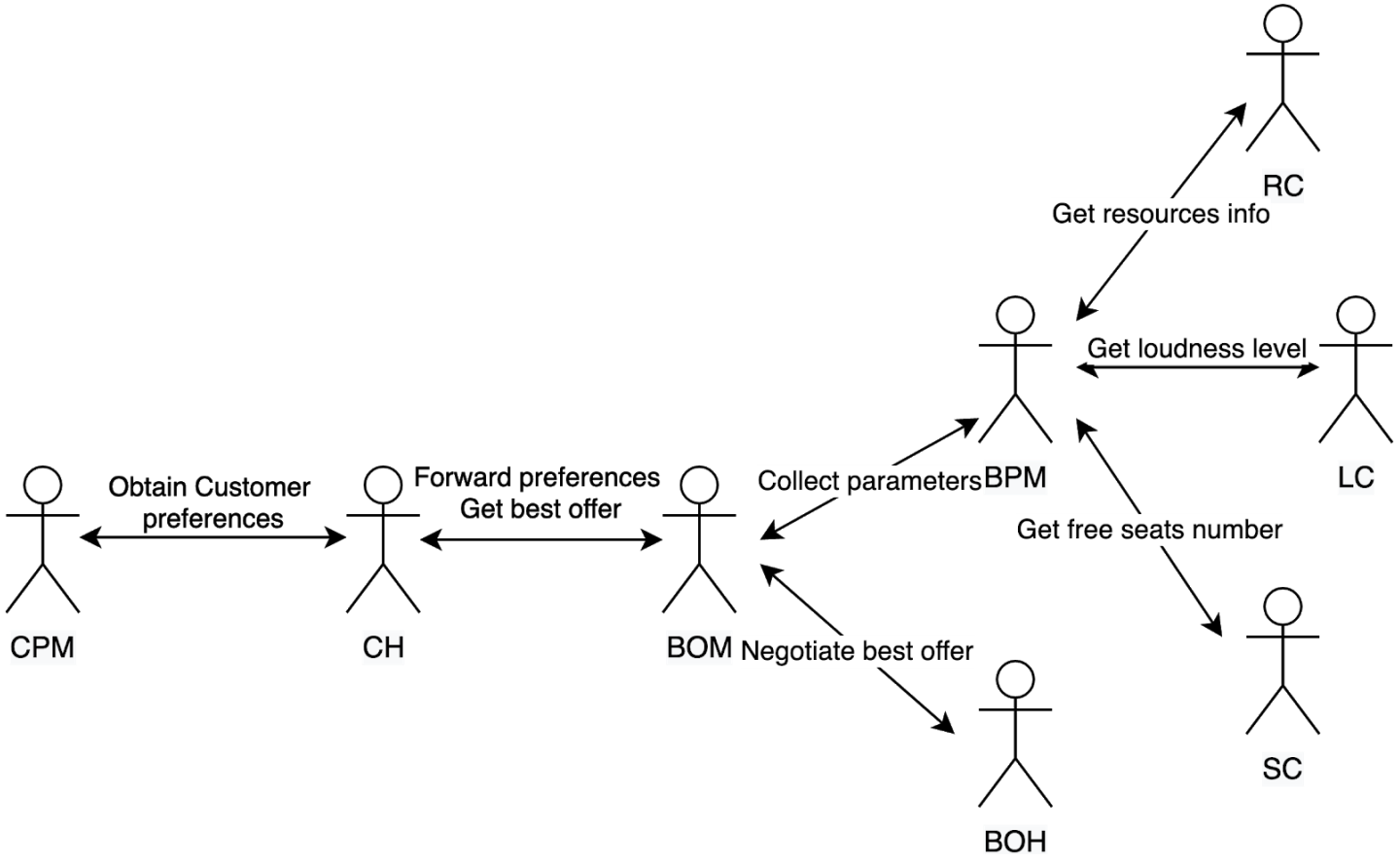
Protokoły:

- SeatsNumberRequest - żąda informacji o liczbie wolnych miejsc w barze.
- SeatsNumberResponse - dostarcza informację o liczbie wolnych miejsc.

Role Schema: SeatsController
Description: Zadaniem tej roli jest nadzorowanie liczby miejsc w lokalu.
Protocols and activities: SeatsNumberRequest, SeatsNumberResponse
Permissions: **generates** seatsNumber
Responsibilities: Liveness: SeatsController = (SeatsNumberRequest . SeatsNumberResponse) Safety: true

Model interakcji

Skróty nazw ról użyte do modelu interakcji:
CPM - CustomerPreferencesManager
CH - CustomerHandler
BOH - BestOfferHolder
BOM - BarOfferManager
BPM - BarParametersManager
SC - SeatsController
LC - LoudnessController
RC - ResourcesController



Diagramy protokołów:

GivePreferences		
CPM	CH	
Przekazanie ustalonych preferencji klienta.		*customerPreferences*

Uwaga do protokołu SendPreferences:
Po tym, jak BOM otrzymuje preferencje klienta od CH, zostaje mu przyznana rola BOH.

SendPreferences		
CH	BOM	
Przekazanie preferencji oraz danych klienta.		*customerPreferences, customerDetails*

SendOffer		
BOH	BOM	
Przekazanie aktualnie najlepszej oferty do kolejnego baru, wraz z preferencjami klienta.		*bestOffer, customerPreferences*

SendCounteroffer		
BOM	BOH	*bestOffer, ownOffer, customerPreferences*
Porównanie ofert i w przypadku posiadania lepszej oferty, wysłanie jej do konkuretna.		*ownOffer*

AdmitDefeat		
BOH	BOM	*bestOffer, otherOffer, customerPreferences*
Porównanie ofert i wysłanie komunikatu o braku możliwości przebicia oferty.		

Uwaga do protokołu SetNewBestOffer:
Bar odpowiedzialny za tę operację po jej wykonaniu utraci rolę BOH. Za to zostanie ona przyznana drugiemu baru, który jest odbiorcą tej komunikacji.

SetNewBestOffer		
BOH	BOM	
Ustanawia ofertę rywalizującego baru za najlepszą.		*customerDetails*

ProvideBestOffer		
BOH	CH	*customerDetails*
Przekazanie informacji o najlepszej ofercie.		*bestOffer*

ProduceOffer		
BPM	BOM	*loudnessLevel* *resourcesInfo* *seatsNumber*
Przekazanie wartości parametrów, w przypadku wielu czujników zbierających dane tego samego parametru uśrednienie i ostatecznie utworzenie oferty.		*offer*

SeatsNumberRequest		
BPM	SC	
Prośba o informację o ilości wolnych miejsc		

SeatsNumberResponse		
SC	BPM	
Przekazanie informacji o ilości wolnych miejsc		*seatsNumber*

LoudnessLevelRequest	
BPM	LC
Prośba o informację o poziom natężenia dźwięku	

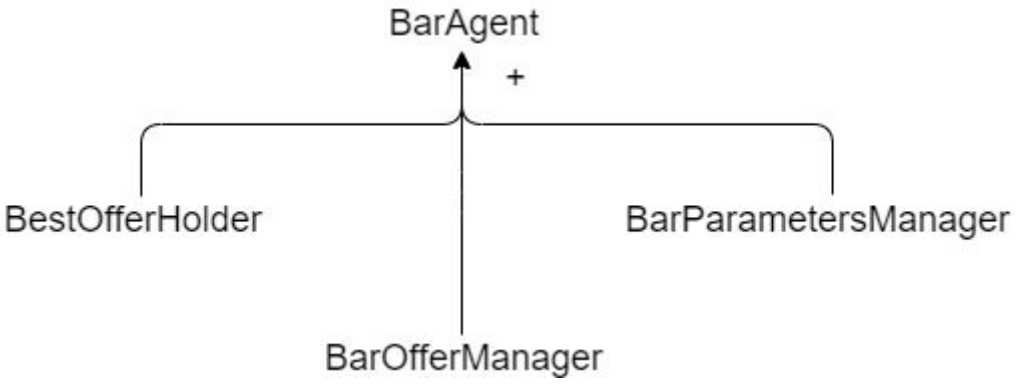
LoudnessLevelResponse	
LC	BPM
Przekazanie informacji o poziomie natężenia dźwięku	*loudnessLevel*

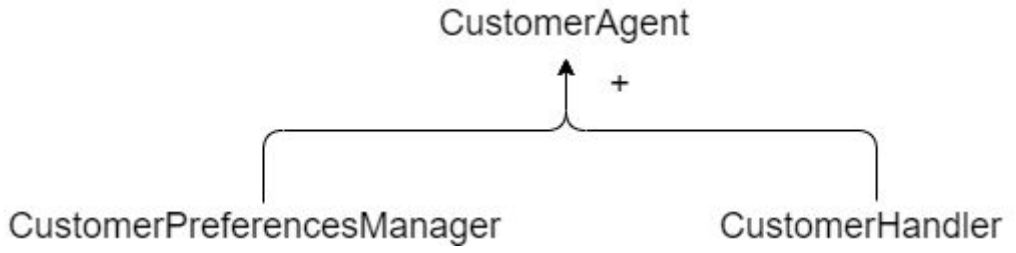
ResourcesInfoRequest	
BPM	RC
Prośba o informację o zasobach baru	

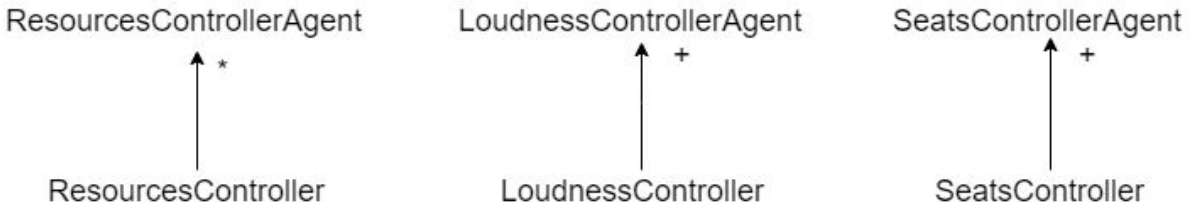
ResourcesInfoResponse	
RC	BPM
Przekazanie informacji o zasobach baru	*resourcesInfo*

Faza projektowania

Model agentów







Wydzielone zostały następujące agenty:

BarAgent - agent zbierający informacje opisujące parametry baru i negocjujący z innymi agentami, w celu wybrania baru najlepiej dopasowanego do preferencji klienta. W przypadku zwycięstwa w negocjacjach przekazuje zwycięską ofertę. Przewidziano wielu agentów, po jednym na bar / lokal.

CustomerAgent - agent pobierający wskazanych przez użytkownika wartości i priorytetów parametrów, przekazujące zapytania do barów, oraz wskazujący użytkownikowi wybrany bar. Przewidzianych zostało wielu agentów, po jednym na użytkownika końcowego.

ResourcesControllerAgent - agent pobierający dane o stanie zasobów w barze (ilość i cena poszczególnych piw) i przekazujący pomiary do BarAgent. Przewidziano wielu

agentów, po jednym na każdy czujnik, czujnik nie jest jednak obligatoryjny (informację, czy cenę danego piwa może wprowadzić ręcznie barman).

LoudnessControllerAgent - agent pobierający dane z decybelomierza i przekazujący pomiary do BarAgent. Przewidziano wielu agentów, po jednym na każdy czujnik.

SeatsControllerAgent - agent pobierający dane z odpowiedniego systemu do analizy ilości wolnych miejsc i przekazujący pomiary do BarAgent. Przewidziano wielu agentów, po jednym na każdy element systemu.

Model usług

Usługa	Wejścia	Wyjścia	Warunki Wstępne	Warunki Końcowe
Obtain Customer Preferences	*customerDetails*	*customerPreferences*	customerPreferences !=NULL customerDetails !=NULL	true
Get Best Offers	*customerDetails customerPreferences*	*bestOffer*	customerPreferences !=NULL customerDetails, offers !=NULL	true
Negotiate Best Offer	*customerPreferences ownOffer bestOffer*	*bestOffer*	customerPreferences !=NULL	true
Monitor Parameters and Produce Offer	*loudnessLevel resourcesInfo seatsNumber*	*offer*	loudnessLevel !=NULL, resourcesInfo !=NULL, seatsNumber !=NULL	true
Monitor Resources		*resourcesInfo*	*true*	*true*
Monitor Loudnesss		*loudnessLevel*	*true*	*true*
Monitor Free Seats		*seatsNumber*	*true*	*true*

Model znajomości

Relacje pomiędzy agentami są przedstawione na poniższym diagramie: