Programowanie Agentowe
Industry 4.0 (project #2)
Raport III

4 VI 2020

Paweł Fornalik Szymon Tomulewicz Mateusz Zieliński

Spis treści

1 Opis projektu

Projekt będzie badał możliwość wykorzystania systemu wieloagentowego do modelowania złożonych procesów produkcyjnych, a w szczególności transportu surowców i dóbr pośrednich między jednostkami produkcyjnymi (groups of ultra modern machines - GoM) za pośrednictwem należących do nich autonomicznych jednostek transportowych (transport robot - TR) w inteligentnej, zautomatyzowanej fabryce.

Z założenia jednostki muszą ze sobą współpracować, aby móc realizować zamówienia swoich *GoMów*, co pociąga za sobą konieczność prowadzenia negocjacji w celu ustalenia porządku realizacji zleceń. Właśnie ta funkcjonalność ma zostać odwzorowana przy pomocy systemu agentowego, który umożliwi spójną współpracę wszystkich TR w obrębie zakładu, zapewnienie przepływu towarów według odpowiednich priorytetów, jak również obsługę rozmaitych sytuacji awaryjnych.

Projekt zostanie wykonany przy pomocy platformy Smart Python Agent Development Environment w wersji 3.1.4 (SPADE) w języku Python 3.8, w oparciu o protokół komunikacyjny XMPP. Platforma udostępnia biblioteki zawierające zarówno bazowe klasy agentów, jak i klasy wiadomości zgodnych ze standardem FIPA ACL oraz metody umożliwiające ich obsługę wewnątrz kodu.

2 Agenci

2.1 Factory

2.1.1 Role

- Factory

2.2 Management Agent

2.2.1 Role

- Manager

2.3 GoM - groups of ultra modern machines

2.3.1 Role

- Craftsman

2.4 TR - transport robot

2.4.1 Role

- Negotiator
- Transporter

3 Role

3.1 Factory

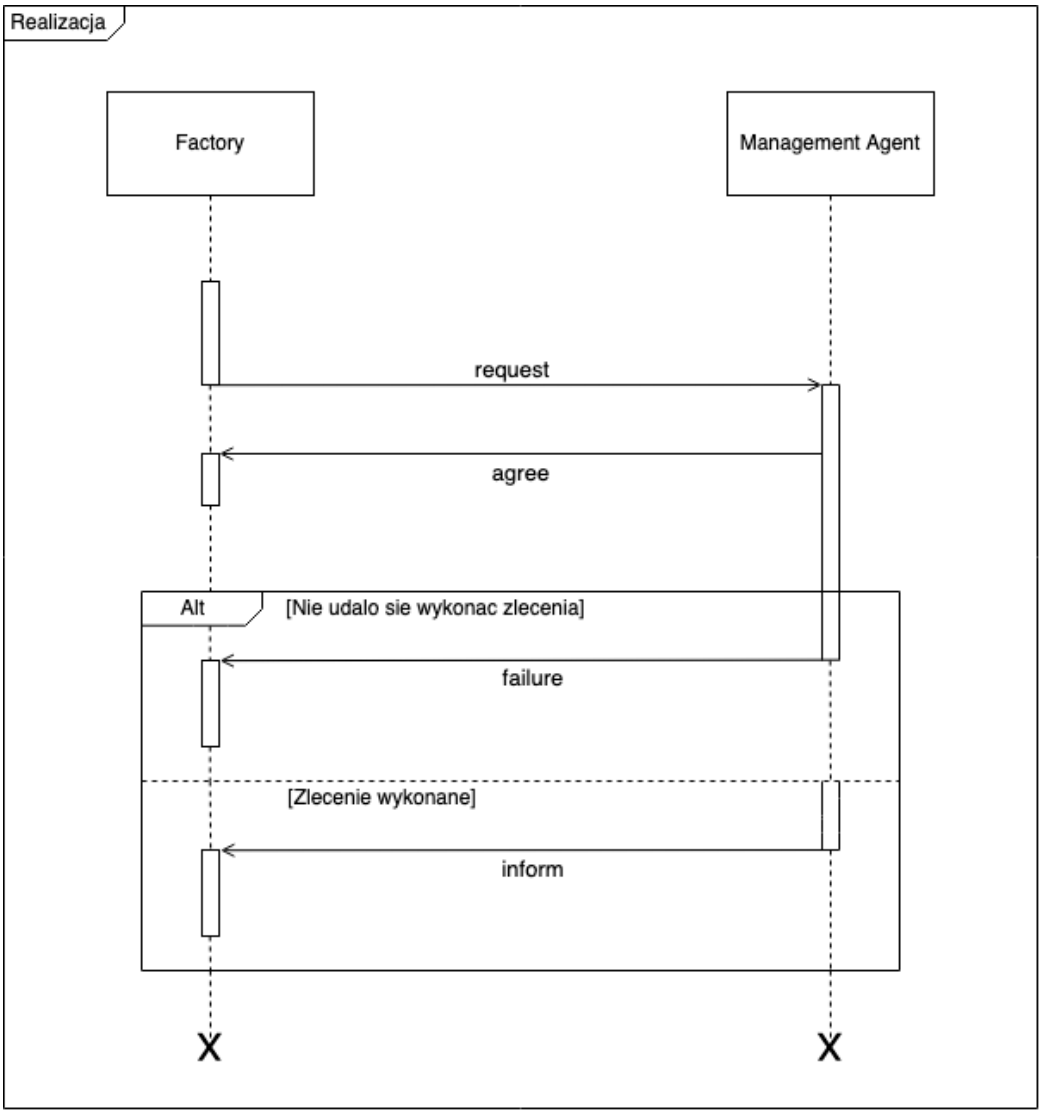
3.1.1 Opis

Posiada wiele *GoMów*, magazyn z materiałami i zatrudnia *Management Agenta*. Zajmuje się produkcją poprzez realizację zleceń, które są reprezentowane przez priorytet oraz ciąg operacji.

3.1.2 Funkcjonalność

- Generuje zamówienia o różnym priorytecie i różnym stopniu skomplikowania.
- Zleca wykonanie zamówienia *Management Agentowi* poprzez protokół *FIPA-request*, *Management Agent* nie może odmówić ani się zepsuć. (Rys. 1)

3.1.3 Komunikacja



Rysunek 1: Zlecenie realizacja zamówienia *Management Agentowi*

3.2 Manager

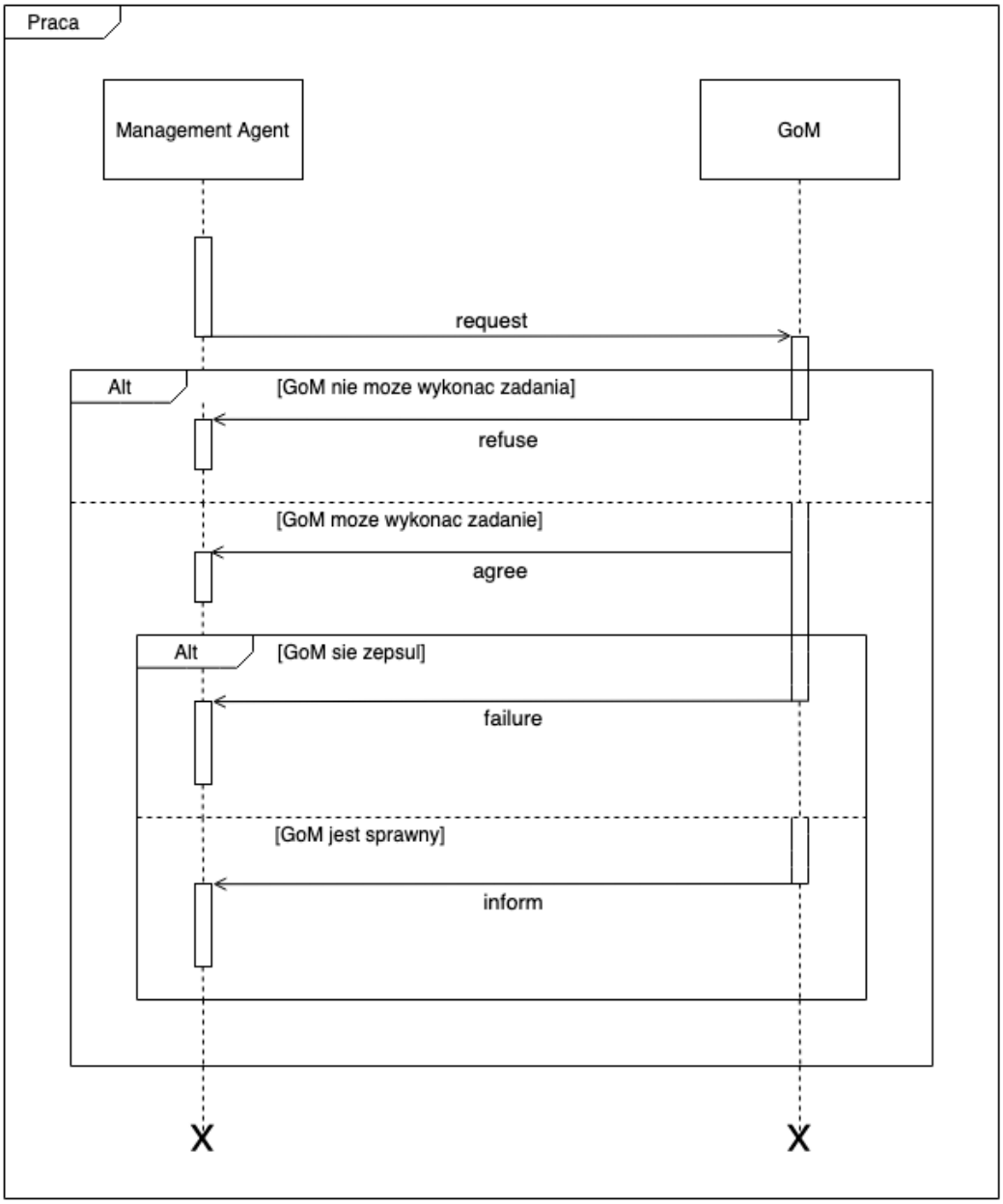
3.2.1 Opis

Realizuje zamówienia zlecając pracę *GoMom*. Jest głównym strategiem i zarządza całością. Posiada kolejkę priorytetową z zamówieniami oraz kolekcję *GoMów* wraz z ich stanami (wolny, zajęty, zepsuty *TR*) oraz informacjami o maszynach w gnieździe, czyli stan urządzenia (sprawny, zepsuty) i oferowana operacja.

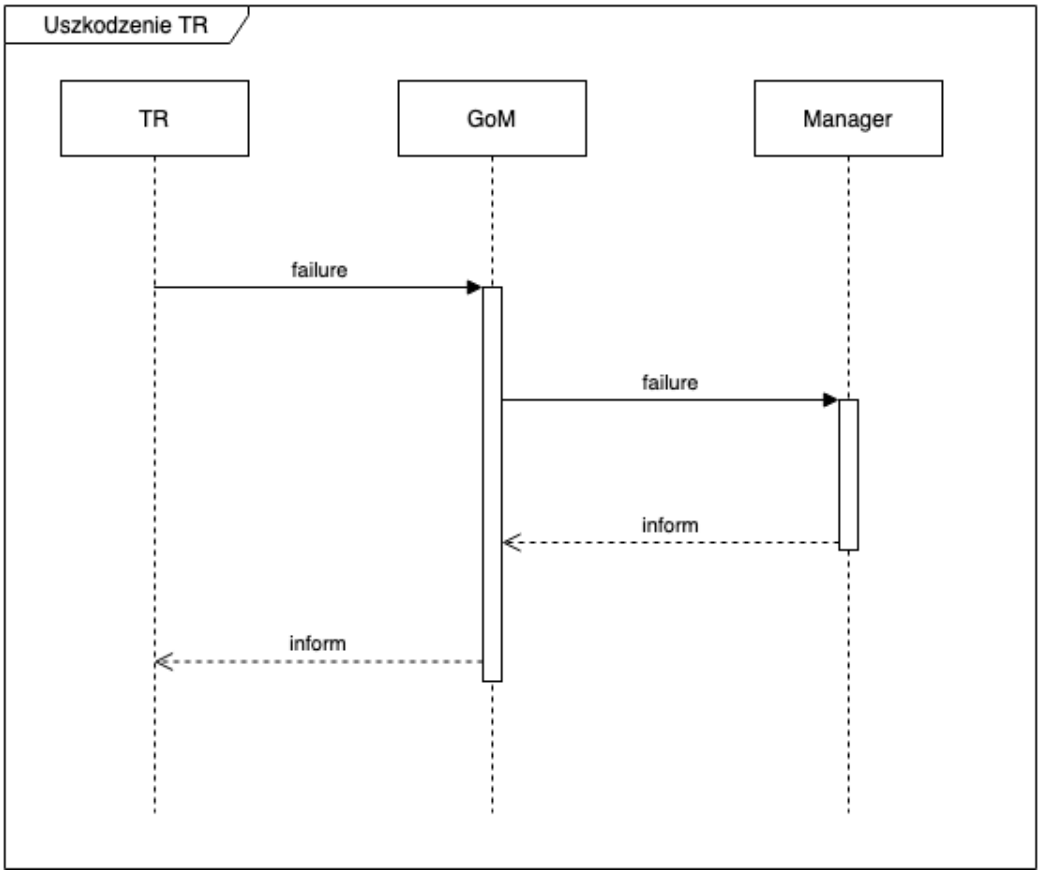
3.2.2 Funkcjonalność

- Odbiera zamówienia o różnych priorytetach od *Factory* i wstawia je do kolejki priorytetowej. Dzieje się to poprzez akceptację w protokole *FIPA-request*, nie może odmówić. (Rys. 1)
- Odbiera od *GoMów* wiadomości *failure* o awariach (samej maszyny jak i zepsucia *TRa*), aktualizuje kolekcję stanów i odpowiada wiadomością *inform*. (Rys. 3)
- Zleca wybranemu *GoMowi* wykonanie jednej operacji z zamówienia poprzez protokół *FIPA-request* (Rys. 2). *GoM* może odmówić gdy wszystkie maszyny w gnieździe będące w stanie wykonać zadaną operację są zepsute, nie potrafi wykonać danej operacji lub jest w trakcie wykonywania innego zamówienia. W przypadku akceptacji zamówienie jest zdejmowane z kolejki. Jeśli *GoM* odpowie niepowodzeniem to zamówienie jest ponownie wstawiane do kolejki. Natomiast w przypadku zakończenia operacji sukcesem również wraca do kolejki, ale jest przesuwany postęp prac w ciągu operacji.
- Informuje *Factory* o wykonaniu całego zamówienia kończąc protokół *FIPA-request*.

3.2.3 Komunikacja



Rysunek 2: Zlecenie pracy *GoMowi*



Rysunek 3: Uszkodzenie *TR*

3.3 Craftsman

3.3.1 Opis

Realizuje zlecenia przyjęte od *Managera*. Każdy *GoM* posiada zbiór maszyn, każde urządzenie wykonuje tylko jeden rodzaj operacji. Operacje te reprezentowane są abstrakcyjnie i mogą pokrywać się między różnymi *GoMami*. W danym momencie *GoM* może pracować nad jednym zamówieniem używając jednej maszyny. Z uwagi na złożoność procesu produkcyjnego, pojedynczy *GoM* zwykle nie jest w stanie samodzielnie zrealizować zlecenia. W tym celu każdemu *GoMowi* przyporządkowany jest jeden robot transportowy (*TR* 3.4).

3.3.2 Funkcjonalność

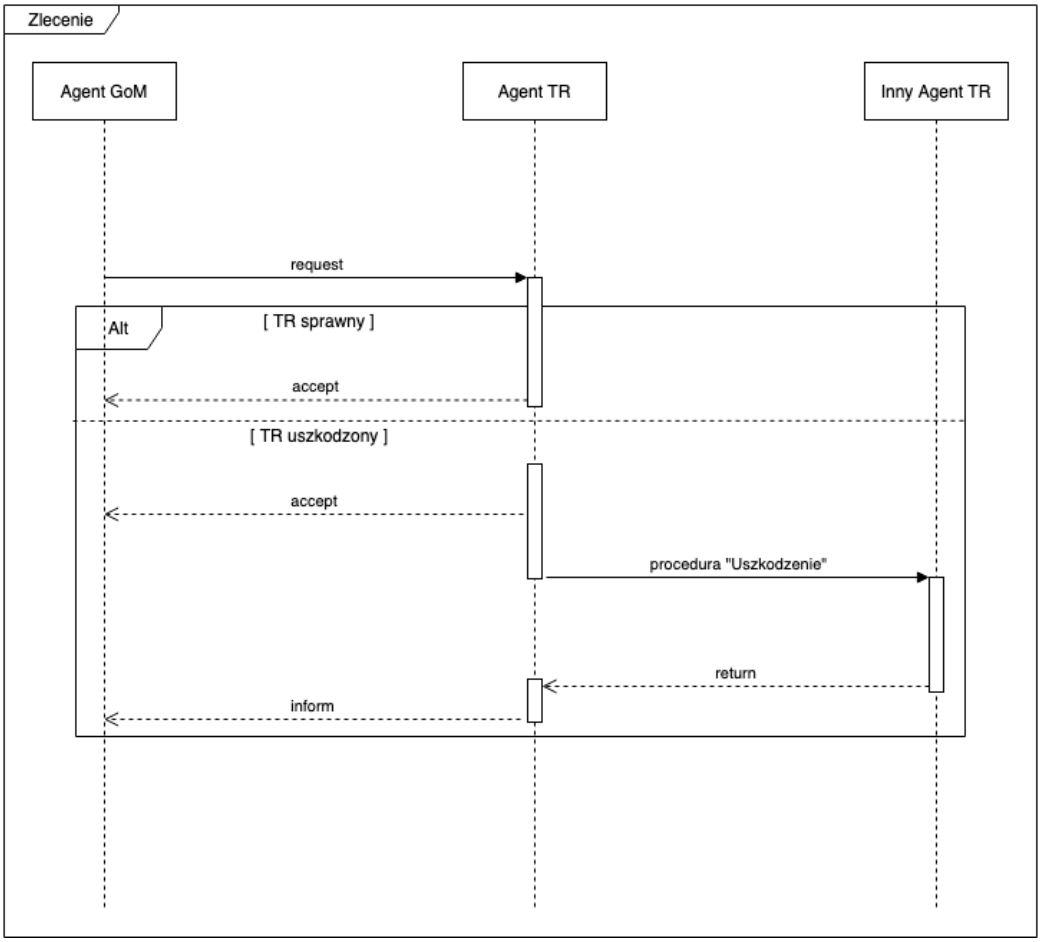
- Odbiera zlecenia wykonania jednej operacji na zamówieniu w protokole *FIPA-request*. Odmawia tylko jeśli wszystkie maszyny będące w stanie wykonać zadaną operację są zepsute, nie potrafi wykonać danej operacji lub jest w trakcie wykonywania innego zamówienia. Jeżeli w trakcie pracy zepsuje się maszyna to odpowiada niepowodzeniem, w przeciwnym razie po zakończeniu informuje o sukcesie kończąc protokół *FIPA-request*.

- Jeśli do wykonania zlecenia potrzebuje materiałów, które są poza jego zasięgiem, zleca transport swojemu *TR* z pomocą protokołu *FIPA-request*. *TR* zawsze akceptuje zlecenie (Rys. 4).

- Jeśli zdarzy się, że dany robot ulegnie uszkodzeniu, *GoM* przyjmuje od niego informację w postaci wiadomości *failure*, którą następnie przekazuje *Management Agentowi*. W odpowiedzi spodziewa się wiadomości *inform*, którą z kolei przekazuje do uszkodzonego *TRa* upewniając go w ten sposób, że cały system wie o uszkodzeniu (Rys. 3).

- Może zdarzyć się, że mimo uszkodzonego *robota transportowego GoM* dalej będzie musiał przyjmować zlecenia. W takiej sytuacji *GoM* spodziewa się wiadomości *inform* potwierdzającej pomyślną negocjację (Rys. 6).

3.3.3 Komunikacja



Rysunek 4: Wydawanie zlecenia *TRowi*.

3.4 Negotiator

3.4.1 Opis

Jako że *TR* nie jest standardowo w stanie samodzielnie poradzić ze zleceniem transportu, musi on we własnym zakresie zapewnić sobie pomoc niezbędną do jego wykonania. Przyjmujemy, że każdy *TR* dysponuje strukturą danych zawierającą AID wszystkich *TRów* oraz kolejką gromadzącą zlecenia (i związane z nimi dane pomocnicze) przy których jest on zaangażowany, które odbiera jako wiadomości bezpośrednio od swojego *GoMa*.

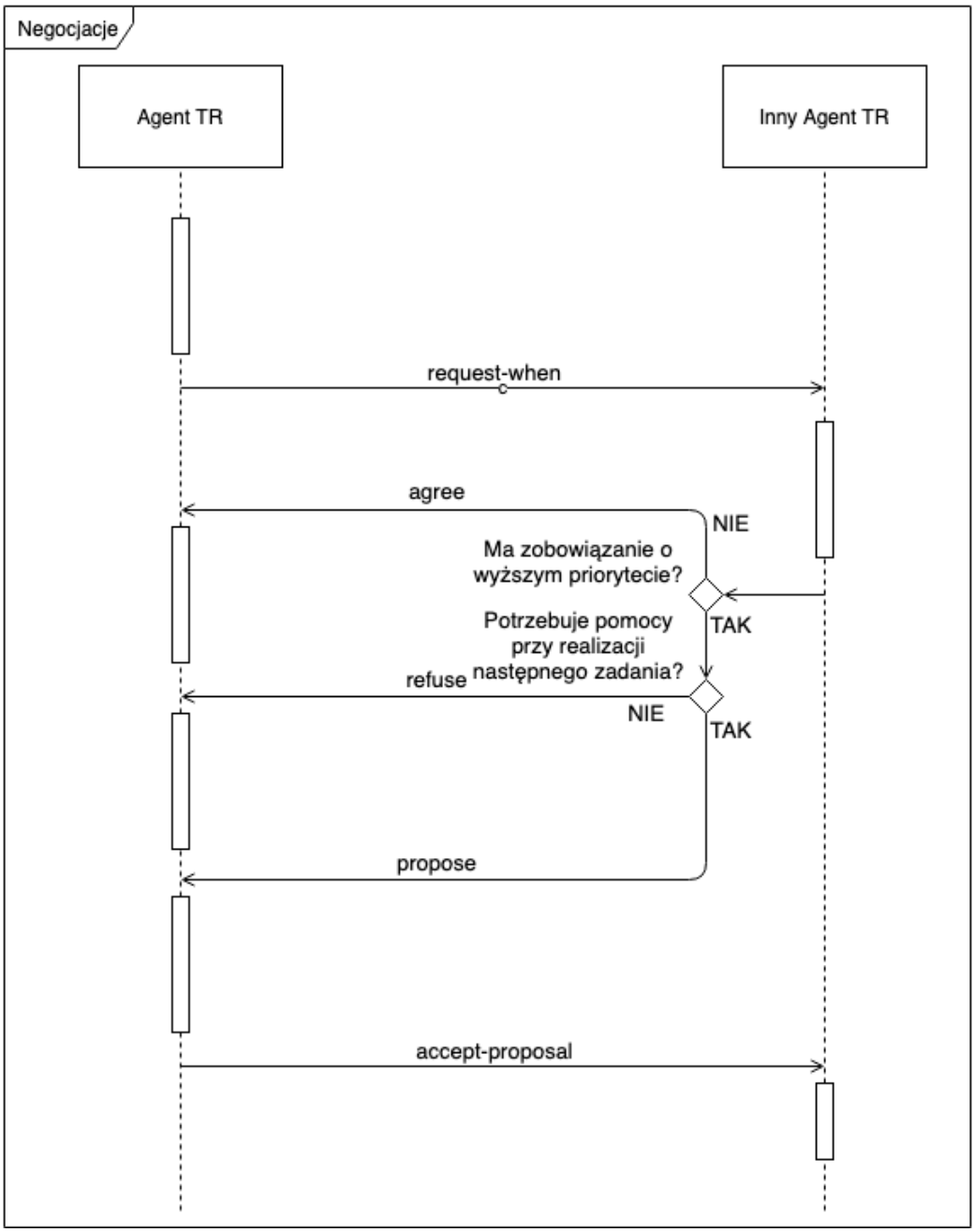
3.4.2 Funkcjonalność

– Przed rozpoczęciem pracy kolejka jest pusta, a kolejka AID uporządkowana rosnąco według fizycznej odległości stanowisk startowych w sensie czasu przemieszczania się między nimi. Kolejność AID będzie się zmieniała w miarę postępu prac tak, aby *TRy* które często ze sobą współpracowały w przeszłości prosiły się o pomoc jako pierwsze, co ma ograniczyć czasochłonne przepytywanie licznych agentów oraz zminimalizować czas oczekiwania na zwolnienie współpracowników. Kolejka będzie natomiast zasilana wiadomościami poprzez odbieranie przez *TR* stosownego komunikatu od *GoM* oraz na skutek negocjacji (z uwzględnieniem priorytetów zleceń).

– Punktem wyjścia głównej pętli obsługi zlecenia jest moment pobrania ze szczytu kolejki zlecenia, a głównym czynnikiem determinującym przebieg pracy jest czas potrzebny na wykonywanie zadań, tj. przewóz dóbr pomiędzy *GoMami* i przemieszczanie się między nimi w celu rozpoczęcia realizacji nowego zlecenia. Z kolekcji AID zostanie wybrany pierwszy inny *TR* i zostanie do niego wysłana prośba o pomoc. Tamten zgodzi się, jeśli różnica między ukończeniem zadania, które chwilowo wykonuje a podobnym zadaniem pytającego nie będzie zbyt duża (parametr systemu) oraz nie ma na szczycie kolejki innych zobowiązań o wyższym lub równym priorytecie, w przeciwnym razie, jeśli ma wystarczająco wielu zadeklarowanych współpracowników do wykonania swojego zadania, odmówi pytającemu pomocy. Gdyby natomiast sam miał zlecenie do którego realizacji potrzebuje pomocy, nawiąże z nim współpracę, tzn. zaangażuje się w transakcję wiązaną, polegającą na udzieleniu sobie wzajemnie wsparcia przy wykonywaniu tych zleceń po

kolei, w porządku ustalonym przez priorytet, potem potrzebny czas, potem drogą losowania. W ramach niej oba zadania zostaną dodane na szczyt kolejek obu graczy i wykonane, gdy tylko obaj gracze zakończą bieżące czynności, w takim wypadku nawet wiadomości o wyższym priorytecie będą odkładane przez nich obu na drugie miejsce w celu uniknięcia powtarzania procesu synchronizacji zwolnienia się uczestników (równoważnie priorytet zleceń wynikających ze współpracy jest wyższy). Współpracownicy przesuną się też wzajemie na szczyt swoich odpowiednich kolejek AID. Powyższy proces zleceniodawca powtarza do czasu, aż uda mu się zaangażować potrzebną liczbę agentów.

3.4.3 Komunikacja



Rysunek 5: Negocjacje między *TRami*

3.5 Transporter

3.5.1 Opis

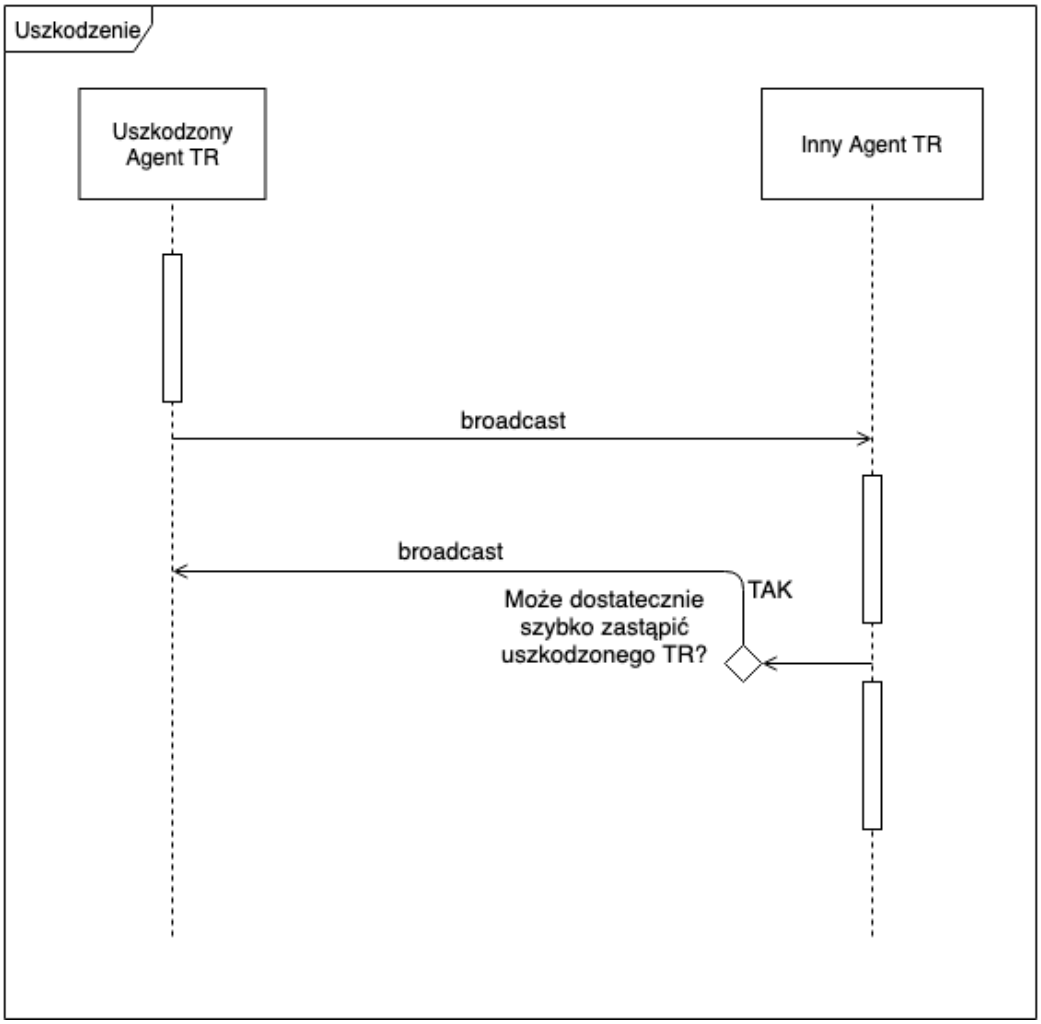
TR transportuje materiały między *GoMami* na ich zlecenie według priorytetów uwarunkowanych potrzebami fabryki i zleceń *GoMów*. Niezależnie od priorytetu, zadania nigdy nie będą przerywane po rozpoczęciu, jeśli mają odpowiadać rzeczywistemu przemieszczaniu się, jest to pozbawione sensu.

3.5.2 Funkcjonalność

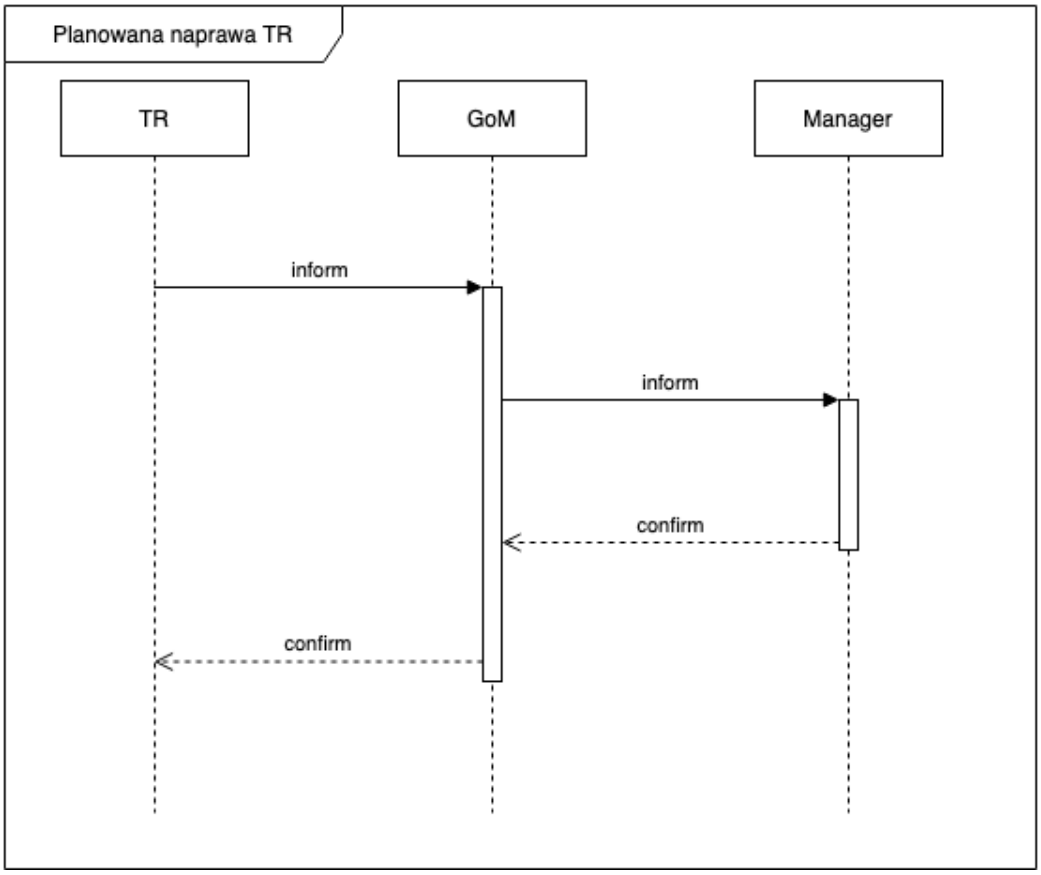
- Realizuje zadania od swojego *GoMa* oraz zlecenia wynegocjowane przez swoich współpracowników.
- *(Do dodania w późniejszym terminie)* Potrafi się zepsuć na nieznany z góry czas. Informuje o awarii swój *GoM*. Uszkodzony *TR* wciąż może otrzymywać zlecenia, wówczas prosi inne *TRy* o pomoc, w taki sam sposób jak przy wykonywaniu wspólnie innych zadań.
- Zakładamy, że czynność transportowania zajmuje przede wszystkim czas, ale nie zasoby obliczeniowe *TRa*, w związku z czym w czasie realizacji bieżącego zadania zostaną przeprowadzone wszystkie negocjacje dotyczące następnych.
- Jest uczciwy i zawsze zrealizuje zawarte kontrakty, o ile nie ulegnie awarii.
- W przypadku uszkodzenia *TRa*, przez dowolnego jego współpracownika rozsyłana jest wiadomość broadcast o najwyższym priorytecie do wszystkich pozostałych *TRów*. Pierwszy *TR*, który będzie w stanie dotrzeć na miejsce i dokończyć zadanie w dostatecznie krótkim czasie (parametr systemu), zgłasza się do tego zadania, również w trybie broadcast i udaje się je wykonać bez zaciągania przez kogokolwiek żadnych zobowiązań.
- Potrafi monitorować swój stan techniczny i wykrywać nieprawidłowości (konkretne parametry wewnętrzene nie są istotne dla naszego projektu). Umożliwia mu to planowanie przeglądu/naprawy przed uszkodzeniem. Dzięki temu może zapobiegać potencjalnym opóźnieniom wynikającym z nagłej potrzeby naprawy. Kiedy wykryje problem, informuje

swojego *GoMa*, który z kolei przekazuje informację *Managerowi*. (Rys. 7)

3.5.3 Komunikacja



Rysunek 6: Zgłoszenie uszkodzenia *TRom*



Rysunek 7: Zgłoszenie planowanej naprawy/przeglądu przez *TR*

4 Mapa znajomości agentów

	Factory	Management	GoM	TR
Factory		I		
Management	I		I, A	A
GoM		I		I
TR			I	I

Legenda

- **I:** Wchodzi w interakcję (czyt. wystąpienia "I" w wierszach *np.* Factory *wchodzi w interakcję z* Management Agentem).
- **A:** Jest zaznajomiony, posiada dany typ agenta w swojej strukturze `acquaintances` (czyt. wystąpienia "A" w wierszach *np. Typ agenta* Management Agent *jest zaznajomiony z typem agenta* GoM).

5 Implementacja

5.1 Repozytorium

https://github.com/MateuszZ3/industry2

5.2 Uproszczenia

Aktualnie przyjeliśmy kilka uproszczeń, które zostaną usunięte w miarę postępu prac. Na ten moment w fabryce nic się nie psuje i zamówienia są na tyle małe (waga półproduktów), że pojedynczy *TR* jest w stanie je przetransportować samodzielnie, współpraca nie jest wymagana.

5.3 GUI

Do implementacji interfejsu graficznego wybraliśmy bibliotekę *PyQT5*. Spośród dostępnych rozwiązań ta biblioteka oferuje największą stabilność, wydajność i przejrzystość, a także duże wsparcie społeczności. *GoMy* oraz *TRy* reprezentowane są jako punkty na płaszczyźnie wycentrowanej w oknie. Interfejs działa oczywiście w oddzielnym od agentów wątku. *Factory* komunikuje się z *GUI* poprzez zapewnione w *PyQT5* sygnały.

5.4 SPADE

Wszystkie agenty są inicjalizowane i uruchamiane przez *Factory*. Umożliwia to wstrzyknięcie adresów, identyfikatorów XMPP i haseł, które zdefiniowane są w ustawieniach. *GoMy* przyjmują zlecenia zgodnie z założeniami. Wysyłają informację o potrzebnym transporcie do swoich *TRów* jeśli nie posiadają zamówienia, które mają zrealizować u siebie. *TRy* realizują prośby swoich *GoMów*. W trakcie ich realizacji poruszają się z określoną prędkością po mapie i co stały interwał czasowy informują *GUI* o swoim położeniu.

6 Lista referencji

- https://spade-mas.readthedocs.io/en/latest/index.html
- https://jade.tilab.com/doc/tutorials/JADEProgramming-Tutorial-for-beginners.pdf
- https://www.geeksforgeeks.org/xmpp-protocol
- https://prosody.im/doc
- https://www.igniterealtime.org/projects/openfire/documentation.jsp
- https://www.learnpyqt.com/courses/custom-widgets/bitmap-graphics/
- https://kivy.org/doc/stable/tutorials/firstwidget.html