AASD Projekt - część E - końcowa

Zespół X
Gabriel Rębacz
Robert Ostoja-Lniski
Dawid Brzozowski
Michał Belniak

30 stycznia 2022

1 Identyfikacja i opis problemu

Problem, który zostanie rozwiązany w ramach projektu na przedmiot AASD dotyczy społeczności wędkarzy. Z ogólnego punktu widzenia każdy z nich ma na celu uzyskanie jak najwyższego zadowolenia z połowów. Istotne dla nich są takie czynniki jak między innymi wielkość złowionej ryby, liczba złowionych ryb, poniesiony koszt finansowy związany z połowem czy poświęcony czas. Problemami, jakie mogą oni napotkać, są na przykład: nieznajomość akwenu, brak informacji o miejscu, gdzie można nabyć zezwolenie wędkarskie lub nadspodziewanie zatłoczone miejsca łowieckie.

Wyzwaniem jest także dobór asortymentu przez właścicieli sklepów wędkarskich w taki sposób, by odpowiadał potrzebom lokalnej społeczności. Wpływ na popyt na poszczególne produkty ma na przykład przekrój gatunkowy ryb w pobliskim jeziorze lub po prostu ich zatłoczenie. Jeśli w pewnym jeziorze będzie miało miejsce ponadprzeciętne występowanie gatunków drapieżnych, to prawdopodobnie bardziej opłacalna stanie się sprzedaż akcesoriów do spinningu niż spławików.

Dodatkowo połowy odbywają się na terenach prywatnych, należących do odpowiednich związków rybackich. Owe organy wydają zezwolenia wędkarskie oraz dokonują ich kontroli. Znając zatłoczenie na swoich akwenach mogą oni optymalizować ceny zezwoleń oraz zdecydować, do jakiego akwenu skierować kontrolę, aby mieć największe prawdopodobieństwo na naliczenie kar za brak zezwoleń. Nie muszą być to zawsze najbardziej zatłoczone łowiska. Przykładowo, jeśli w pewnym miejscu znajduje się dużo wędkarzy, ale prognozowane

są tam opady deszczu w najbliższym czasie, wysłanie kontroli nie przyniosłoby spodziewanych rezultatów, gdyż wkrótce wędkarze pojadą do domu.

Kolejnym istotnym aspektem jest zarządzanie akwenami. Właściciele prowadzą swoiste gospodarstwa wodne. Z ich punktu widzenia, problemem może być niepełna wiedza o stanie zarybienia ich akwenów, z uwzględnieniem podziału na gatunki. Wiedza ta jest potrzebna do utrzymywania równowagi ekosystemu jeziora, zatem do ich zadań należy przeprowadzanie różnych kontroli między innymi połowy statystyczne ryb, ustalanie topologii dna, badanie jakości wód i wielu innych parametrów. Na podstawie zebranych informacji podejmują oni decyzje dotyczące zarybiania łowisk różnymi gatunkami ryb, sprzątania, wycinania tataraku i trzcin aby stan akwenu był jak najlepszy przy zachowaniu wysokiej atrakcyjności dla wędkarzy. Gospodarstwa wodne dodatkowo uzyskują przychody z połowów ryb ze swoich jezior i rzek. Znając różnorodność gatunków w swoich jeziorach mogą podejmować bardziej świadome decyzje w jakim akwenie (lub nawet w jakiej jego części) dokonać połowu konkretnego gatunku.

Podsumowując, wyzwaniem jest optymalizacja zadowolenia wędkarzy, właścicieli jezior, sklepów oraz dbanie o ekosystem akwenów poprzez wymianę informacji na temat różnych parametrów łowisk, zapotrzebowania wędkarzy i podejmowanie odpowiednich decyzji z nimi związanymi.

2 Rozwiązanie

Rozwiązanie będzie polegało na utworzeniu systemu składającego się z sensorów oraz dronów, które zbierają cenne informacje o łowiskach oraz o aktualnej pogodzie. Za przekazywanie tych danych do zarządców akwenów, właścicieli sklepów oraz wędkarzy będą pośredniczyły interpretujące zebrane dane agenty, których zadaniem będzie również podejmowanie autonomicznych decyzji związanych z zarządzaniem akwenami, na przykład uruchamianie oczyszczalni wód, automatyczne wypuszczanie ryb z hodowli zapasowych oraz wspomaganie decyzji. Pomiary środowiska wykonywane będą z ustalonym okresem.

2.1 Elementy Systemu

System będzie składał się z trzech głównych komponentów.

- Sensorów oraz dronów zbierających informacje
- Agentów podejmujących akcji na podstawie owych informacji

- Agentów agregujących zebrane dane
- Docelowych odbiorców zebranych danych

2.1.1 Sensory oraz drony

Sensory oraz drony mają na celu umożliwić automatyczne zbieranie danych o danym jeziorze. Najbardziej wartościowe informacje dla wszystkich potencjalnych klientów systemu to warunki pogodowe, zatłoczenie łowisk na danych jeziorze, występujące w jeziorze gatunki ryb oraz poziom zarybienia.

Warunki pogodowe będą badane przy wykorzystaniu stacji badawczych rozmieszczonych w okolicach danego akwenu (lub kilku akwenów). Sensory tworzące te stacje badawcze odpowiedzialne będą za odczyt między innymi ciśnienia, temperatury, prędkości wiatru oraz wilgotności powietrza. Warunki takie jak wilgotność lub samym szansa na wystąpienie opadów w oczywisty sposób mogą wpłynąć na zatłoczenie wędkarzy w obrębie pewnego jeziora w danym dniu. Z drugiej strony istnieje związek między ciśnieniem oraz temperaturą, a zachowaniem ryb w jeziorze.

Kolejny zestaw sensorów będzie rozmieszczony w wodach akwenu. Ich podstawowymi zadaniami będą badanie parametrów wody (temperatury oraz czystości) oraz rozpoznawanie gatunków i wielkości ryb.

Dodatkowo w skład systemu wejdą drony wyposażone w kamery i wyznaczające zatłoczenie wędkarzy. Drony będą mogły przeprowadzać okresowe kontrole nad kilkoma jeziorami, dzięki czemu ich tor zostanie ustalony. Manualne sterowanie stanie się w ten sposób zbędne.

2.1.2 Agenty

Celem agentów będzie agregacja informacji dotyczących danego aspektu związanego z obserwowanym jeziorem i reagowanie na określone zdarzenia. Przykładem jest decyzja o wyborze najlepszego łowiska dla zainteresowanego wędkarza, który niekoniecznie zna akwen, na którym zamierza wędkować. Agent, na podstawie zatłoczenia łowisk oraz liczby ryb w ich okolicy dobierze najlepsze miejsce do połowów. Kolejnym przykładem działania podjęcie działania w przypadku przekroczenia w danym jeziorze dolnego limitu liczby ryb pewnego gatunku. W podobnym przypadku, agent rozpocznie odpowiednią procedurę, jeśli obserwowany przez niego czujnik stężenia szkodliwych substancji w wodzie przekroczy dozwoloną wartość. Stacje pogodowe mogą być natomiast wykorzystane przez agenty w celu ostrzeżenia wszystkich okolicznych wędkarzy o zbliżającym się groźnym zjawisku, takim jak silny wiatr lub opady gradu.

2.1.3 Odbiorcy danych

Agenty będą udostępniały dane w formie zrozumiałej dla użytkowników końcowych systemu. Zagregowane dane zostaną dodane do bazy danych lub pliku przechowującego raportowane podejmowane akcje. Dane w takiej formie będą mogły być pobrane za pomocą odpowiednich interfejsów i zostać zaprezentowane w zarówno aplikacji mobilnej, jak i na stronie internetowej. Taka dowolność pozwala sprostać oczekiwaniom potencjalnych użytkowników końcowych. Zakładamy, że wersja mobilna będzie dedykowana dla wędkarzy, natomiast gospodarstwa rybackie i właściciele sklepów zainteresują się w większym stopniu stroną internetową.

3 Proponowana Architektura Systemu

Wstępnie planujemy system z podziałem agentów na rodzaje odpowiedzialnych za zadania o danym charakterze.

3.1 Agent kontrolujący warunki atmosferyczne

Agent ten będzie odpowiedzialny za zbieranie informacji dotyczących warunków atmosferycznych oraz w razie groźnych zjawisk będzie on rozsyłał alerty pogodowe do wędkarzy oraz uruchamiał syreny alarmowe.

3.2 Agent kontrolujący parametry wody

Będzie on zbierał parametry wody takie jak poziom zanieczyszczeń, temperatura czy zawartość biomasy. Akcje jakie będzie on podejmował to uruchamianie oczyszczalni wód oraz rozsyłanie decyzji o zamknięciu łowisk w razie przekroczenia wartości krytycznych parametrów wody. Będzie on również uwzględniał dane pobierane z agenta kontrolującego warunki atmosferyczne.

3.3 Agent kontrolujący charakterystykę zarybienia

Do jego zadań będzie należało zbieranie danych dotyczących statystyk zarybienia poszczególnymi gatunkami ryb oraz wykonywanie akcji mających na celu uzyskania balansu gatunkowego akwenów. Ponadto będzie on uwzględniał parametry wody w celu dobrania odpowiednich gatunków.

3.4 Agent kontrolujący zatłoczenie łowisk

Będzie on zbierał informacje dotyczące zatłoczenia łowisk, które będą wykorzystywane w dalszych procesach.

3.5 Agent akumulujący dane do interfejsów użytkownika

Zadaniem tego agenta będzie pobieranie danych ze wszystkich pozostałych agentów w celu ich przetworzenia i prezentacji w interfejsach użytkownika, za pomocą których będzie możliwa poglądowa kontrola stanu akwenów i dodatkowe wspieranie procesu podejmowania decyzji.

3.6 Agent określający rekomendację łowiska

Agent będzie określał, czy jest on w stanie aktualnie zarekomendować łowisko do połowów na podstawie informacji o warunkach atmosferycznych, zatłoczeniu i poziomie zarybienia łowisk danych zebranych ze wszystkich agentów, które będzie pobierał od agenta akumulującego.

Ponadto wszystkie agenty w razie potrzeb będą mogły wymieniać się ze sobą informacjami z zebranych czujników.

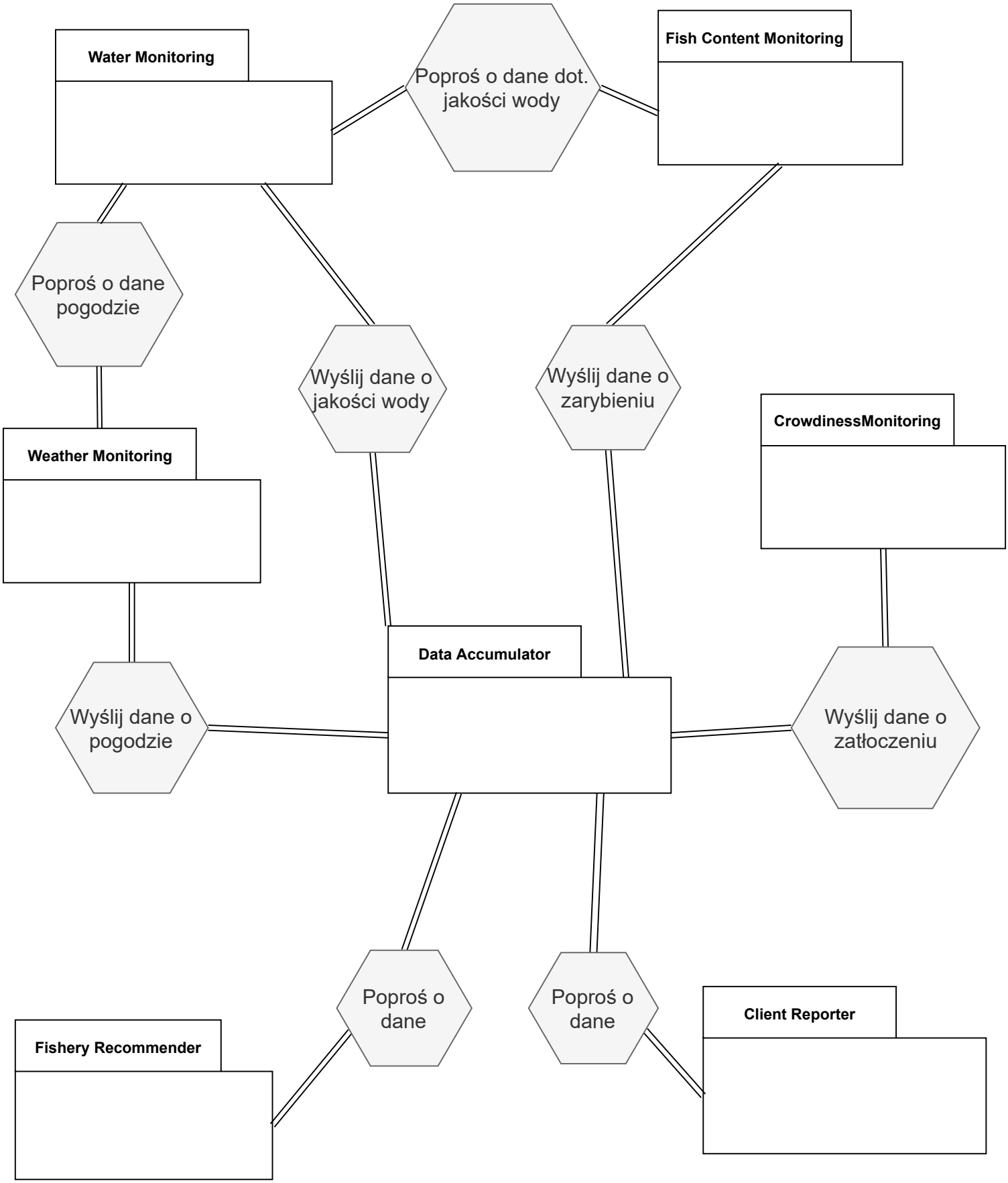
3.7 Interfejsy

W systemie zostaną utworzone emulacje interfejsów służących do pobierania danych z sensorów oraz dronów. Będą one specjalizacjami pewnego abstrakcyjnego interfejsu bazowego z podstawowymi metodami do odczytu danych. Agent będzie posiadał referencję do interfejsu bazowego, a dana implementacja określić źródło danych.

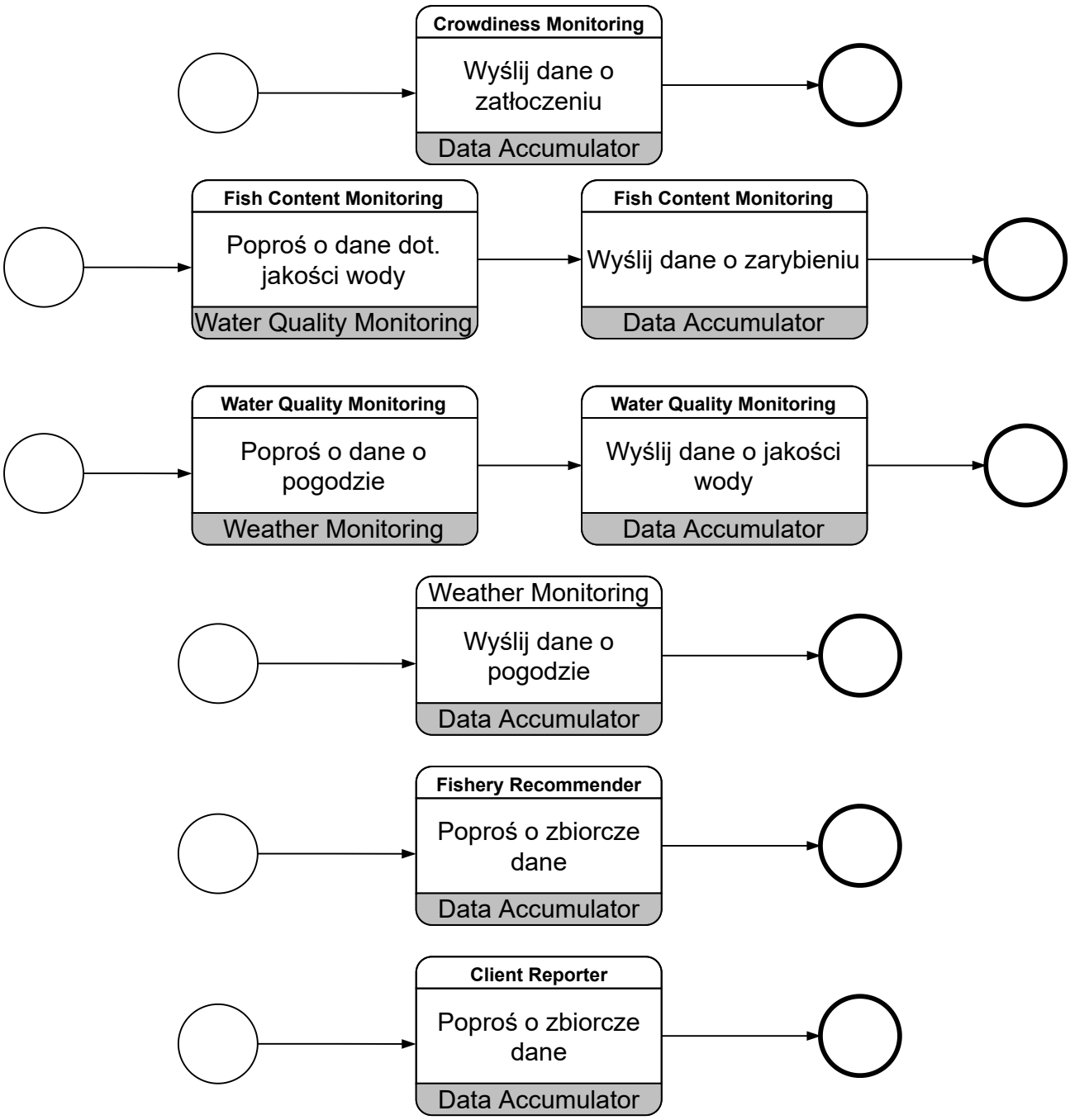
4 Projekt systemu w notacji BPMN

System został zaprojektowany z wykorzystaniem notacji BPMN [1, 2], architektura została przedstawiona poniżej.

Diagram konwersacji

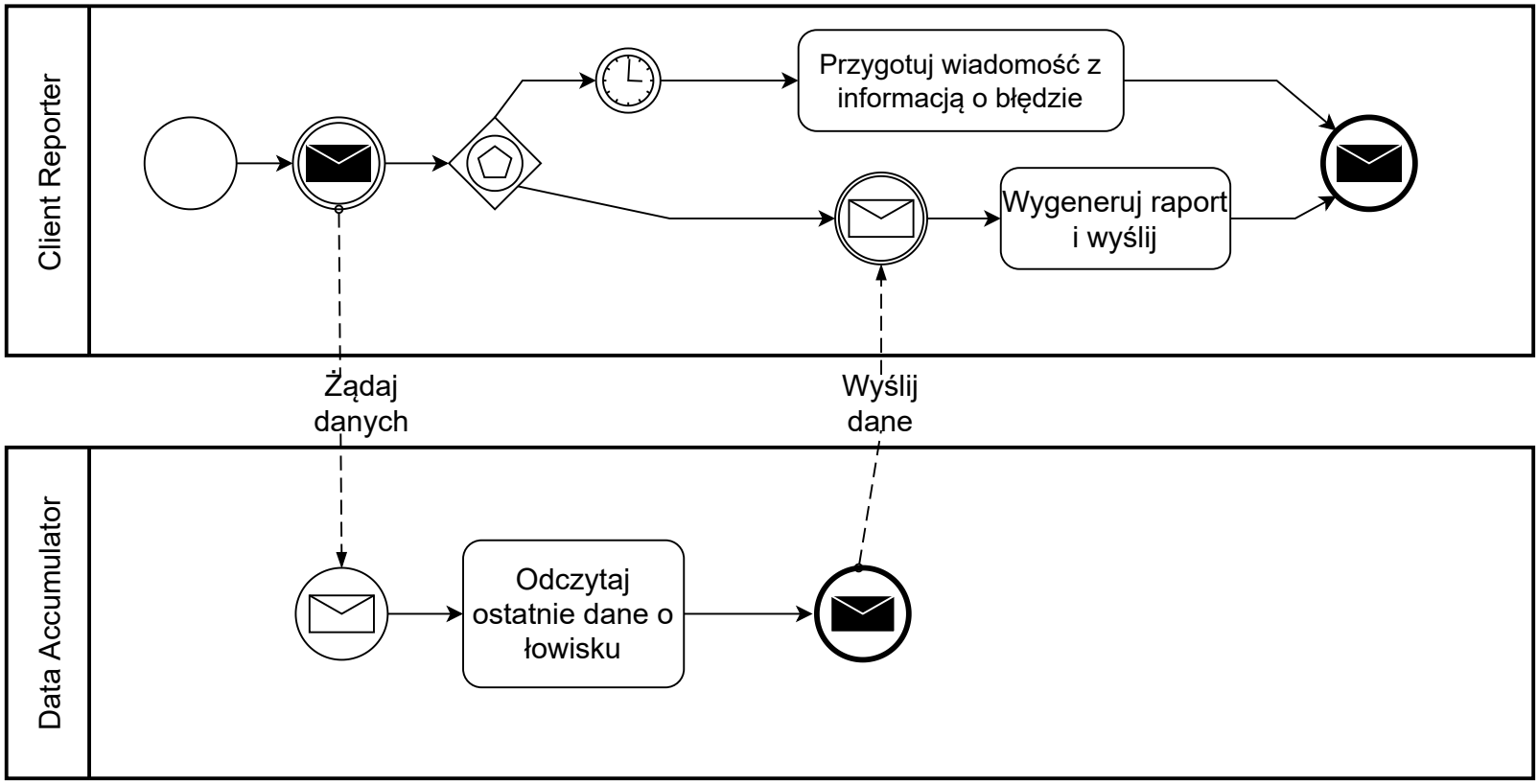


Diagramy choreografii

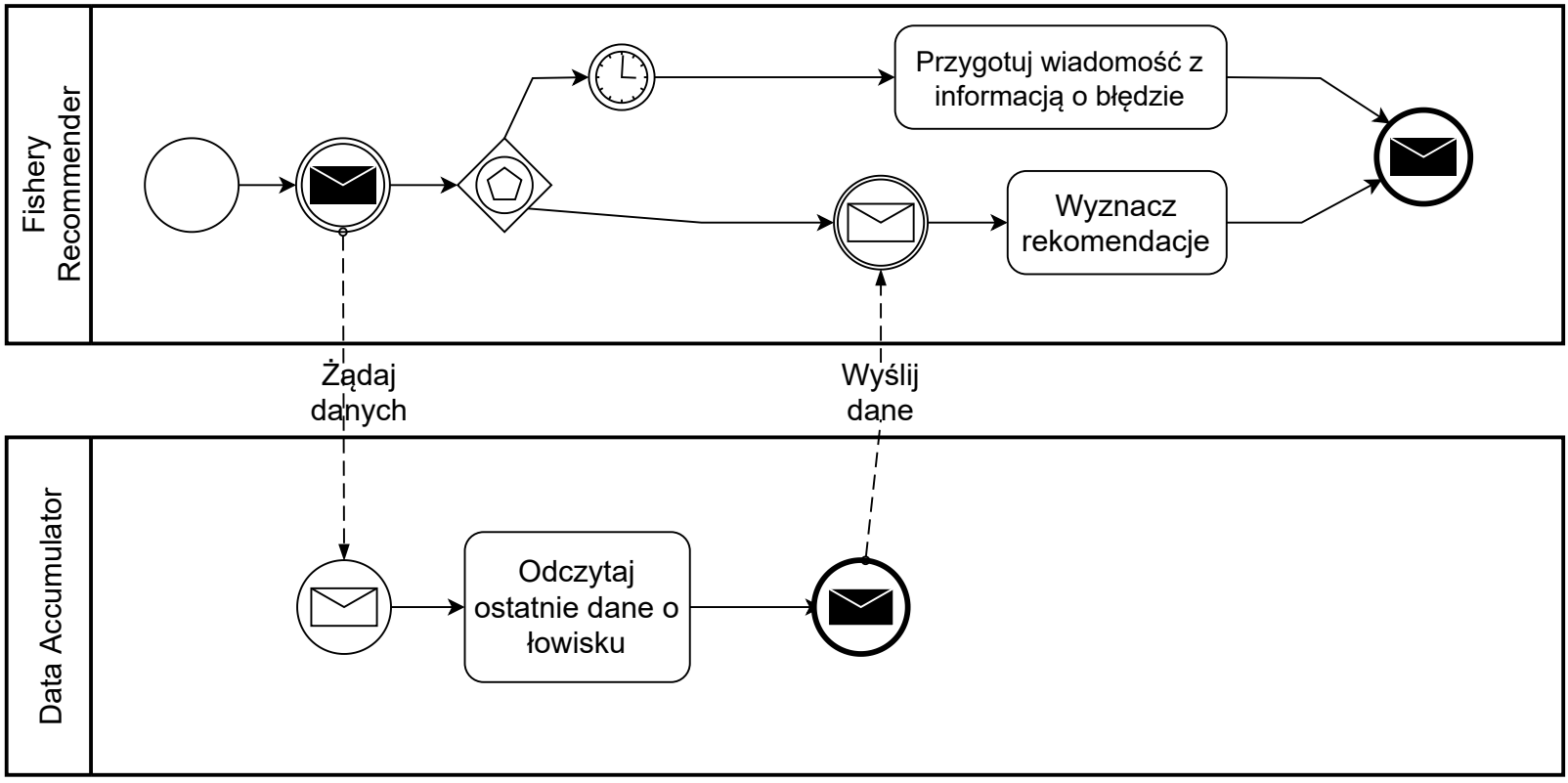


Diagramy kolaboracji

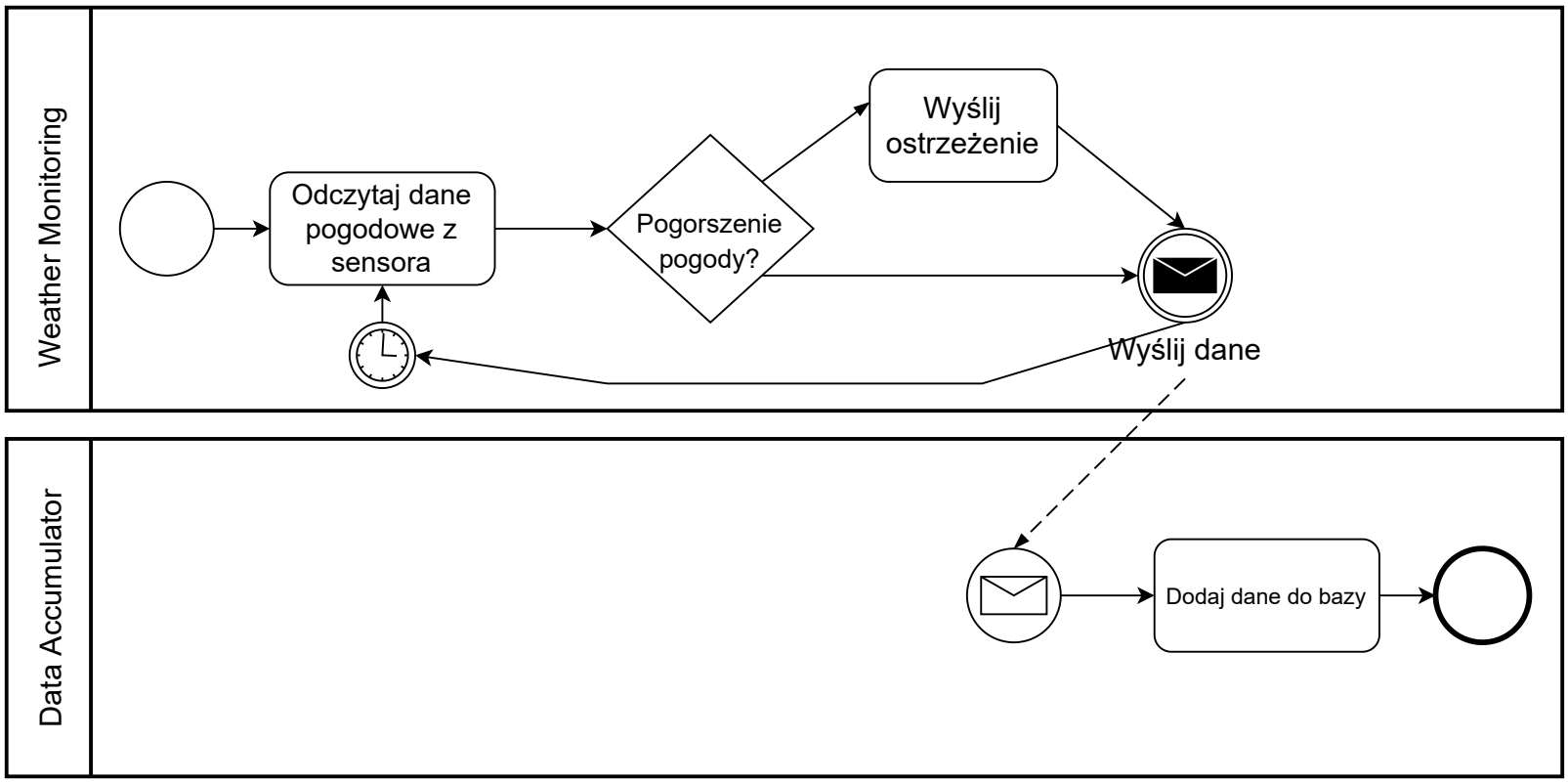
Generowanie raportów



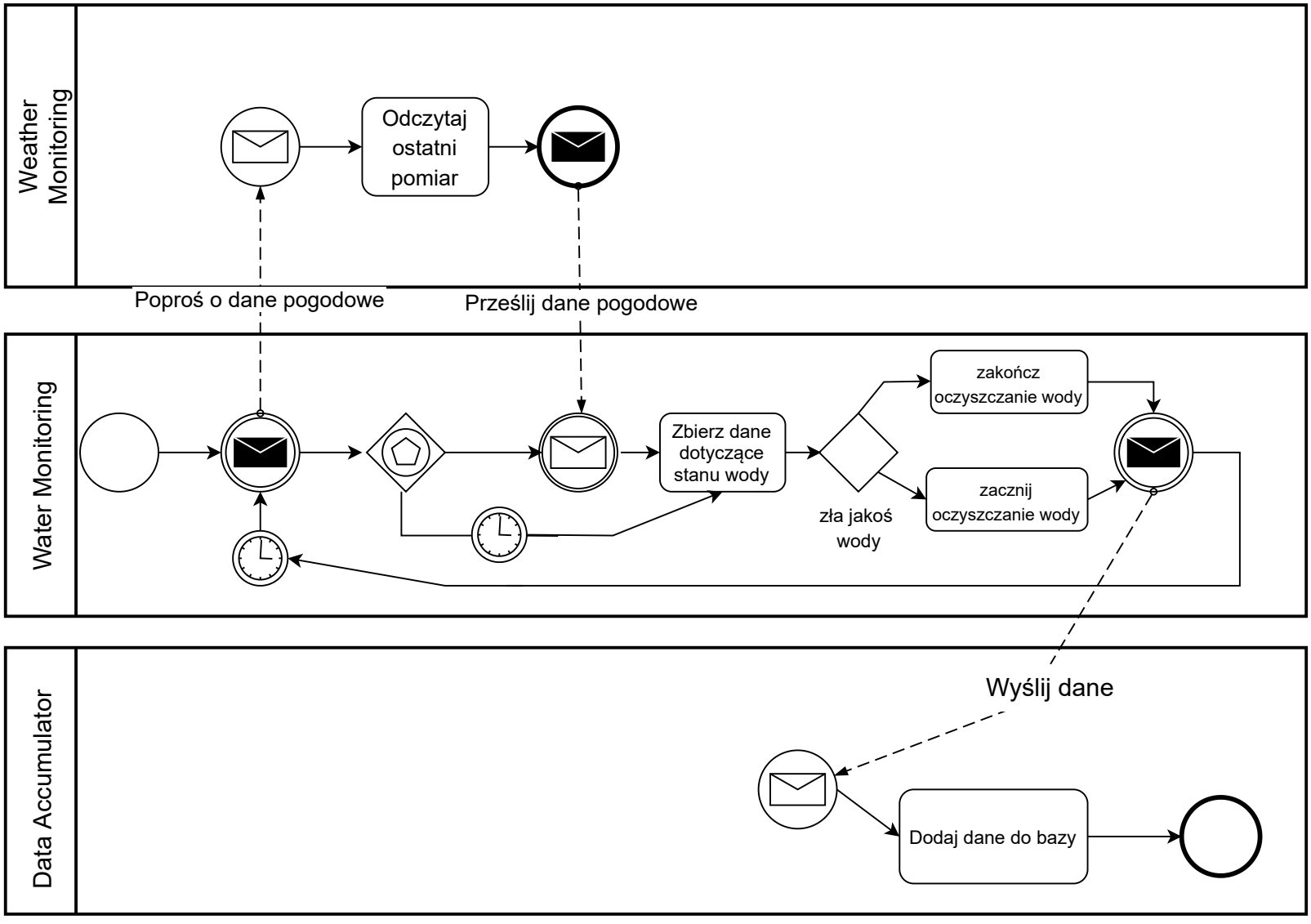
Określanie rekomendacji



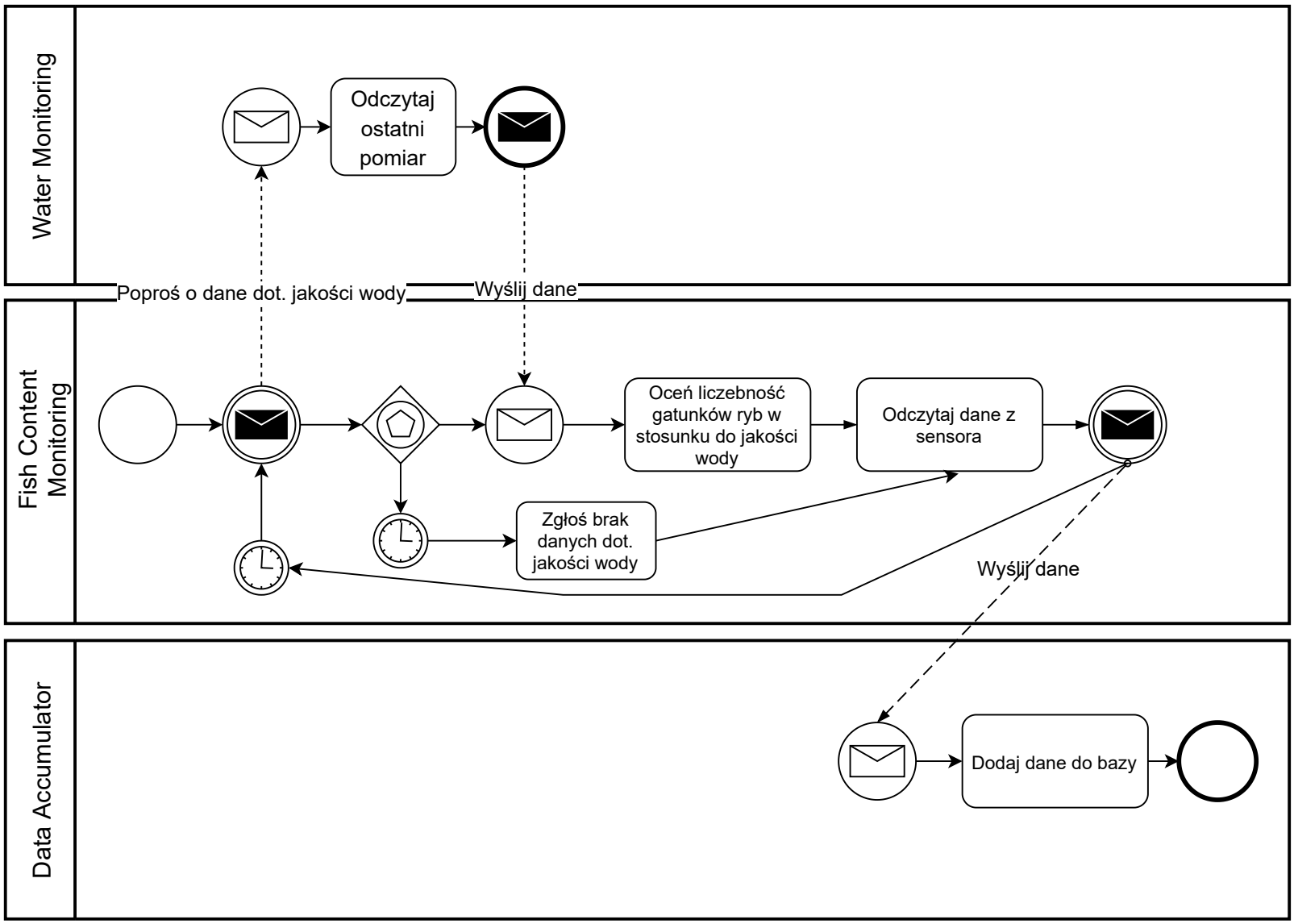
Akumulowanie danych pogodowych



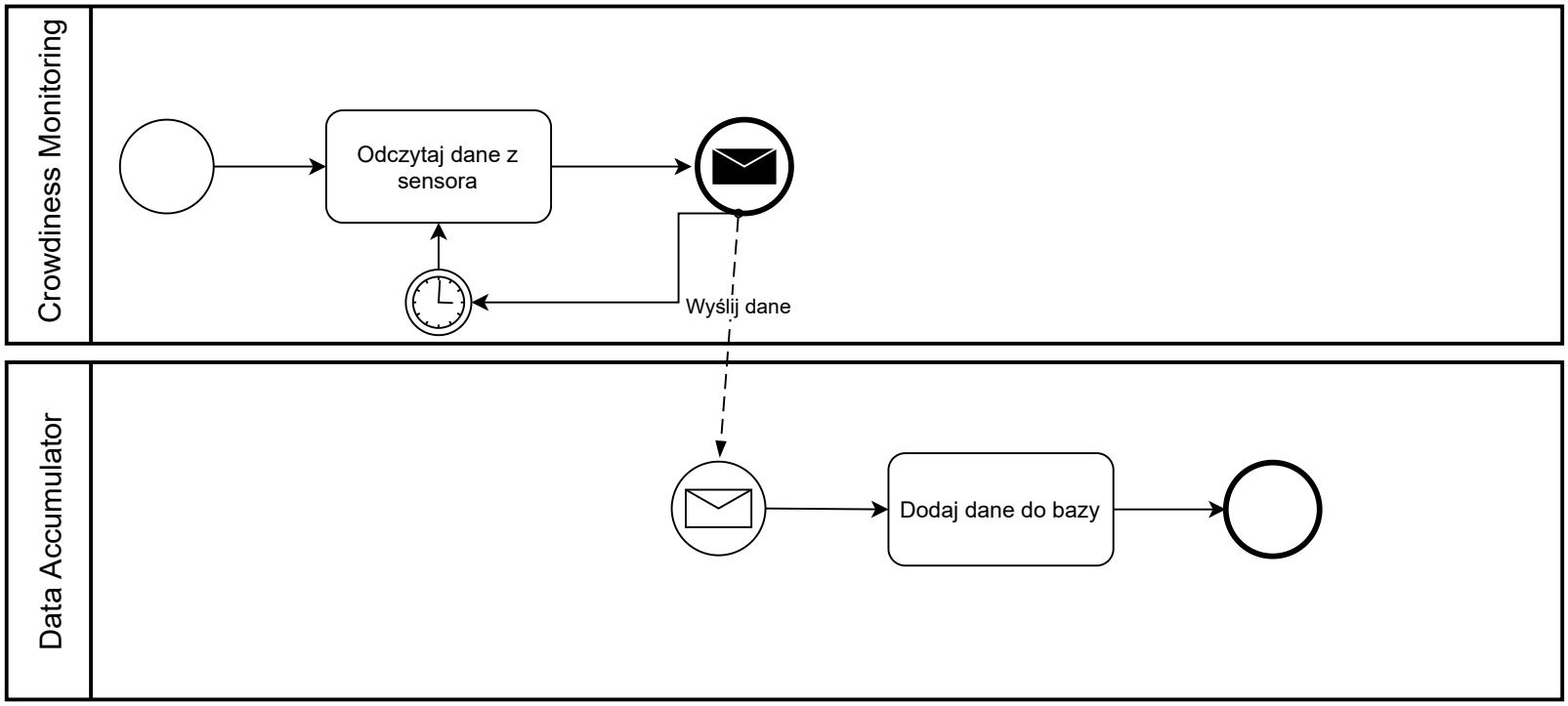
Akumulowanie danych o warunkach wody



Akumulowanie danych o zarybieniu



Akumulowanie danych o zatłoczeniu



5 Opis implementacji systemu

System został zaimplementowany w języku *Python* oraz środowisku *SPADE* (*Smart Python Agent Development Environment*) [4].

5.1 Ogólny opis środowiska

SPADE jest biblioteką języka Python, która pozwala na tworzenie systemu wieloagentowego wspierającego komunikację zgodną z *FIPA* przy użyciu serwera *XMPP* (*Extensible Messaging and Presence Protocol* [5]). *SPADE* pozwala na:

- Odczyt stanu agenta w czasie rzeczywistym
- Asynchroniczne przetwarzanie zdarzeń (w oparciu o moduł *asyncio*)
- Dostęp do webowego interfejsu graficznego
- Komunikację poprzez dowolny serwer implementujący standard *XMPP*
- Wewnętrzną implementację kolejki wiadomości

Serwer *XMPP* pełni funkcję platformy wieloagentowej, na której każdy agent ma unikalny identyfikator *JID*. Służy on do rejestracji i zalogowania agenta. Aby była możliwość stworzenia strumienia komunikacyjnego między serwerem a agentem niezbędne jest poprawne zalogowanie agenta. Uwierzytelnienie jest obsługiwane wewnętrznie w bibliotece *SPADE* poprzez implementację protokołu *XMPP*.

5.1.1 Reprezentacja Agenta

Agent w bibliotece *SPADE* jest reprezentowany jako klasa `spade.agent.Agent`. Udostępnia ona interfejs służący między innymi do jego konfiguracji, uruchomienia, zatrzymania działania, czyszczenia zasobów po jego usunięciu oraz modyfikacji jego zachowania. W naszym rozwiązaniu każdy agent jest instancją klasy dziedziczącej.

5.1.2 Zachowania Agenta

Zgodnie z dokumentacją zachowanie (*behaviour*) jest rozumiane jako zadanie wykonywane przez agenta w pewien zdefiniowany, powtarzalny sposób. Biblioteka udostępnia trzy kategorie zachowań:

- Cykliczne i Okresowe (dla powtarzających się zadań)
- Jednorazowe, z limitem czasu wykonania (dla zwykłych zadań)
- Reprezentowane jako automat skończony

Do implementacji zachowania służy dostarczony w bibliotece moduł `spade.behaviour`, w których zdefiniowane są interfejsy do każdego typu zdarzenia.

5.2 Implementacja Agentów

Typy agentów zostały zaimplementowane poprzez klasy:

- ClientReporter
- CrowdMonitoring
- DataAccumulator
- FishContentMonitoring
- FisheryRecommender
- User
- WaterMonitoring
- WeatherMonitoring

Są one specjalizacjami klasy `BaseAgent`, która implementuje wspólne dla wszystkich agentów operacje (takie jak tworzenie identyfikatora *JID*, przypisanie łowiska do agenta czy zapisywanie subskrybentów - innych agentów, które są zainteresowane otrzymywaniem wiadomości od danego agenta). `BaseAgent` dziedziczy z kolei po klasie `Agent` (z modułu `spade.agent`).

Inicjalizacja agentów odbywa się w skrypcie `main.py`. Dla każdego numeru łowiska (użytkownik ma możliwość podania liczby łowisk jako parametr wejściowy, maksymalnie 10) tworzony jest zestaw agentów monitorujących, po jednym z każdego typu, oraz agenty *User*, *ClientReporter* i *FisheryRecommender*. Po inicjalizacji, wszystkie agenty są uruchamiane razem z usługą serwującą natywny interfejs webowy na unikalnym porcie dla każdego agenta. Numery portów to kolejne liczby od 10001 wzwyż. Adres interfejsu webowego to `http://localhost:<port>/spade`.

5.3 Komunikacja

Mechanizm komunikacji agentów jest w pełni wspierany przez *SPADE*. Połączenia między nadawcami oraz adresatami są jawnie zdefiniowane w skrypcie *main.py* poprzez wywołania metody `subscribe_to()`.

5.3.1 Wysyłanie wiadomości

Komunikaty są przekazywane za pomocą obiektów typu `Message` z modułu `spade.message`. Typ komunikatu określony jest w polu `metadata`, gdzie podawana jest nazwa ontologii, performatywa, typ wiadomości oraz jej format, a zawartość komunikatu w polu `body`. Jako format zawartości przyjęliśmy format JSON. Każda wiadomość ma określonego adresata, który przyjmie wiadomość, jeśli jej typ jest taki sam jak oczekiwany przez przynajmniej jedno z jego określonych zachowań (wykorzystano mechanizm szablonów ze środowiska SPADE [3]). Obsługa wiadomości jest częścią zachowania agenta.

5.3.2 Odebranie wiadomości

Jeśli odebranie wiadomości jest częścią zachowania agenta, agent czeka na nią określony czas. W praktyce oznacza to okresowe sprawdzania kolejki wiadomości. Jeśli odczyt się powiedzie, to obsługa wiadomości jest wykonywana bezpośrednio przed właściwą akcją agenta. Jeśli agent nie otrzyma wiadomości w ustalonym czasie, wykonuje tylko część zdefiniowanych zadań (obsługa przypadku, gdy brak jest wiadomości) lub od razu ponawia oczekiwanie.

5.3.3 Format wiadomości

Wysyłane wiadomości posiadają pola:

- `to` - adresat wiadomości (JID)
- `body` - zawartość wiadomości w formacie JSON (zależna od agenta),
- `metadata` - metadane wiadomości w formacie JSON:
  - `ontology` - ontologia,
  - `performative` - performatywa (Inform lub Request),
  - `language` - format wiadomości,
  - `type` - typ wiadomości (np. 'Fish content data')

5.3.4 Przykłady komunikatów

Wiadomość o stanie jakości wody na łowisku *convivial avocet*.

```
{
    "to": "fish_content_0@localhost",
    "body": {
        "fishery": "convivial avocet",
        "data": {
            "temperature": 30,
            "oxygen_level": 0.95,
            "contamination_level": 0.23
        }
    }.
    "metadata": {
        "ontology": "fishery-system",
        "language": "JSON",
        "performative": "Inform",
        "type": "Water quality"
    }
}
```

Wiadomość o zawartości ryb na łowisku *bouncy worm*.

```
{
    "to": "data_accumulator@localhost",
    "body": {
        "fishery": "bouncy worm",
        "data": {
            "fish_content": {
                "Salmon": 124,
                "Carp": 154,
                "Tuna": 43
            },
            "fish_content_rating": "Average"
       }
    },
    "metadata": {
        "ontology": "fishery-system",
        "language": "JSON",
        "performative": "Inform",
```

```
        "type": "Fish content"
    }
}
```

5.3.5 Wykorzystany serwer XMPP

Komunikacja odbywa się dzięki serwerowi *Prosody* zgodnym z *XMPP*, który jest uruchomiony na maszynie lokalnej. Aby w większym stopniu wydzielić zasoby dla tego serwera oraz odpowiednio wirtualizować porty wykorzystywany jest obraz *Prosody* dla kontenera *Docker*. W momencie tworzenia kontenera ma miejsce mapowanie plików konfiguracyjnych z maszyny lokalnej na odpowiadające zasoby po stronie kontenera. Rejestracja agentów odbywa się za pomocą osobnego skryptu wywołującego polecenia *prosodyctl register <args>* i przekazującego dane do uwierzytelnienia agenta (*JID* oraz hasło). Plik konfiguracyjny serwera Prosody *prosody.cfg.lua* znajduje się w folderze *prosody* projektu.

5.4 Skrócony opis standardowego przebiegu programu

- Konfiguracja agentów wraz z zależnościami pomiędzy nimi.
- Rozpoczęcie monitorowania danych z jezior za pomocą agentów odpowiedzialnych za monitorowanie zatłoczenia, zarybienia, jakości wody oraz pogody. Proces ten, powtarzany jest co 2 sekundy dla każdego z agentów. Po każdym odczycie dane wysyłane są do agenta `DataAccumulator`,
- Jeśli agent monitorujący jakość wody zauważy znaczne pogorszenie jej jakości, uruchomi on procedurę oczyszczania wody,
- Dane gromadzone są przez agenta `DataAccumulator`, który odpowiada za odbieranie informacji z agentów monitorujących obecny stan na łowiskach,
- Użytkownik prosi o wygenerowanie rekomendacji za pośrednictwem agenta `FisheryRecommender`, który generuje w danej chwili rekomendację dotyczącą najlepszego łowiska. Jest ona następnie zapisywana w odpowiednim katalogu na dysku,
- Cyklicznie tworzone są raporty z łowisk przez agenta `ClientReporter`, a następnie są one zapisywane w odpowiednim katalogu na dysku,
- Proces ten jest powtarzany aż do wyłączenia programu przez użytkownika.

5.5 Algorytm generowania rekomendacji

Zaimplementowany algorytm jest jedynie formą przykładu - może być on w rzeczywistości daleki od optymalnego.

- Dla każdego łowiska wyznaczana jest ocena od 0 do 4. Dla każdej z kategorii: pogoda, zarybienie, zatłoczenie, jakość wody, algorytm przyznaje ocenę od 0 do 1.
- Ocena za pogodę to średnia ważona z temperatury powietrza (waga 30%), natężenia opadów (30%), prędkości wiatru (waga 20%), ciśnienia (10%) oraz zachmurzenia (waga 10%).
- Ocena za zatłoczenie to wynik normalizacji *min-max* po liczbie osób na każdym z łowisk.
- Ocena zarybienia jest już wyznaczona przez agenta monitorującego zarybienie, gdyż to on teoretycznie najlepiej zna specyfikę danego łowiska (np. wielkość i głębokość akwenu).
- Ocena jakości wody to średnia ocen za poziom zanieczyszczenia wody, temperaturę wody oraz poziom tlenu.

Za optymalną temperaturę powietrza przyjęliśmy 20°C, optymalne ciśnienie 1000hPa, optymalną temperaturę wody 15°C, a dla pozostałych parametrów - 0.

Poniżej znajduje się przykładowa rekomendacja. Została ona wygenerowana dla dwóch łowisk. Wybrane łowisko posiada wysoki wynik (3.7 / 4).

```
Best fishery: illustrious trogon
Overall score: 3.7 out of max 4
Crowd at the fishery: 9 persons
Fish content: {'Salmon': 66, 'Carp': 154, 'Tuna': 207}
Fish content rating: VERY_HIGH
Water contamination level: 30.8%
Water oxygen level: 100.0%
Water temperature: 12.0°C
Air temperature: 16.9°C
Precipitation: 24.8mm/h
Air pressure: 1035.6hPa
Wind speed: 20.4km/h
Clouds: 51%
```

5.6 Tworzenie raportu

Raport jest generowany na podstawie zebranych danych przez *DataAccumulator*. Aby został stworzony wymagane jest otrzymanie wszystkich niezbędnych danych (nie ma możliwości wygenerowania raportu zawierającego przykładowo tylko dane o pogodzie). Dzięki temu system zapewnia, że każdy raport ma pełne informacje o łowiskach. Proces dostarczania raportu składa się z następujących kroków:

- po upłynięciu określonego czasu *Client Reporter* wysyła prośbę do *Data Accumulator* o zestaw aktualnych danych
- Dane otrzymane w odpowiedzi są odpowiednio formatowane do generowania raportu (jest zmieniana ich struktura, ma miejsce mapowanie etykiet). Format wyjściowy to *JSON*.
- Raport jest zapisywany w dedykowanym katalogu.

Przykładowy raport dla dwóch łowisk wygląda następująco:


```
[
{
    "name": "mindful carp",
    "weather": {
        "cloudiness": 86.84,
        "precipitation_rate": 90.41,
        "pressure": 1042.42,
        "temperature": -11.95,
        "wind_speed": 4.70
    },
    "water_quality": {
        "contamination_level": 0.31,
        "oxygen_level": 0.56,
        "temperature": 14
    },
    "fish": {
        "fish_content": {
            "Salmon": 174,
            "Carp": 15,
            "Tuna": 65
        },
        "fish_content_rating": 0.6
```

```
        },
        "crowd": 16
    },
    {
        "name": "busy bettong",
        "weather": {
            "cloudiness": 0.0,
            "precipitation_rate": 98.76,
            "pressure": 1036.10,
            "temperature": 4.812,
            "wind_speed": 13.74
        },
        "water_quality": {
            "contamination_level": 0.39,
            "oxygen_level": 1,
            "temperature": 35
        },
        "fish": {
            "fish_content": {
                "Salmon": 7,
                "Carp": 81,
                "Tuna": 135
            },
            "fish_content_rating": 0
        },
        "crowd": 0
    }
]
```


5.7 Logowanie

W projekcie korzystamy z biblioteki logging do przekierowywania informacji na standardowe wyjście oraz do plików. Każdy *Agent* ma swoją własną instacją klasy *logger*, która zapisuje wiadomości do konkretnego pliku. Dodatkowo, istnieje zbiorczy plik z logami od poszczególnych aktorów. Możliwe jest również włączenie logowania na standardowe wyjście flagą *–verbose*. Nie przyjmuje ona żadnych argumentów. Poniżej znajduje się przykładowy fragment loga agenta *CrowdMonitoring*

```
2022-01-25 23:00:11,603 - crowd_monitoring_1 (CrowdMonitoring) - INFO
```

```
- initialization
2022-01-25 23:00:12,996 - crowd_monitoring_1 (CrowdMonitoring) - INFO
- is running
2022-01-25 23:00:12,997 - crowd_monitoring_1 (CrowdMonitoring) - INFO
- sent crowd data: {"fishery": "watchful chamois", "data": "39"}
2022-01-25 23:00:14,998 - crowd_monitoring_1 (CrowdMonitoring) - INFO
- sent crowd data: {"fishery": "watchful chamois", "data": "36"}
2022-01-25 23:00:17,001 - crowd_monitoring_1 (CrowdMonitoring) - INFO
- sent crowd data: {"fishery": "watchful chamois", "data": "39"}
2022-01-25 23:00:18,918 - crowd_monitoring_1 (CrowdMonitoring) - INFO
- sent crowd data: {"fishery": "watchful chamois", "data": "41"}
```

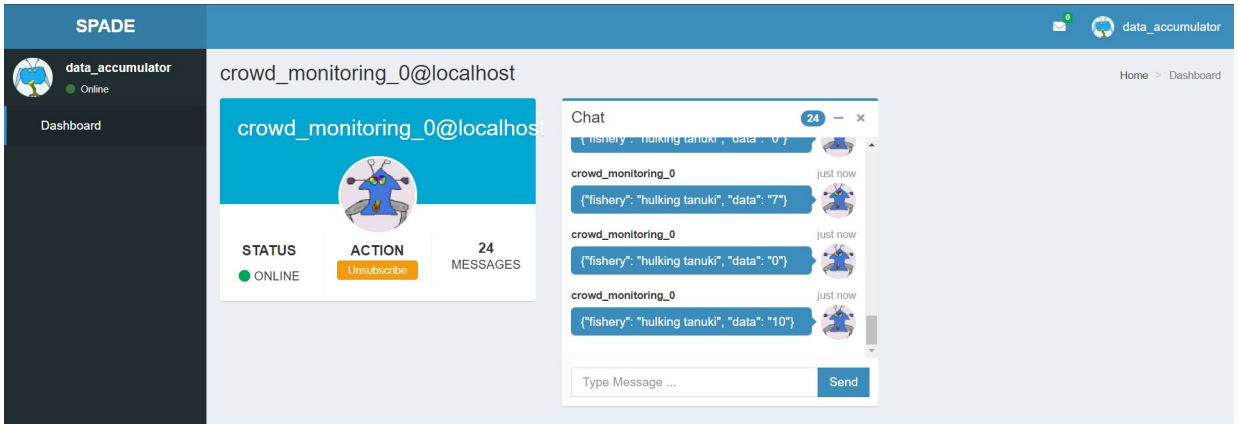
Zbiorczy log prezentuje się następująco:

```
2022-01-25 21:46:46,916 - data_accumulator (DataAccumulator) - INFO
- received data: 28 from crowd_monitoring_0@localhost for fishery:
nickel hoatzin.
2022-01-25 21:46:48,543 - weather_monitoring_0 (WeatherMonitoring)
- INFO - sent weather data: {"type": "Weather", "fishery": "nickel
hoatzin", "data": "{\"temperature\": -8, \"pressure\": 1045.0603946242868,
\"precipitation_rate\": 11.6564154692241, \"wind_speed\": 81.2016833293017,
\"cloudiness\": 23.481353436856104}"}
2022-01-25 21:46:48,546 - data_accumulator (DataAccumulator) - INFO
- received data: {"temperature": -8, "pressure": 1045.0603946242868,
"precipitation_rate": 11.6564154692241, "wind_speed": 81.2016833293017,
"cloudiness": 23.481353436856104} from weather_monitoring_0@localhost
for fishery: nickel hoatzin.
2022-01-25 21:46:48,559 - water_monitoring_0 (WaterMonitoring) - INFO
- sent water quality data: {"type": "Water quality", "fishery": "nickel
hoatzin", "data": "{\"temperature\": 14, \"oxygen_level\": 1}"}
2022-01-25 21:46:48,562 - data_accumulator (DataAccumulator) - INFO
- received data: {"temperature": 14, "oxygen_level": 1, "contamination_level":
1} from water_monitoring_0@localhost for fishery: nickel hoatzin.
2022-01-25 21:46:48,924 - crowd_monitoring_0 (CrowdMonitoring) - INFO
- sent crowd data: {"type": "Crowd", "fishery": "nickel hoatzin",
"data": "30"}
2022-01-25 21:46:48,926 - data_accumulator (DataAccumulator) - INFO
- received data: 30 from crowd_monitoring_0@localhost for fishery:
nickel hoatzin.
```

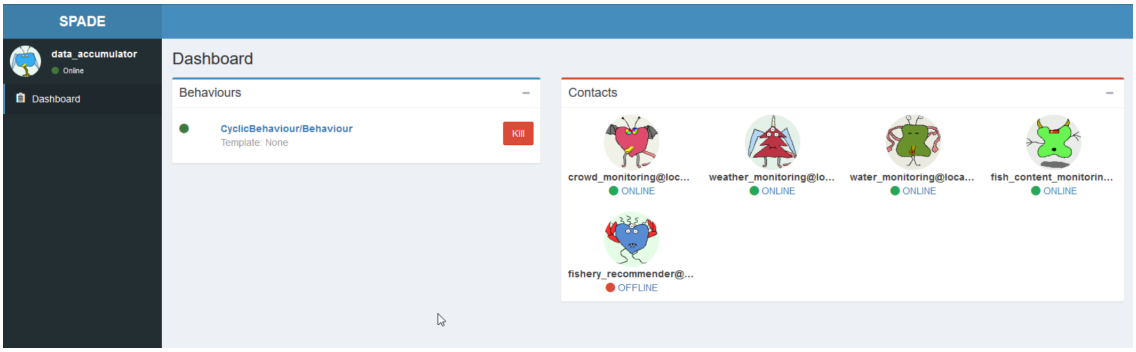
5.8 Interfejs webowy

Każdy agent serwuje aplikację webową na określonym porcie korzystając z modułu *web* (który jest częścią *SPADE*). Proces jest uruchamiany wywołaniem *agent.web.start(hostname="127.0.0.1", port=port)*, gdzie *agent* jest instacją pewnej specjalizacji klasy `spade.agent`. Skrypt *main.py* uruchamia aplikację webową dla każdego utworzonego agenta.

5.8.1 Przykłady interfejsu



Rysunek 1: Komunikacja między agentami.



Rysunek 2: Tablica kontaktów agenta *data_accumulator*.

5.9 Wykonane testy

Oprócz manualnej weryfikacji działania programu, stworzyliśmy także zestaw testów integracyjnych sprawdzających komunikację między agentami. Spraw-

dzają one, czy agenty wymieniają się komunikatami w wyznaczonym limicie czasu i czy format wiadomości jest poprawny. Sprawdzane jest także czy uruchamiana jest procedura oczyszczania wody w razie potrzeby oraz czy rekomendacje są generowane na żądanie użytkownika. Testy znajdują się w katalogu *test*, a uruchomić je można komendą `python -m unittest discover -s test -t .` uruchomioną w katalogu głównym projektu.

5.10 Napotkane problemy

Z uwagi na niejasną dokumentację serwera *Prosody* konfiguracja protokołu *XMPP* była niespodziewanie czasochłonna. Dodanie przykładowych konfiguracji do dokumentacji mogłoby rozwiązać ten problem.

SPADE jest biblioteką, która została w pełni przepisana na język *Python* w wersji 3.6 w 2017 roku. Z powodu stosunkowo krótkiego okresu dostępności niektóre funkcje działają niestabilnie lub nie są wystarczająco dobrze opisane.

Napotkaliśmy problemy z działaniem mechanizmu subskrypcji. Jedną z możliwych form subskrypcji w serwerze XMPP jest forma 'both', który mówi o obustronnej chęci odbierania wiadomości dla dwóch agentów. Niestety, dokumentacja, mimo że wspomina o takiej formie, nie podaje w jaki sposób ją uzyskać, a próby stworzenia takiej subskrypcji w naszym projekcie były nieudane. Dlatego, w naszym projekcie każdy agent jawnie przechowuje listę subskrybentów oraz agentów których subskrybuje, aby tę wiedzę wykorzystać w trakcie wysyłania wiadomości.

Kłopotliwe było także tworzenie testów, gdyż API agentów zapewnione przez bibliotekę SPADE jest ograniczone. Nie ma na przykład możliwości eleganckiego "wpięciaśię w moment zakończenia wykonania kroku cyklicznego zachowania i potrzebne jest opakowywanie metod za to odpowiadających.

Dodatkowo podczas implementacji natrafiliśmy na problem z biblioteką *keyboard* na systemie operacyjnym *MacOs*. W momencie próby jej wykorzystania rzucany był wyjątek *OSError: Error 13 - Must be run as administrator*. Proponowane na forach internetowych rozwiązania skupiały się na uruchamianiu programu jako użytkownik poleceniem *sudo*, co naszym zdaniem łamało *Zasadę Najmniejszych Uprawnień*. Ostatecznie zmieniliśmy tę bibliotekę.

5.11 Brakujące elementy w implementacji

Nie ma także możliwości kontrolowania przez użytkownika danych "generowanych"przez sensory. Są one generowane losowo, lecz pozwala to na faktyczne odzwierciedlenie zmiennych warunków panujących w środowisku. Nie można również symulować większej ilości łowisk niż 10. Wynika to z faktu,

że w narzędziu SPADE istnieje błąd, który nie pozwala na dynamiczną rejestrację nowych użytkowników na serwerze XMPP (Prosody). Użytkownicy odpowiadający poszczególnym agentom są zdefiniowani ręcznie. Zostało zarejestrowanych po dziesięciu użytkowników na każdy rodzaj agenta poza Data Accumulator, Fishery Recommender, Client Reporter oraz User (w tym przypadku każdy agent ma jednego użytkownika).

6 Link do repozytorium kodu

Implementacja systemu opisanego powyżej z wykorzystaniem narzędzia SPADE [4] znajduje się pod podanym url: `https://github.com/robertostoja-lniski/AASD_ZespolX`

Bibliografia

[1] Object Management Group. *Dokumentacja Business Process Model and Notation*. [Strona internetowa; data odwiedzin: 27.11.2021]. URL: `https://www.bpmn.org/`.

[2] Oracle. *Dokumentacja Business Process Model and Notation*. [Strona internetowa; data odwiedzin: 27.11.2021]. URL: `https://docs.oracle.com/cd/E28271_01/doc.1111/e15176/timers_bpmpd.htm`.

[3] SPADE Contributors. *Agent communications*. [Strona internetowa; data odwiedzin: 06.01.2022]. URL: `https://spade-mas.readthedocs.io/en/latest/agents.html`.

[4] Spade contributors. *Dokumentacja SPADE*. [Strona internetowa; data odwiedzin: 27.11.2021]. URL: `https://spade-mas.readthedocs.io/`.

[5] XMPP Contributors. *Extensible Messaging and Presence Protocol*. [Strona internetowa; data odwiedzin: 06.01.2022]. URL: `https://xmpp.org/about/technology-overview/`.