

# SerShare

Raport projektowy, wersja z 20.01.2019

Dziennik zmian:

1. 24.10.2018 - utworzenie dokumentu
2. 25.10.2018 - dodanie Danych projektu
3. 26.10.2018 - dodanie Części A
4. 12.11.2018 - dodanie Części B, modyfikacja Części A (zaznaczona na **żółto**)
5. 09.12.2018 - dodanie Części C, modyfikacja Części B w oparciu o komentarze dr Pałki (zaznaczone na **zielono**)
6. 20.01.2019 - dodanie Części D, modyfikacja Części C w oparciu o komentarze dr Pałki oraz recenzje (zaznaczone na **niebiesko**)

## 1. Dane projektu:

- a. Tytuł projektu: SerShare (dawne MilkShare)
- b. Numer zespołu: Zespół A
- c. Skład z podziałem na role:
  - i. **Maciej Wiraszka** - kierownik projektu + specjalista ds. standardów FIPA
  - ii. **Julita Oltusek** - specjalistka ds. algorytmów agentowych
  - iii. **Szymon Borodziuk** - specjalista ds. architektury
  - iv. **Michał Starosta** - specjalista ds. jakości agentów
  - v. **Jędrzej Kalisiak** - specjalista ds. implementacji
  - vi. **Cezary Modzelewski** - specjalista ds. implementacji
- d. Repozytorium: <https://github.com/Zwirek009/SerShare>

## 2. Część A: Identyfikacja problemu

### a. Problem

Ludzie marnują jedzenie, kupując produkty spożywcze, które wykorzystują nie do końca. Poza tym, marnują czas na robienie zakupów. Dodatkowo zanieczyszczają środowisko, bo każdy jedzie po zakupy, często własnym samochodem.

### b. Propozycja rozwiązania

System zarządzania zasobami żywieniowymi w społeczności lokalnej (np. w budynku lub na osiedlu). System wykorzystuje połączone, inteligentne lodówki, które monitorują własną zawartość i na tej podstawie automatycznie zamawiają jedzenie przez internet. Produkty dostarczane są zbiorczo, okresowo. System umożliwia także wymianę produktów z

sąsiadem, w przypadku kiedy nam coś zostało (np. wykorzystaliśmy niecałą zawartość kartonu z mlekiem), a sąsiadowi akurat było to potrzebne.

### c. Koncepcja rozwiązania

#### Rodzaje agentów:

1. **Fridge Agent (agent lodówkowy)** - agent zajmujący się zapewnieniem wszystkich wymaganych towarów w lodówce i obsługą mechanizmów dzielenia się produktami.

##### Zadania:

- Monitoring aktualnego stanu jedzenia w lodówce
- Tworzenie planu zawartości danej lodówki
- Obsługa mechanizmu udostępniania jedzenia

##### Komunikacja z:

- Mobile Agent - uzyskiwanie informacji o planowanym zapotrzebowaniu na produkty
- Storekeeper Agent - wymiana informacji o przewidywanej ilości produktów w lodówce w czasie, informowanie o aktualnych zapasach.
- Fridge Agent - uzgadnianie wymian produktów.

##### Cel:

- Agent dąży do tego aby lodówka posiadała wymagane produkty

##### Umiejscowienie:

- Agent lodówkowy będzie zainstalowany na każdej lodówce w ilości jeden

2. **Mobile Agent (agent mobilny)** - agent wydzielony tylko do wprowadzania zmian w planie jedzeniowym lub do zgłaszania potrzeb in-time

##### Zadania:

- Pobieranie od użytkownika zmian planu jedzeniowego
- Pobieranie od użytkownika zgłoszeń in-time

##### Komunikacja z:

- Fridge Agent - wysłanie informacji o zmianach planu jedzeniowego i zgłoszeń in-time

##### Cel:

- Przekazanie agentowi Fridge informacji o planach zakupowych oraz nagłych potrzebach

##### Umiejscowienie:

- Urządzenia mobilne

3. **Storekeeper Agent** - agent zajmujący planowaniem zakupów produktów na podstawie stanu lodówki oraz planów żywieniowych.

Zadania:

- Zbieranie planów żywieniowych
- Planowanie zakupów
- Zbieranie informacji o stanie lodówki

Komunikacja z:

- Fridge Agent - wymiana informacji o aktualnych zasobach
- Merchant Agent - składanie zamówień na zakupy
- Mobile Agent - pobieranie informacji o planach żywieniowych

Cel:

- Zapewnienie dostępności produktów ustawionych w planach żywieniowych

Umiejscowienie:

- Każda lodówka

4. **Merchant Agent** - agent reprezentujący sklep

Zadania:

- Zbieranie zamówień od wielu Storekeeper Agent
- Planowanie optymalnych zakupów
- Zamawianie produktów w sklepach (przy tym wybór sklepów)

Kontakty:

- Storekeeper Agent - wymiana informacji o aktualnych potrzebach i planowanych dostawach jedzenia

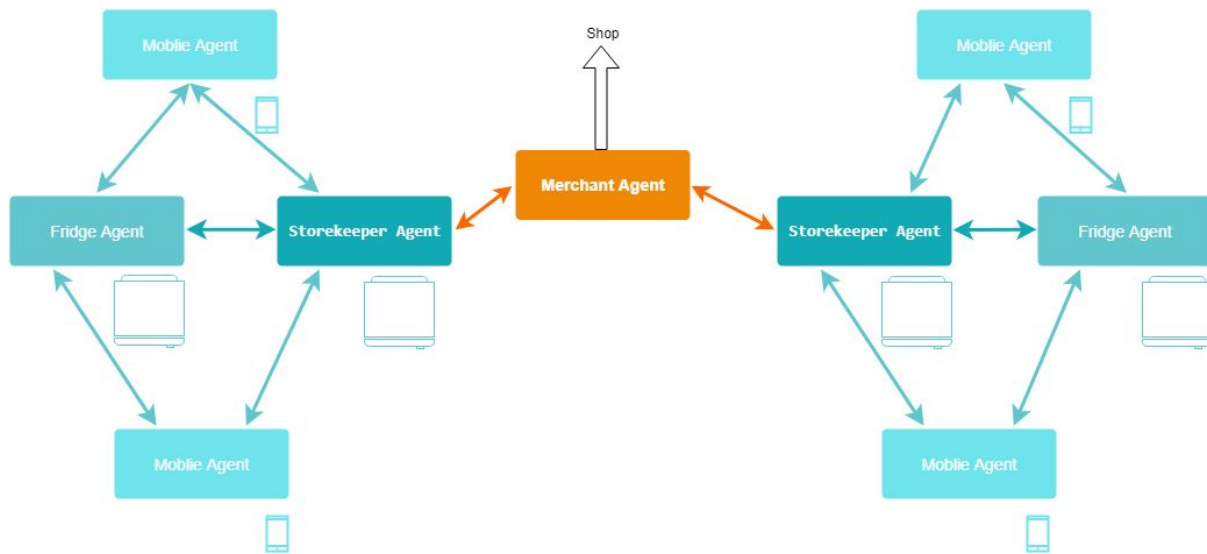
Cel:

- Planowanie jak najrzadszych i najlepiej dopasowanych czasowo zakupów na podstawie zamówień Storekeeper Agentów

Umiejscowienie:

- Chmura

## Architektura rozwiązania:



### 3. Część B: Projekt systemu wieloagentowego (w metodyce GAIA)

#### a. Identyfikacja ról

- i. **FridgeStateController** - monitorowanie stanu lodówki, czyli ilości produktów w jej wnętrzu.
- ii. **ProductDemandPlanner** - planowanie zapotrzebowania na produkty.
- iii. **MinimumSupplyGuard** - dbanie o zapewnienia produktów dla lodówki w krótkim okresie czasu.
- iv. **SharingHandler** - zarządzanie wymianą produktów pomiędzy klientami.
- v. **OrderManager** - zbieranie zamówień i ich optymalizacja.
- vi. **Customer** - układanie planów żywieniowych.

#### b. Model ról

##### ***FridgeStateController***

###### Aktywności:

- CheckFridgeInternals - pobieranie danych z lodówki za pomocą czujników.

###### Protokoły:

- GetFridgeInternalStateRequest- zapytanie o stan zawartości lodówki.
- SendFridgeInternalStateResponse - informowanie o aktualnym stanie zawartości lodówki.

Role schema: <i>FridgeStateController</i>
Description: Zadaniem tej roli jest monitorowanie stanu lodówki, czyli ilości produktów oraz terminu ich ważności, a także informowanie o swoim stanie.
Protocols and Activities: GetFridgeInternalStateRequest, <u>CheckFridgeInternals</u> , SendFridgeInternalStateResponse
Permissions: <b>generates:</b> fridge_internal_state - informacja o produktach w lodówce
Responsibilities: Liveness: <i>FridgeStateController</i> = (GetFridgeInternalStateRequest . <u>CheckFridgeInternals</u> . SendFridgeInternalStateResponse ) <sup>w</sup> Safety: <ul style="list-style-type: none"><li>• true</li></ul>

## ***ProductDemandPlanner***

Aktywności:

- EstimateFridgeStatePlan - estymowanie zapasów lodówki w czasie, z uwzględnieniem planów żywieniowych właścicieli.

Protokoły:

- GetEstimatedFridgeStatePlanRequest - odbieranie żądania przekazania planu lodówkowego.
- SendEstimatedFridgeStatePlanResponse - przekazanie planu lodówkowego.
- SendFridgeInternalStateRequest - pytanie o aktualny stan produktów w lodówce.
- SendFoodPlanRequest - pytanie o plany żywieniowe właścicieli.

Role schema: <i>ProductDemandPlanner</i>
Description: Zadaniem tej roli jest wyznaczanie zapotrzebowania (zakupy) na produkty na podstawie planów żywieniowych i obecnego stanu lodówki. Określa zarówno spodziewane braki produktów jak i potencjalne ich zapasy.
Protocols and Activities: GetEstimatedFridgeStatePlanRequest, SendFridgeInternalStateRequest , SendFoodPlanRequest , <u>EstimateFutureFridgeStatePlan</u> , SendEstimatedFridgeStatePlanResponse
Permissions: <b>reads:</b> fridge_internal_state - informacja o dostępnych produktach food_plans - plany żywieniowe od wszystkich właścicieli lodówki <b>generates:</b> future_fridge_state_plan - informacja o zapasach i brakach produktów w lodówce
Responsibilities: Liveness: <i>ProductDemandPlanner</i> = (GetEstimatedFridgeStatePlanRequest . SendFridgeInternalStateRequest . SendFoodPlanRequest * . <u>EstimateFridgeStatePlan</u> . SendEstimatedFridgeStatePlanResponse) <sup>w</sup>  Safety: • <b>true</b>

## ***MinimumSupplyGuard***

Aktywności:

- CalculateShortTermProductDemand - wyznaczenie zapotrzebowania na najbliższy czas (np. dzień)

Protokoły:

- SendEstimatedFridgeStatePlanRequest - zapytanie do *ProductDemandPlanner* o plan zapotrzebowania
- SendProductShareRequest - zapytanie do sąsiadów o pożyczenie produktów.

Role schema: <i>MinimumSupplyGuard</i>
Description: Zadaniem tej roli jest zapewnienie produktów potrzebnych w najbliższym czasie, stosując mechanizm pożyczania.
Protocols and Activities: SendEstimatedFridgeStatePlanRequest, <u>CalculateShortTermProductDemand</u> , SendProductShareRequest
Permissions: <b>reads:</b> future_fridge_state_plan - informacja o zapasach i brakach produktów w lodówce <b>generates:</b> short_term_product_demand - aktualne zapotrzebowanie na produkty
Responsibilities: Liveness: $MinimumSupplyGuard = ( SendEstimatedFridgeStatePlanRequest . \underline{CalculateShortTermProductDemand} . \underline{SendProductShareRequest}^* )^\omega$  Safety: <b>true</b>

## SharingHandler

Aktywności:

- CheckIfCanShare - sprawdzenie czy można pożyczyć produkt bez naruszenia swojego planu.

Protokoły:

- GetProductShareRequest - oczekiwanie na żądanie produktu.
- SendEstimatedFridgeStatePlanRequest - zapytanie o plan stanu lodówki.
- SendCustomerShareRequest - prośba o podzielenie się produktem.
- SendProductShareResponse - informacja o produktach, które udało się uzyskać.

Role schema: <i>SharingHandler</i>
Description: Zadaniem tej roli jest zadbanie o udostępnienie produktów nadmiarowych, o ile klient wyrazi na to zgodę.
Protocols and Activities: GetProductShareRequest , <u>CheckIfCanShare</u> , GetEstimatedFridgeStatePlan, SendCustomerShareRequest, SendProductShareResponse
Permissions: <b>reads:</b> short_term_product_demand - aktualne zapotrzebowanie na produkty customer_share_response - zgoda/odmowa właściciela future_fridge_state_plan - informacja o zapasach i brakach produktów w lodówce <b>generates:</b> fridge_state_balance - informacja o ilości produktów do wymiany, products_shared - informacja o produktach, które zostaną wymienione
Responsibilities: Liveness: SharingHandler = (GetProductShareRequest . CheckSharePossible . SendProductShareResponse) CheckSharePossible = (SendEstimatedFridgeStatePlanRequest . <u>CheckIfCanShare</u> . [SendCustomerShareRequest]) Safety: fridge_state_balance <= 0 ⇒ products_shared = EMPTY customer_share_response = false ⇒ products_shared = EMPTY



## ***OrderManager***

### Aktywności:

- PrepareMassOrder - przygotowanie zamówienie, wraz z podziałem produktów na odbiorców.
- OrderProducts - zamawianie produktów ze sklepu wraz z podziałem dla kuriera.

### Protokoły:

- SendEstimatedFrigdeStatePlanRequest - prośba o przesłanie planów lodówkowych.

Role schema: <i>OrderManager</i>
Description: Zadaniem tej roli jest planowanie zamówień produktów ze sklepu oraz ich optymalizacja.
Protocols and Activities: GetEstimatedFrigdeStatePlan, <u>PrepareMassOrder</u> , <u>OrderProducts</u>
Permissions: <b>reads:</b> future_fridge_state_plan - informacja o zapasach i brakach produktów w lodówce <b>generates:</b> order_list - lista z zamówieniem
Responsibilities: Liveness: $OrderManager = (SendEstimatedFrigdeStatePlanRequest^* . \underline{PrepareMassOrder} . \underline{OrderProducts})^\omega (per-day)$ Safety: <b>true</b>

## Customer

### Aktywności:

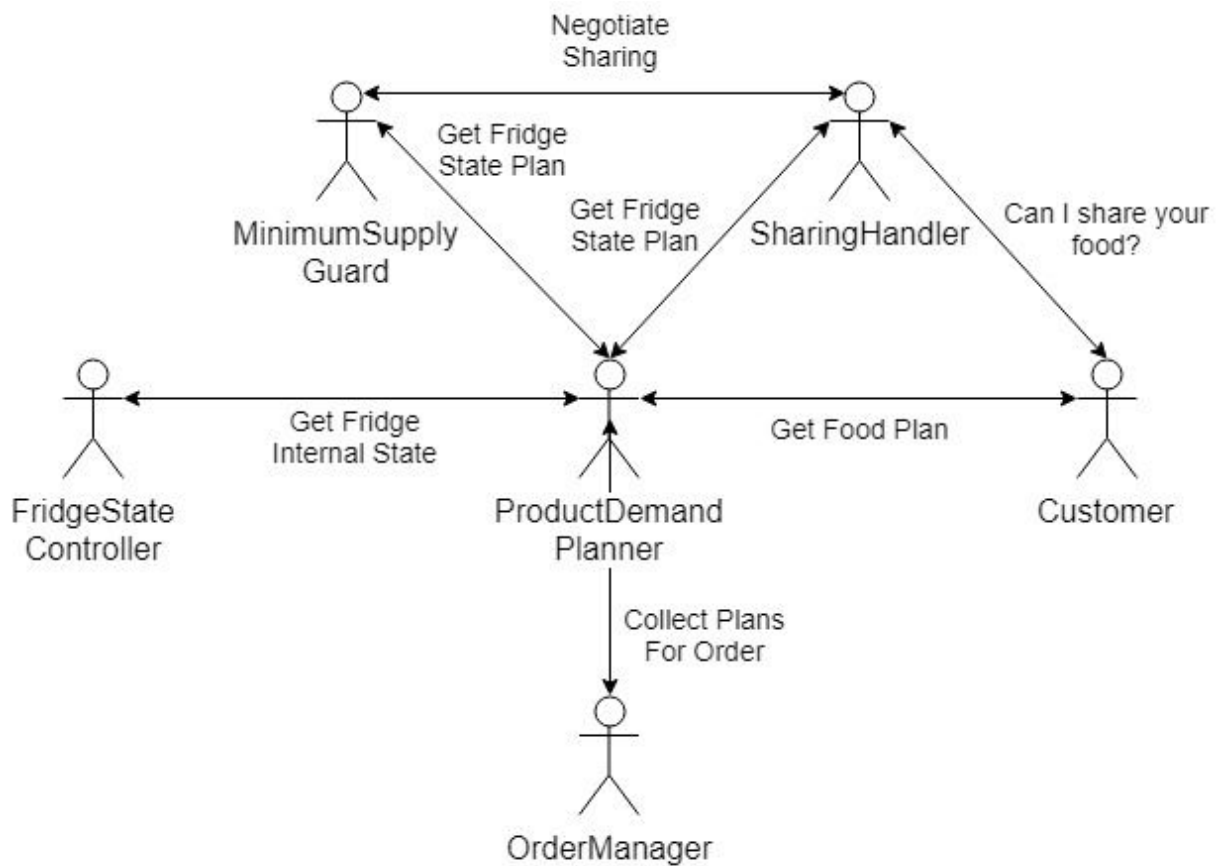
- CheckFoodPlan - pobieranie planu żywieniowego użytkownika
- CheckUserAgreement - pytanie użytkownika o pozwolenie

### Protokoły:

- GetShareRequest - czekanie na zapytanie o zgodę na podzielenie się produktami.
- AcceptOrRefuseShare - deklaracja zgody na podzielenie się produktem.
- GetFoodPlanRequest - czekanie na żądanie przesłania planu
- SendFoodPlanResponse - przesłanie planu zapotrzebowania na produkty.

Role schema: <i>Customer</i>
Description: Zadaniem tej roli jest odpowiadanie na zapytanie o plan żywieniowy właściciela i odpowiadanie na zapytanie o podzielenie się produktem.
Protocols and Activities: GetFoodPlanRequest, <u>CheckFoodPlan</u> , SendFoodPlanResponse , GetShareRequest, <u>CheckUserAgreement</u> , AcceptOrRefuseShare
Permissions: <b>generates:</b> customer_shared_response - zgoda/odmowa użytkownika <b>reads:</b> food_plan - plan żywieniowy
Responsibilities: Liveness: $Customer = (SendFoodPlan \parallel SendAgreementForShare)^{\omega}$ $SendFoodPlan = (GetFoodPlanRequest . \underline{CheckFoodPlan} .$ $SendFoodPlanResponse )$ $SendAgreementForShare = (GetShareRequest .$ $\underline{CheckUserAgreement} . AcceptOrRefuseShare )$  Safety: <b>true</b>

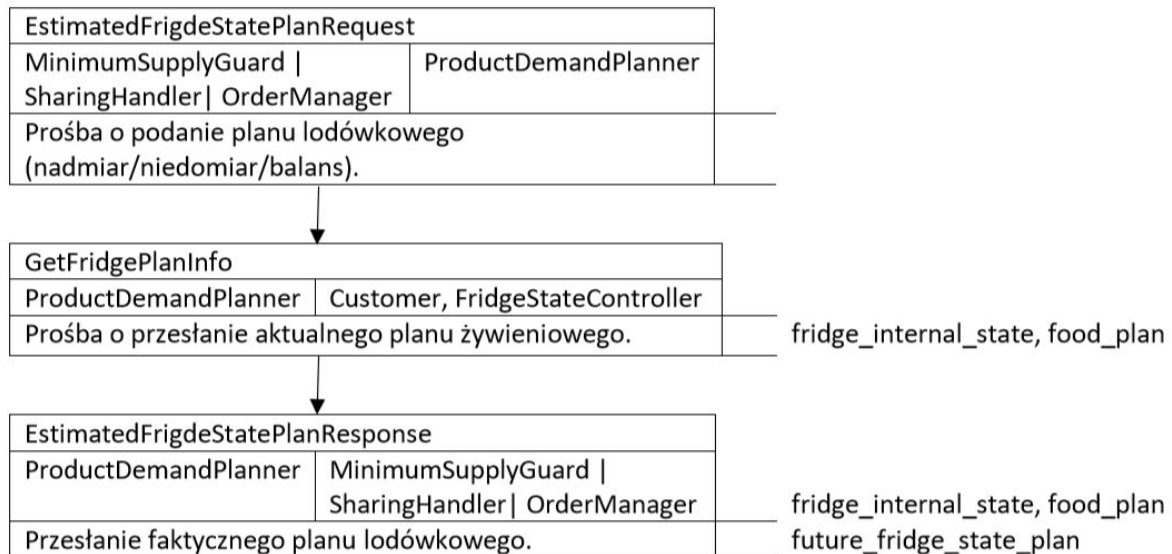
### c. Model interakcji



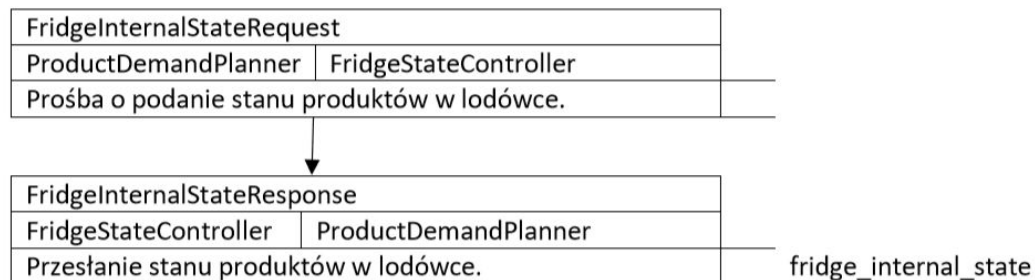
Z powyższego schematu przedstawiającego model interakcji widać, że kluczową rolą w komunikacji i odpowiednim działaniu systemu *SerShare* jest *ProductDemandPlanner*. Wchodzi on w bezpośrednią interakcję z wszystkimi innymi agentami. W związku z tym należy przyłożyć szczególną uwagę do prawidłowej implementacji agenta implementującego rolę *ProductDemandPlanner*.

## Definicje protokołów

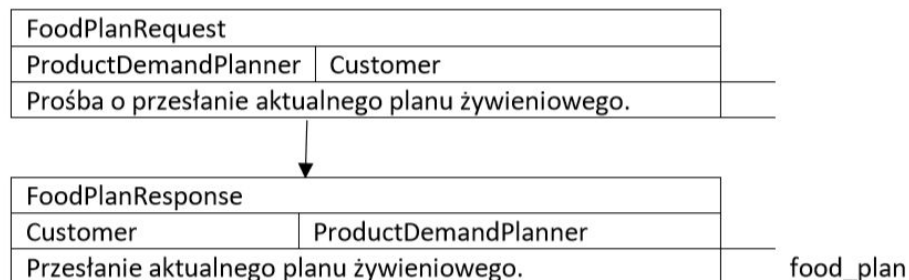
Protokół: GetFridgeStatePlan.



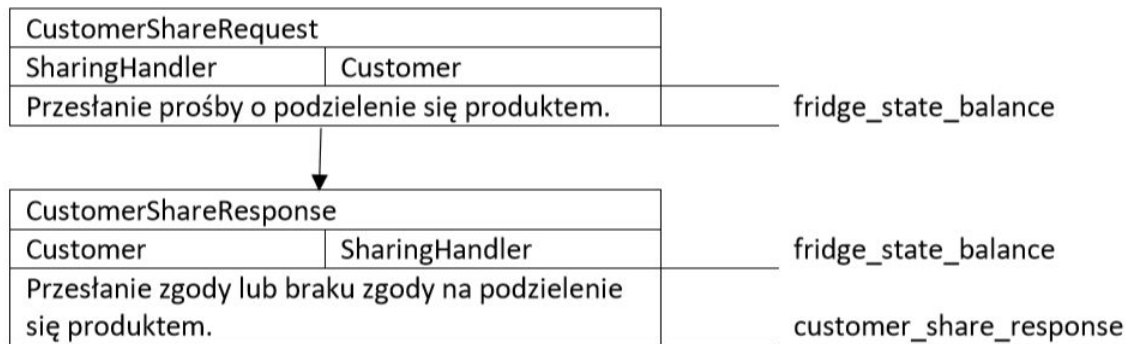
Protokół: GetFridgeInternalState.



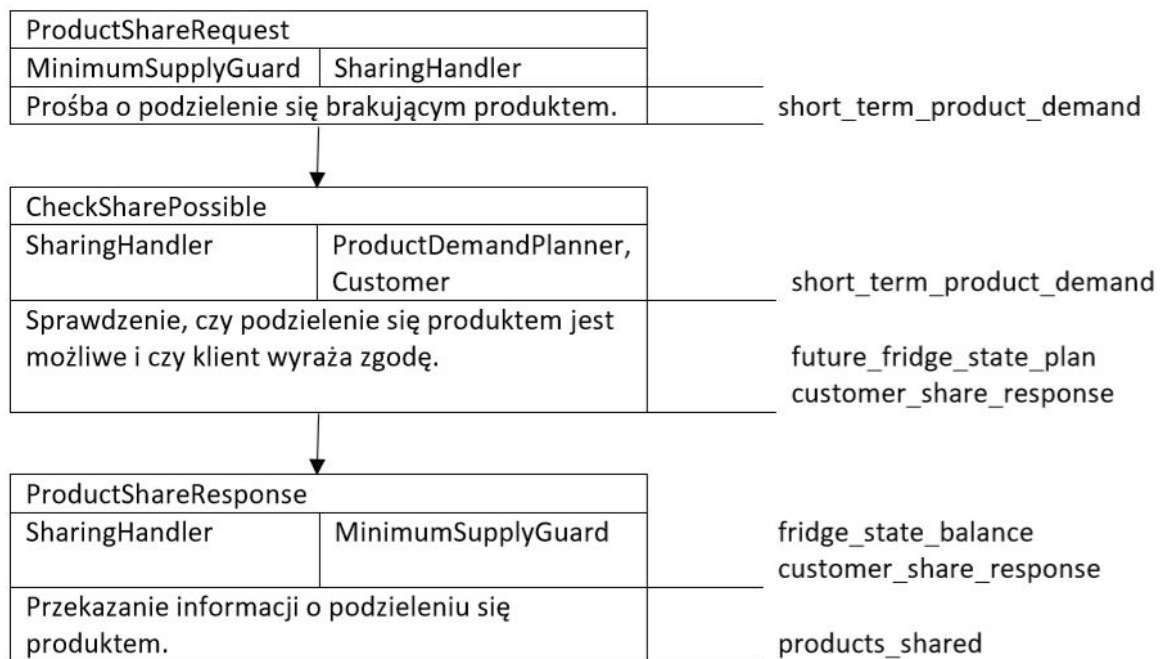
Protokół: GetFoodPlan.



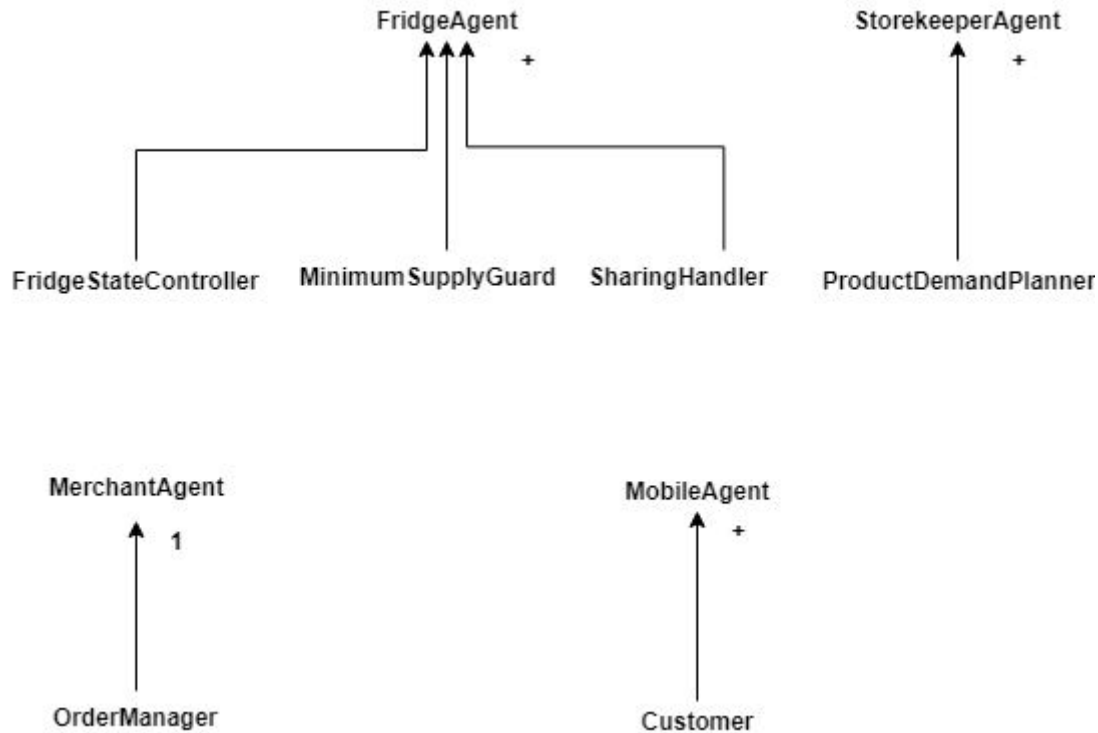
Protokół: GetAgreementForShare.



Protokół: NegotiateShare.



#### d. Model agentów



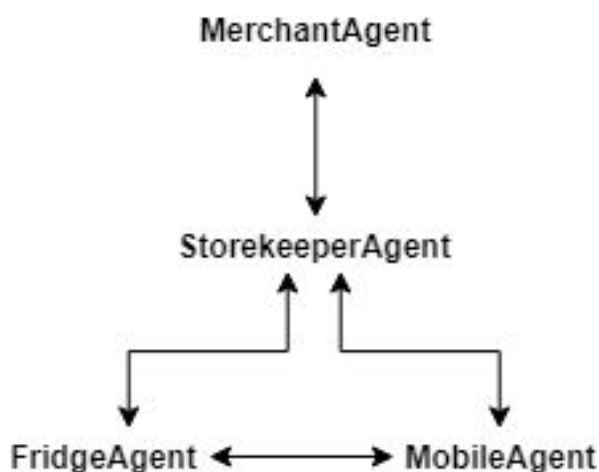
Z powyższych schematów modelu agentów widać, że najbardziej złożonym w kontekście ilości implementowanych ról jest *FridgeAgent*. Wybór taki podyktowany jest faktem, iż role te realizują akcje bezpośrednio powiązane z produktami znajdującymi się w danym momencie w lodówce, którą obsługuje *FridgeAgent*.

Zgodnie z wyszczególnieniem *ProductDemandPlanner* jako szczególnie istotnej roli w komunikacji ze wszystkimi innymi rolami (patrz 2c. Model interakcji), *StorekeeperAgent* implementuje wyłącznie tą rolę, dzięki czemu jego zachowania oraz cele będą nakierowane na realizację wyłącznie odpowiedzialności tej kluczowej roli.

### e. Model usług

Usługa	Wejścia	Wyjścia	Warunki wstępne	Warunki końcowe
ustalanie planu zapotrzebowania produktów	fridge_internal_state food_plans	future_fridge_state_plan	food_plans != NULL	future_fridge_state_plan != NULL
sprawdzanie stanu produktów	wyniki z kontrolerów	fridge_internal_state	true	true
ustalanie dziennego zapotrzebowania	future_fridge_state_plan	short_term_product_demand	true	short_term_product_demand != NULL
udostępnianie produktów	short_term_product_demand customer_share_response future_fridge_state_plan	fridge_state_balance products_shared	fridge_state_balance <= 0 ⇒ products_shared = EMPTY  customer_share_response = false ⇒ products_shared = EMPTY	fridge_state_balance <= 0 ⇒ products_shared = EMPTY  customer_share_response = false ⇒ products_shared = EMPTY
przygotowanie zamówienia	future_fridge_state_plan	order_list	true	order_list != NULL
udostępnienie planu żywieniowego	-	food_plan	true	food_plan != NULL
ustalenie pozwolenia na udostępnianie	-	customer_share_response	true	true

## f. Model znajomości



Na powyższym schemacie widać, że *StorekeeperAgent* jest agentem integrującym wszystkie inne agenty. Ponadto, dzięki jego istnieniu, nie ma potrzeby aby *FridgeAgent* oraz *MobileAgent* komunikowały się bezpośrednio z *MerchantAgent*.

## Część C: Opis implementacji systemu

### a. Framework użyty do implementacji systemu

Do implementacji systemu wybraliśmy framework JADE (Java Agent DEvelopment Framework) - wersja 4.50, język programowania Java.

Głównymi motywacjami tego wyboru były:

- implementacja systemu w JADE odbywa się w języku Java, który wszyscy w zespole znamy, dzięki czemu możemy skupić się na implementacji samego systemu wieloagentowego, a nie poznawaniu nowego języka programowania
- JADE jest bardzo popularnym, dobrze udokumentowanym frameworkiem agentowym, dzięki czemu w przypadku problemów możemy w łatwy sposób uzyskać wsparcie przeglądając dokumentację lub serwisy społeczności developerów JADE
- JADE jest zgodny ze standardami FIPA, dzięki czemu w łatwy sposób możemy zaimplementować koncepty wieloagentowe w naszym systemie, bez potrzeby tworzenia ich samodzielnie



W celu łatwiejszego procesu konfiguracji środowiska deweloperskiego na różnych komputerach (m.in. instalacja JADE itp.) cały proces ściągania zależności (np. instalacja JADE), kompilacji oraz tworzenia plików wykonywalnych systemu agentowego został zdefiniowany w Mavenie (plik [pom.xml](#)). Dzięki temu możemy implementować system mając wcześniej zainstalowane wyłącznie JDK Javy oraz IDE wspierające Mavena.

## b. Sposób implementacji agentów

Utworzyliśmy klasę *SerShareAgent* - standardowego agenta systemu *SerShare*, która dziedziczy po standardowej klasie *Agent* frameworku JADE i zawiera dodatkowo:

- odwołanie na *ContentManager* systemu
- listę id innych znanych agentów
- definicję *SLCodec* dla komunikatów *SerShare*
- metody wspólne dla wszystkich agentów

Wszystkie agenty systemu dziedziczą po klasie *SerShareAgent*, co ułatwia ich implementację w zakresie ich spójności między sobą i pozwala na niepotrzebne powielanie kodu.

Agenty przesyłają między sobą zserializowane obiekty (serializacja/deserializacja przy użyciu wbudowanego w JADE *SLCodec*). Zapewnia to użycie FIPA SL jako języka treści z punktu widzenia przetwarzania komunikatu przez agenta.

Agenty wykorzystują (przez dziedziczenie) zdefiniowane we frameworku JADE zachowania (*behaviours*), które służą do implementacji aktywności (*activities*) oraz protokołów (*protocols*) zdefiniowanych dla ról (*roles*), które implementują (zgodnie z opisem w rozdziale [Model ról](#)). Agenty zostały podzielone według wcześniej zaproponowanych typów: fridge, merchant, storekeeper i mobile.

**StorekeeperAgent** - agent do generowania planów zakupowych dla poszczególnych lodówek. Posiada zapisane informacje o pobranych planach zakupowych oraz stan produktów w lodówce. Ponadto posiada adresy podłączonych mobile agentów oraz fridge agentów. Na podstawie zebranych planów zakupowych oraz stanu lodówki estymuje zapotrzebowanie i planuje następne zakupy. Wykorzystuje do tego funkcje *estimate()*. Do implementacji zachowań wykorzystuje następujące *behaviours*:

(*abstract*) *MYQueryBehaviours* - specjalne stworzone zachowanie do wysyłania zapytań typu *QUERY\_REF*. Składa się z 2 akcji:

wysłanie zapytania do wszystkich zainteresowanych odbiorców i odbieranie ich odpowiedzi. Akcje te występują kolejno po sobie. Ponowne wysłania zapytania może nastąpić poprzez wykorzystanie funkcji *reset()*. Klasa *MYQueryBehaviours* dostarcza dodatkowo metody do obsługi błędnie skonfigurowanych wiadomości. Aby je obsłużyć można wykorzystać mechanizm podstawowy - wysłanie informacji *not understood* do nadawcy. Dodatkowo można dodać własną obsługę błędów poprzez nadpisanie metody *onErrors()*. Następną wykorzystywaną funkcją jest *validateMessage()*, która służy do walidacji otrzymanych

wiadomości. W podstawowym przypadku przyjmuje wiadomości typu inform. Metodę tą można nadpisać własną walidacją

*SendFoodPlanRequest* - służy do pobierania informacji na temat planów od agentów typu mobile. Nadpisuje klasę MYQueryBehavoiurs. Odpowiedziami są plany produktów poszczególnych mobile agentów, które są zapisywane przez storekeeper agenta do listy planów. Przyjmuje podstawową walidację i obsługę błędów.

*SendFridgeStateRequest* - służy do pobierania informacji na temat stanu produktów od agentów typu fridge. Wykorzystuje klasę MYQueryBehavoiurs. Odpowiedziami są zawartości poszczególnych agentów typu fridge, które są zapisywane przez storekeeper agenta do stanu lodówki. Przyjmuje podstawową walidację i obsługę błędów.

*SendEstimatedFridgeStatePlan* - służy do wysłania wyliczonego planu do merchant agenta. Jest uruchamiany przez funkcję estimate, wtedy gdy agent uzna to za słuszne.

*EstimateFridgeStatePlan* - uruchamia się cyklicznie co określony czas(w modelu symulacyjnym co minutę). Jego zadaniem jest obliczanie nowych wartości planu lodówkowego i gdy minie określony czas(w przypadku symulacyjnym 2 min) i agent uzna to za stosowne wysłanie informacji o nowym planie lodówkowym do merchant agenta. Główną funkcją agenta wywoływaną przez to zachowanie jest estimate.

estimate() - funkcja do planowania nowych zamówień lodówki. Na podstawie zebranych informacji o planach i stanie lodówki określa plan zamówienia wysłanego do MerchantAgent. W naszej implementacji jest to prosty algorytm sprawdzający czy ilość produktów w lodówce jest zgodny z planami zakupowymi wysłanymi przez mobileAgentów. Data sprawdzenia ustalona została na dzień dzisiejszy, czyli stan lodówki jest sprawdzany w stosunku do planów zakupowych na dzisiaj. Jeżeli jakieś plany nie zostały spełnione to są one wysyłane do MerchantAgent.

**MerchantAgent** - agent do zamawiania zakupów. Posiada listę planów zakupowych wraz z adresami agentów (typu storekeeper) którzy go wysłali. Jego głównym zadaniem jest określenie optymalnych zamówień. Wykonuje to dzięki funkcji planOrder która wylicza datę i listę zakupów następnego zamówienia. Ponadto ta metoda po ustaleniu terminu zakupu, dokonuje go za pomocą metody makeOrder(). Oprócz zamawiania zakupów, metoda makeOrder() powiadamia zainteresowanych agentów o zbliżającej się dostawie. Agent ten wykorzystuje następujące zachowania:

*GettingFoodPlans* - cykliczne nasłuchiwanie na wiadomości dostarczające listy zakupów. Po otrzymaniu wiadomości zapisuje je na liście zakupów wraz z adresem wysyłający zamówienie.

*PlanningOrder* - zachowanie wykorzystywane do planowania nowych zamówień. Jest wykonywane cyklicznie(w symulacji co 1min). Jego celem jest wywołanie metody planOrder która określa następne zamówienie.

**FridgeAgent** - agent do pobierania danych o posiadanych produktach z lodówek. Posiada listę produktów, które pobiera z lodówki. Jego głównym zadaniem jest przesyłanie stanu lodówki do agenta StorekeeperAgent. Agent ten wykorzystuje następujące zachowania:

*CheckFridgeInternals* - cyklicznie (testowo co minutę) odpytuje lodówkę o jej stan

*GetFridgeInternalStateResponse* - odpowiedź na pytanie o listę produktów, wysyła zserializowany obiekt agentowi który zapytał w wiadomości o performatywie INFO\_REF. Przechwytuje wyjątki IO.

**MobileAgent** - agent zajmuje się pobieraniem planu żywieniowego w postaci mapy Map<LocalDate,List<FoodPosition>> od użytkownika (mock) i przesyłaniem go do agenta StorekeeperAgent. Agent ten wykorzystuje następujące zachowanie:

*GetFoodPlan* - to zachowanie jest cykliczne i polega na oczekiwaniu na wiadomość od agenta StorekeeperAgent zawierającą performatywę QUERY\_REF z prośbą o przesłanie planu żywieniowego. W wyniku otrzymania takiej wiadomości, plan jest pobierany od użytkownika (co zostało zasymulowane poprzez losowanie kilku dat, produktów wraz z ich ilością), a następnie odsyłany do oczekującego agenta w postaci wiadomości o performatywie INFO\_REF.

### c. Komunikacja

- Zastosowane performatywy
  1. INFORM
  2. QUERY\_REF
  3. INFORM\_REF
  4. NOT\_UNDERSTOOD
- Zastosowane protokoły komunikacyjne

Użyliśmy jednego zdefiniowanego w standardzie FIPA IP protokołu: FIPA Query Interaction Protocol (do przesyłania do agenta StorekeeperAgent stanu lodówki od agenta FridgeAgent i planu żywieniowego od agenta MobileAgent), dodatkowo zaimplementowaliśmy własne, specyficzne dla naszego przypadku. Wszystkie protokoły stworzone za pomocą mechanizmu behaviours.

- Zastosowane języki treści

Komunikacja między agentami odbywa się przy użyciu FIPA Semantic Language (SL).

Główną motywacją wyboru tego języka treści jest to, iż jest on językiem treści zdefiniowanym przez FIPA oraz jest standardowo zaimplementowany we frameworku JADE.

### d. Wykorzystane standardy

Z racji wykorzystywania JADE'a, nasz system jest zgodny ze standardami FIPA:

- FIPA ACL (Agent Communication Language)

- FIPA SL (Semantic Language)
- FIPA CAL (Communicative Act Library)

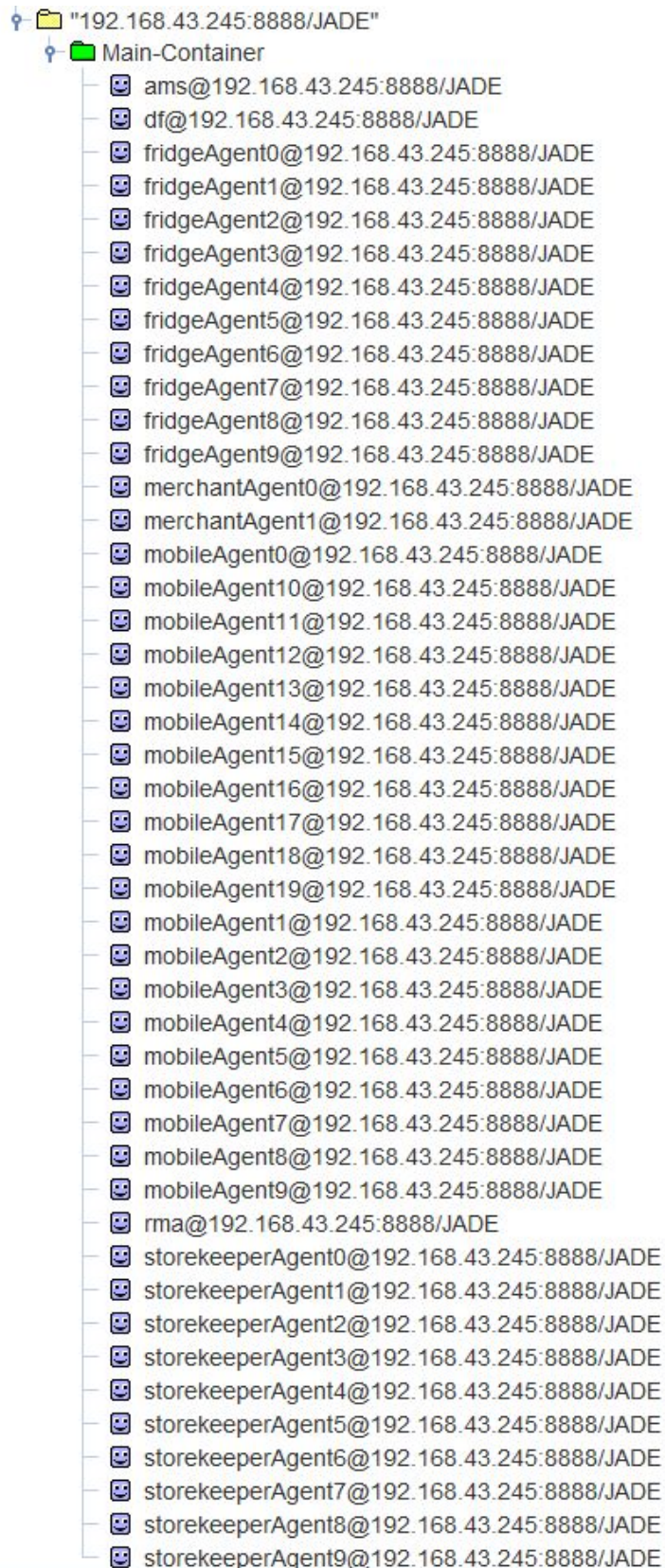
## e. Algorytmy

Wszystkie algorytmy używane w systemie *SerShare* zostały zaprojektowane i zaimplementowane specjalnie do ich zastosowań i nie korzystamy ze standardowych algorytmów agentowych. Opis algorytmów został przedstawiony w rozdziale [Sposób implementacji agentów](#).

## f. Napotkane problemy

- Podczas implementacji równoległych zachowań, w których następowało oczekiwanie na wiadomość, problemem okazało się przechwytywanie wiadomości przez niewłaściwe zachowania. W celu rozwiązania tego problemu użyliśmy szablonów wiadomości, na budowanie których pozwala klasa *jade.lang.acl.MessageTemplate*. Okazuje się, że klasa ta nie filtruje wiadomości w 100%, ponieważ wystąpił jeden przypadek, kiedy pomimo zastosowania filtrowania id konwersacji, została odebrana wiadomość niespełniająca tego warunku. Nie udało się więcej odtworzyć tej sytuacji, dlatego uznaliśmy to jako niezbyt uciążliwą anomalię, spowodowaną prawdopodobnie błędem w bibliotece.
- Początkowo uruchomiliśmy system przy pomocy komend konsolowych (przykładowe utworzenie agenta: "fridge1:agents.fridge.FridgeAgent"). Jednak w przypadku uruchomienia wielu instancji agentów ten sposób nie sprawdza się. Ręczne wpisywanie kilkudziesięciu nazw agentów jest bardzo nieefektywne. Dlatego zdecydowaliśmy się wykorzystać klasę *jade.core.Runtime*. Pozwoliło to na utworzenie kontenera, w którym uruchomienie agentów można było zrealizować w sposób bardziej zautomatyzowany (nazwy tych samych typów agentów wygenerowane w pętli z dodawaniem indeksu iteratora). Tworzone agenty dostają w argumentach nazwy innych agentów, z którymi się komunikują.
- Do tej pory nie został zaimplementowany protokół NegotiateShare. Rozważamy rozwinięcie systemu o ten właśnie protokół w kolejnym etapie projektu.

## g. Przykładowe zrzuty ekranu z działania



Rysunek 1. Zrzut ekranu z JADE Remote Agent Management GUI.



```

INFO: Send cfp with id: food-plan
gru 10, 2018 2:38:47 AM agents.storekeeper.behaviours.MyCFPBbehaviour sendRequest
INFO: Send cfp with id: fridge-state
gru 10, 2018 2:38:47 AM agents.fridge.behaviours.GetFridgeInternalStateResponse action
INFO: Get fridge state request (CFP)
:sender ( agent-identifier :name storekeeperAgent00192.168.43.245:8888/JADE :addresses (sequence http://DESKTOP-5OQQLK0:7778/acc ))
:receiver (set ( agent-identifier :name fridgeAgent00192.168.43.245:8888/JADE :addresses (sequence http://DESKTOP-5OQQLK0:7778/acc )) )
:reply-with cfp1544405927679 :language Java_Serialization :conversation-id fridge-state )
gru 10, 2018 2:38:47 AM agents.mobile.behaviours.GetFoodPlan action
INFO: Received food plan request (CFP)
:sender ( agent-identifier :name storekeeperAgent00192.168.43.245:8888/JADE :addresses (sequence http://DESKTOP-5OQQLK0:7778/acc ))
:receiver (set ( agent-identifier :name mobileAgent00192.168.43.245:8888/JADE :addresses (sequence http://DESKTOP-5OQQLK0:7778/acc )) )
:reply-with cfp1544405927679 :language Java_Serialization :conversation-id food-plan )
gru 10, 2018 2:38:47 AM agents.mobile.behaviours.GetFoodPlan action
INFO: Food plan ready to send.
gru 10, 2018 2:38:47 AM agents.storekeeper.behaviours.SendEstimatedFridgeStatePlan action
INFO: Send estimate fridge state plan request
gru 10, 2018 2:38:47 AM agents.merchant.behaviours.GettingFoodPlans action
INFO: Waiting for plan
gru 10, 2018 2:38:47 AM agents.merchant.behaviours.GettingFoodPlans action
INFO: Get plan
gru 10, 2018 2:38:47 AM agents.fridge.behaviours.GetFridgeInternalStateResponse action
0
gru 10, 2018 2:38:47 AM agents.merchant.behaviours.GettingFoodPlans action
INFO: Waiting for plan
gru 10, 2018 2:38:47 AM agents.mobile.behaviours.GetFoodPlan action
INFO: Sent food plan response FoodPlan[plan=[2018-12-27=[1.5 of Milk, 2.5 of Bread], 2018-12-19=[2.5 of Milk, 4.5 of Bread]]
gru 10, 2018 2:38:47 AM agents.mobile.behaviours.GetFoodPlan action
INFO: Waiting for food plan request.
gru 10, 2018 2:38:47 AM agents.storekeeper.behaviours.SendFoodPlanRequest onGettingReply
INFO: Get plan FoodPlan[plan=[2018-12-27=[1.5 of Milk, 2.5 of Bread], 2018-12-19=[2.5 of Milk, 4.5 of Bread]]
gru 10, 2018 2:38:47 AM agents.storekeeper.behaviours.MyCFPBbehaviour getReply
INFO: Reply with id: food-plan
gru 10, 2018 2:38:47 AM agents.storekeeper.behaviours.SendFridgeStateRequest onGettingReply
INFO: Get fridge state FridgeState[state=[1.0 of Milk, 2.0 of Bread]]
gru 10, 2018 2:38:47 AM agents.storekeeper.behaviours.MyCFPBbehaviour getReply
INFO: Reply with id: fridge-state

```

Rysunek 2. Zrzut ekranu z logami systemu. W ramkach zaznaczone zostały informacje wyświetlane podczas realizacji protokołu GetFoodPlan.

```

INFO: Send cfp with id: food-plan
gru 10, 2018 2:38:47 AM agents.storekeeper.behaviours.MyCFPBbehaviour sendRequest
INFO: Send cfp with id: fridge-state
gru 10, 2018 2:38:47 AM agents.fridge.behaviours.GetFridgeInternalStateResponse action
INFO: Get fridge state request (CFP)
:sender ( agent-identifier :name storekeeperAgent00192.168.43.245:8888/JADE :addresses (sequence http://DESKTOP-5OQQLK0:7778/acc ))
:receiver (set ( agent-identifier :name fridgeAgent00192.168.43.245:8888/JADE :addresses (sequence http://DESKTOP-5OQQLK0:7778/acc )) )
:reply-with cfp1544405927679 :language Java_Serialization :conversation-id fridge-state )
gru 10, 2018 2:38:47 AM agents.mobile.behaviours.GetFoodPlan action
INFO: Received food plan request (CFP)
:sender ( agent-identifier :name storekeeperAgent00192.168.43.245:8888/JADE :addresses (sequence http://DESKTOP-5OQQLK0:7778/acc ))
:receiver (set ( agent-identifier :name mobileAgent00192.168.43.245:8888/JADE :addresses (sequence http://DESKTOP-5OQQLK0:7778/acc )) )
:reply-with cfp1544405927679 :language Java_Serialization :conversation-id food-plan )
gru 10, 2018 2:38:47 AM agents.mobile.behaviours.GetFoodPlan action
INFO: Food plan ready to send.
gru 10, 2018 2:38:47 AM agents.storekeeper.behaviours.SendEstimatedFridgeStatePlan action
INFO: Send estimate fridge state plan request
gru 10, 2018 2:38:47 AM agents.merchant.behaviours.GettingFoodPlans action
INFO: Waiting for plan
gru 10, 2018 2:38:47 AM agents.merchant.behaviours.GettingFoodPlans action
INFO: Get plan
gru 10, 2018 2:38:47 AM agents.fridge.behaviours.GetFridgeInternalStateResponse action
0
gru 10, 2018 2:38:47 AM agents.merchant.behaviours.GettingFoodPlans action
INFO: Waiting for plan
gru 10, 2018 2:38:47 AM agents.mobile.behaviours.GetFoodPlan action
INFO: Sent food plan response FoodPlan[plan=[2018-12-27=[1.5 of Milk, 2.5 of Bread], 2018-12-19=[2.5 of Milk, 4.5 of Bread]]
gru 10, 2018 2:38:47 AM agents.mobile.behaviours.GetFoodPlan action
INFO: Waiting for food plan request.
gru 10, 2018 2:38:47 AM agents.storekeeper.behaviours.SendFoodPlanRequest onGettingReply
INFO: Get plan FoodPlan[plan=[2018-12-27=[1.5 of Milk, 2.5 of Bread], 2018-12-19=[2.5 of Milk, 4.5 of Bread]]
gru 10, 2018 2:38:47 AM agents.storekeeper.behaviours.MyCFPBbehaviour getReply
INFO: Reply with id: food-plan
gru 10, 2018 2:38:47 AM agents.storekeeper.behaviours.SendFridgeStateRequest onGettingReply
INFO: Get fridge state FridgeState[state=[1.0 of Milk, 2.0 of Bread]]
gru 10, 2018 2:38:47 AM agents.storekeeper.behaviours.MyCFPBbehaviour getReply
INFO: Reply with id: fridge-state

```

Rysunek 3. Zrzut ekranu z logami systemu. W ramkach zaznaczone zostały informacje wyświetlane podczas realizacji protokołu GetFridgeInternalState.

## Część D: Opis integracji systemu

### a. Opis całego systemu

Pełny opis systemu, zaktualizowany o poprawki zgłoszone przez dr Pałkę oraz recenzjach znajduje się w Częściach A-C.

### b. Opis braków

- Nie udało nam się wykonać implementacji protokołu *NegotiateShare* (zależny od niego protokół *GetAgreementForShare* nie jest wykorzystywany w systemie). Powodem tego braku był natłok obowiązków na uczelni pod koniec semestru.
- System nie posiada przyjaznego interfejsu użytkownika oraz integracji z elementami zewnętrznymi (lodówki, sklepy itp.), pozwalających na jego użycie w życiu codziennym.

### c. Napotkane problemy

Opis napotkanych problemów znajduje się w punkcie [Napotkane problemy](#).

### d. Opis i wyniki testów systemu

Testy były przeprowadzane manualnie, równoległe z rozwijaniem nowych funkcjonalności systemu.

Wyniki przeprowadzonego przykładowego testu systemu (jednego cyklu zamówień, od złożenia planów żywieniowych i stanów lodówki do realizacji zamówienia) dla finalnej wersji kodu źródłowego przedstawiono poniżej.

Uruchomione zostały następujące agenty:

- MerchantAgent - 1
- StorekeeperAgent - 1
- FridgeAgent - 1
- MobileAgent - 2

1. Agenty FridgeAgent oraz MobileAgent oczekują na zapytania od StorekeeperAgent, a MerchantAgent na plan żywieniowy od StorekeeperAgent.

```
sty 21, 2019 2:15:56 AM agents.merchant.behaviours.GettingFoodPlans action
INFO: Waiting for plan
sty 21, 2019 2:15:56 AM agents.mobile.behaviours.GetFoodPlan action
INFO: Waiting for food plan request.
sty 21, 2019 2:15:56 AM agents.fridge.behaviours.GetFridgeInternalStateResponse action
INFO: Waiting for fridge state request.
sty 21, 2019 2:15:56 AM agents.mobile.behaviours.GetFoodPlan action
INFO: Waiting for food plan request.
```

2. StorekeeperAgent wysyła zapytanie QUERY\_REF do agentów MobileAgent o plany żywieniowe. MobileAgent otrzymuje zapytanie i odpowiada INFO\_REF. Storekeeper dostaje zamówienie na 20 stycznia: 1.5 litra mleka i 3.5 bochenka chleba.

```
sty 21, 2019 2:15:56 AM agents.mobile.behaviours.GetFoodPlan action
INFO: Waiting for food plan request.
sty 21, 2019 2:15:56 AM agents.storekeeper.behaviours.MyQueryBehaviour sendRequest
INFO: Send query_ref with id: food-plan
sty 21, 2019 2:15:56 AM agents.mobile.behaviours.GetFoodPlan action
INFO: Received food plan request (QUERY-REF
:sender ( agent-identifier :name storekeeperAgent0@192.168.0.103:8888/JADE :addresses (sequence http://DESKTOP-5OQQLK0:7778/acc ))
:receiver (set ( agent-identifier :name mobileAgent0@192.168.0.103:8888/JADE :addresses (sequence http://DESKTOP-5OQQLK0:7778/acc ))
:reply-with query_ref1548033356976 :language Java_Serialization :conversation-id food-plan )
sty 21, 2019 2:15:56 AM agents.mobile.behaviours.GetFoodPlan action
```

```
sty 21, 2019 2:15:56 AM agents.storekeeper.behaviours.MyQueryBehaviour action
INFO: Send food-plan request
sty 21, 2019 2:15:57 AM agents.mobile.behaviours.GetFoodPlan action
INFO: Sent food plan response FoodPlan(plan={2019-01-20=[1.5 of Milk, 3.5 of Bread]})
```

3. StorekeeperAgent wysyła zapytanie QUERY\_REF do agenta FridgeAgent o stan lodówki. FridgeAgent otrzymuje zapytanie i odpowiada INFO\_REF. Storekeeper otrzymuje pusty stan lodówki.

```
sty 21, 2019 2:15:57 AM agents.storekeeper.behaviours.MyQueryBehaviour sendRequest
INFO: Send query_ref with id: fridge-state
sty 21, 2019 2:15:57 AM agents.storekeeper.behaviours.MyQueryBehaviour action
INFO: Send fridge-state request
sty 21, 2019 2:15:57 AM agents.fridge.behaviours.GetFridgeInternalStateResponse action
INFO: Waiting for fridge state request.
sty 21, 2019 2:15:57 AM agents.fridge.behaviours.GetFridgeInternalStateResponse action
INFO: Get fridge state request (QUERY-REF
:sender ( agent-identifier :name storekeeperAgent0@192.168.0.103:8888/JADE :addresses (sequence http://DESKTOP-5OQQLK0:7778/acc ))
:receiver (set ( agent-identifier :name fridgeAgent0@192.168.0.103:8888/JADE :addresses (sequence http://DESKTOP-5OQQLK0:7778/acc ))
:reply-with query_ref1548033357096 :language Java_Serialization :conversation-id fridge-state )
sty 21, 2019 2:15:57 AM agents.fridge.behaviours.GetFridgeInternalStateResponse action
INFO: Send fridge state response FridgeState[state=[])
sty 21, 2019 2:15:57 AM agents.storekeeper.behaviours.SendFoodPlanRequest onGettingReply
```

```
INFO: Get plan FoodPlan(plan={2019-01-20=[1.5 of Milk, 3.5 of Bread]})
sty 21, 2019 2:15:57 AM agents.storekeeper.behaviours.MyQueryBehaviour getReply
INFO: Got reply: (INFORM-REF
:sender ( agent-identifier :name mobileAgent0@192.168.0.103:8888/JADE :addresses (sequence http://DESKTOP-5OQQLK0:7778/acc ))
:receiver (set ( agent-identifier :name storekeeperAgent0@192.168.0.103:8888/JADE :addresses (sequence http://DESKTOP-5OQQLK0:7778/acc ))
:X-JADE-Encoding Base64
:content "r00ABXNyABFjdXN0b211ci5Gb29kUGxhbnITpkl5CmJBAgABTAAEcGxhbnQAD0xqYXZl3V0aWwvTWFW03hwc3IAEWphdmEudXRpbC5IYXN0TWFWBQfawcMMWYNEDAAJGAAp
:reply-with storekeeperAgent0@192.168.0.103:8888/JADE1548033356976 :in-reply-to query_ref1548033356976 :language Java_Serialization :con
sty 21, 2019 2:15:57 AM agents.storekeeper.behaviours.SendFridgeStateRequest onGettingReply
INFO: Get fridge state FridgeState[state=[])
sty 21, 2019 2:15:57 AM agents.storekeeper.behaviours.MyQueryBehaviour getReply
INFO: Got reply: (INFORM-REF
:sender ( agent-identifier :name fridgeAgent0@192.168.0.103:8888/JADE :addresses (sequence http://DESKTOP-5OQQLK0:7778/acc ))
:receiver (set ( agent-identifier :name storekeeperAgent0@192.168.0.103:8888/JADE :addresses (sequence http://DESKTOP-5OQQLK0:7778/acc ))
:X-JADE-Encoding Base64
:content "r00ABXNyAA5kyi5Gcm1k22VTdG9yZU2+OJIFMs2HAgABTAAIcHJvZHVjdHNOABBMamF2YS91dGlsL0xpc3Q7eHBzcGATamF2YS51dGlsLkFycmF5TGldHib0h22x2GdAw
:reply-with storekeeperAgent0@192.168.0.103:8888/JADE1548033357111 :in-reply-to query_ref1548033357096 :language Java_Serialization :con
```

4. MerchantAgent otrzymuje od StorekeeperAgent INFO EstimatedFridgeStatePlan. Ponieważ stan lodówki jest na początku pusty, to summaryczny plan wynosi 1.5 litra mleka i 3.5 bochenka chleba. MerchantAgent tworzy zamówienie zgodne z planem.

```
sty 21, 2019 2:17:00 AM agents.merchant.behaviours.GettingFoodPlans action
INFO: Waiting for plan
sty 21, 2019 2:17:01 AM agents.merchant.behaviours.GettingFoodPlans action
INFO: Get plan :[FoodPlan(plan={2019-01-20=[1.5 of Milk, 3.5 of Bread]})]
sty 21, 2019 2:17:01 AM agents.merchant.behaviours.GettingFoodPlans action
```

```
sty 21, 2019 2:17:32 AM agents.merchant.MerchantAgent makeOrder
INFO: Make order: [1.5 of Milk, 3.5 of Bread] :
```