AASD 2020Z

Zespół BMTU

Etap 3: Opis implementacji systemu

Skład zespołu:
Bartosz Leszczyński
Michał Wiśniewski
Tomasz Macutkiewicz
Uladzislau Lahoikin

Repozytorium: https://github.com/wisniowy/BMTU_AASD_Project
Wutwaw-sharepoint: https://cutt.ly/tjCz1kD

Identyfikacja problemu

Problem, który zamierzamy rozwiązać jest dość powolna reakcja służb miejskich na występujące codziennie utrudnienia związane z infrastrukturą oraz środowiskiem miejskim, spowodowana brakiem zasobów ludzkich oraz wdrożonych i rozwiniętych inteligentnych systemów informatycznych, wspierających służby miejskie w przetwarzaniu danych, a następnie w konserwacji tej infrastruktury lub też przyrody miejskiej. Problemy objawiają się na wiele sposobów, między innymi:

- Powstawanie ubytków lub nierówności w drogach publicznych.
- Śmieci rozsypane po chodnikach lub parkach.

- Zaniedbanie trawników, krzaki utrudniające poruszanie się po chodnikach.
- Zła synchronizacja sygnalizacji świetlnej.

Koncepcja rozwiązania

W celu rozwiązania wyżej wymienionych problemów zostanie zaimplementowany aktorowy system decyzyjny wspierający służby miejskie. Inspirowaliśmy się rozwiązaniami typu CRM (Customer Relationship Management System), które działają w postaci aplikacji komputerowej oraz aplikacji mobilnej do zgłaszania powstałych problemów. System ten będzie oparty na architekturze klient-serwer. Zadaniem klienta będzie stworzenie i przesłanie do systemu zgłoszenia, które będzie zawierało takie dane, jak:
- lokalizacja wykrytego problemu,
- numer identyfikacyjny zgłaszającego,
- ewentualny krótki opis problemu.

W przyszłości można by było rozbudować zgłoszenia o możliwość dodawania i przekazywania bardziej zaawansowanych informacji:
- zdjęcie problemu/zdarzenia,
- czas utworzenia zgłoszenia/usunięcia problemu,
- lokalizacja wykrytego problemu w postaci współrzędnych geograficznych,
- nadawanie unikatowych identyfikatorów zgłoszeniom (automatycznie nadawany przez system).

System będzie implementowany w taki sposób, żeby móc wspierać aplikację mobilną oraz komputerową. Aplikacja mobilna ma na celu umożliwienie przeciętnym mieszkańcom zgłaszania problemów, które napotkają na drodze. Aplikacja komputerowa będzie przeznaczona dla służb miasta zarządzających monitoringiem miejskim. Bedzie ona w czasie rzeczywistym przetwarzała zgłoszenia i zawarte w nich informacje. Dodatkowo mógłby być stworzony system do przetwarzania obrazów wysyłanych z kamer, który będzie interpretować je oraz wysyłać określone wcześniej dane do serwera (przetwarzanie obrazu raczej po stronie serwera).

Zadaniem serwera jest z kolei klasyfikacja odebranych danych, nadawanie im priorytetów oraz powiadomienie odpowiednich służb miejskich o występującym problemie. Klient będzie mógł śledzić status swojego zgłoszenia (odrzucone, w trakcie realizacji, naprawione).

Wybór technologii

Po przeanalizowaniu istniejących technologii zdecydowaliśmy się na wykonanie projektu w języku Java z wykorzystaniem frameworku Akka. Akka to zestaw narzędzi i środowisko wykonawcze do tworzenia współbieżnych, rozproszonych i odpornych na błędy aplikacji sterowanych zdarzeniami na JVM. Więcej informacji na temat tego frameworku można znaleźć pod adresem: https://akka.io/try-akka/. Wybraliśmy ten framework ponieważ umożliwia implementację systemów wieloaktorowych w języku Java, która jest w miarę znana wszystkim członkom zespołu.

Sposoby implementacji aktorów

W ramach etapu 3 podzieliliśmy nasz system aktorowy na klasy reprezentujące role i komunikaty. Sugerując się diagramem ról i obowiązków z poprzedniego etapu projektu, zdecydowaliśmy się na zaimplementowanie podstawowych klas aktorów dla następujących ról:
- Zgłaszający
- Koordynator
- Konserwator

Podstawą naszego systemu aktorowego jest klasa bazowa Actor, z rozszerzeniem pomocniczym AbstractActor. Każdy aktor występujący w systemie został zaimplementowany za pomocą oddzielnej klasy Javy, dziedziczącej cechy po klasie akka.actor.AbstractActor.

1. Reporter
    - Aktor zgłaszającego
    - Komunikuje się z aktorami klasy Coordinator
    - Tworzy nowe zgłoszenia i wysyła je do koordynatorów
    - Odbiera komunikaty o wykonanej pracę od koordynatorów
2. Coordinator
    - Aktor koordynatora
    - Odpowiada za komunikację z aktorami klasy Reporter i Conservator
    - Odbiera zgłoszenia od aktorów zgłaszających problem
    - Odpowiada za logikę obliczeniową
    - Zarządza odebranymi zgłoszeniami i nadaje im ID
    - Sprawdza dostępność (Busy/Free) istniejących na mapie konserwatorów i ich bieżącą lokalizację
    - Na podstawie listy lokalizacji wyznacza najbliższego do problemu konserwatora i prosi go o usunięcie problemu
    - Odbiera informacji od konserwatora o wykonanej akcji usunięcia problemu i informuje o tym właściwego zgłaszającego
3. Conservator
    - Aktor konserwatora
    - Odpowiada za komunikację z aktorem klasy Coordinator
    - Udostępnia swoją bieżącą lokalizację
    - Odpowiada za logikę obliczeniową, odbiera zgłoszenia od koordynatora
    - Informuje koordynatora o swoim statusie (On the way/Resolving fault/Free)
    - Przemieszcza się po mapie w stronę powstałego problemu
    - Po dotarciu do punktu docelowego rozpoczyna prace naprawcze i informuje o tym koordynatora
    - Po wykonaniu pracy i naprawieniu problemu wysyła informację zwrotną do koordynatora

Dodatkowo mamy klasę BmtuSystemMain, która uruchamia cały system aktorowy. W tej klasie przy pomocy metody system.actorOf() tworzymy aktora klasy SystemSupervisor, który z kolei odpowiada za utrzymanie systemu i tworzenie nowych aktorów. Każdy aktor jest tworzony i uruchamiany w osobnym wątku, od momentu uruchamiania staje się aktywny i wysyła wiadomości do innych aktorów. W tym etapie jeszcze nie została zaimplementowana funkcjonalność priorytetyzacji zgłoszeń i negocjacji między aktorami.

Komunikacja między aktorami i protokoły komunikacyjne

W tym kroku zostały zdefiniowane wszystkie rodzaje komunikatów, niezbędne do poprawnego działania systemu aktorowego. Aktorzy komunikują się między sobą za pomocą komunikatów asynchronicznych. Dzięki temu nadawca nie czeka na przetworzenie wiadomości przez odbiorcę i może wykonywać inne prace. Skrzynka pocztowa aktora to w zasadzie kolejka wiadomości z semantyką porządkowania. Kolejność wielu wiadomości wysłanych przez tego samego Aktora jest zachowywana, ale można ją przeplatać z wiadomościami wysłanymi przez innego Aktora.

Komunikacja między aktorami opiera się na protokołach komunikacyjnych typu FIPA Request When Interaction Protocol, które realizują konwersacje aktorów, z których jeden prosi/żąda drugiego aktora o wykonanie pewnej akcji. Gdy aktor zaakceptuje żądanie, po wykonaniu akcji powinien poinformować aktora który sformułował prośbę o wyniku jej wykonania.

Wszystkie istniejące protokoły komunikacyjne między aktorami zostały zdefiniowane w odrębnych klasach Javy:

- SendReportMessage
- UpdateLocationMessage
- Fault
- RequestAssistanceMessage
- FaultResolvedMessage
- UpdateReport

Przykładowy zrzut ekranu z działania

Po uruchamianiu systemu aktorowego przebieg działania na bieżąco jest wypisywany do konsoli zintegrowanego środowiska programistycznego. Wyniki działania systemu aktorowego są umieszczone na obrazku poniżej.

```
akka.actor.default-dispatcher-4] [akka://bmtu-system/user/system-supervisor] System-supervisor started
akka.actor.default-dispatcher-5] [akka://bmtu-system/user/system-supervisor/area0-reporter0] Reporter actor 0 started
akka.actor.default-dispatcher-6] [akka://bmtu-system/user/area0-coordinator] Coordinator actor started
akka.actor.default-dispatcher-6] [akka://bmtu-system/user/area0-coordinator] Coordinator received msg: SendReportMessage{reporterId=0, location=Coordinate{x=0, y=0}}
akka.actor.default-dispatcher-7] [akka://bmtu-system/user/system-supervisor/area0-conservator0] Conservator actor 0 started
akka.actor.default-dispatcher-5] [akka://bmtu-system/user/system-supervisor/area0-conservator2] Conservator actor 2 started
akka.actor.default-dispatcher-6] [akka://bmtu-system/user/system-supervisor/area0-conservator1] Conservator actor 1 started
akka.actor.default-dispatcher-5] [akka://bmtu-system/user/area0-coordinator] Coordinator received msg: UpdateLocationMessage{location=Coordinate{x=1, y=8}}
akka.actor.default-dispatcher-5] [akka://bmtu-system/user/area0-coordinator] Coordinator received msg: UpdateLocationMessage{location=Coordinate{x=1, y=1}}
akka.actor.default-dispatcher-5] [akka://bmtu-system/user/area0-coordinator] Coordinator received msg: UpdateLocationMessage{location=Coordinate{x=1, y=4}}
akka.actor.default-dispatcher-6] [akka://bmtu-system/user/area0-coordinator] Coordinator received msg: UpdateLocationMessage{location=Coordinate{x=1, y=8}}
akka.actor.default-dispatcher-6] [akka://bmtu-system/user/area0-coordinator] Coordinator received msg: UpdateLocationMessage{location=Coordinate{x=1, y=4}}
akka.actor.default-dispatcher-6] [akka://bmtu-system/user/area0-coordinator] Coordinator received msg: UpdateLocationMessage{location=Coordinate{x=1, y=1}}
akka.actor.default-dispatcher-6] [akka://bmtu-system/user/system-supervisor/area0-conservator0] Conservator received msg: RequestAssistanceMessage{fault=Fault{id=1, faultLocation=Coordinate{x=0, y=0}}}
akka.actor.default-dispatcher-6] [akka://bmtu-system/user/system-supervisor/area0-conservator0] On the way to Fault{id=1, faultLocation=Coordinate{x=0, y=0}}
akka.actor.default-dispatcher-8] [akka://bmtu-system/user/area0-coordinator] Coordinator received msg: UpdateLocationMessage{location=Coordinate{x=1, y=8}}
akka.actor.default-dispatcher-8] [akka://bmtu-system/user/area0-coordinator] Coordinator received msg: UpdateLocationMessage{location=Coordinate{x=1, y=4}}
akka.actor.default-dispatcher-6] [akka://bmtu-system/user/system-supervisor/area0-conservator0] [Coordinate{x=0, y=0}, Coordinate{x=1, y=0}, Coordinate{x=1, y=1}]
akka.actor.default-dispatcher-10] [akka://bmtu-system/user/area0-coordinator] Coordinator received msg: UpdateLocationMessage{location=Coordinate{x=1, y=8}}
akka.actor.default-dispatcher-10] [akka://bmtu-system/user/area0-coordinator] Coordinator received msg: UpdateLocationMessage{location=Coordinate{x=1, y=4}}
akka.actor.default-dispatcher-7] [akka://bmtu-system/user/area0-coordinator] Coordinator received msg: UpdateLocationMessage{location=Coordinate{x=1, y=8}}
akka.actor.default-dispatcher-11] [akka://bmtu-system/user/system-supervisor/area0-conservator0] Started working on Fault{id=1, faultLocation=Coordinate{x=0, y=0}}
akka.actor.default-dispatcher-7] [akka://bmtu-system/user/area0-coordinator] Coordinator received msg: UpdateLocationMessage{location=Coordinate{x=1, y=4}}
akka.actor.default-dispatcher-5] [akka://bmtu-system/user/area0-coordinator] Coordinator received msg: UpdateLocationMessage{location=Coordinate{x=1, y=4}}
akka.actor.default-dispatcher-5] [akka://bmtu-system/user/area0-coordinator] Coordinator received msg: UpdateLocationMessage{location=Coordinate{x=1, y=8}}
akka.actor.default-dispatcher-5] [akka://bmtu-system/user/area0-coordinator] Coordinator received msg: FaultResolvedMessage{faultId=1}
akka.actor.default-dispatcher-9] [akka://bmtu-system/user/system-supervisor/area0-reporter0] Reporter received UpdateReport msg 0
akka.actor.default-dispatcher-9] [akka://bmtu-system/user/system-supervisor/area0-reporter0] Reporter actor 0 stopped
akka.actor.default-dispatcher-11] [akka://bmtu-system/user/area0-coordinator] Coordinator received msg: UpdateLocationMessage{location=Coordinate{x=1, y=8}}
akka.actor.default-dispatcher-11] [akka://bmtu-system/user/area0-coordinator] Coordinator received msg: UpdateLocationMessage{location=Coordinate{x=1, y=4}}
akka.actor.default-dispatcher-9] [akka://bmtu-system/user/area0-coordinator] Coordinator received msg: UpdateLocationMessage{location=Coordinate{x=1, y=8}}
akka.actor.default-dispatcher-9] [akka://bmtu-system/user/area0-coordinator] Coordinator received msg: UpdateLocationMessage{location=Coordinate{x=1, y=4}}
```