JADE – opracowanie do egzaminu

Mateusz Kaflowski AGH WIMIiP 2013

1. Jak uruchomić platformę JADE z GUI?

Uruchamiamy klasę:

Main class:

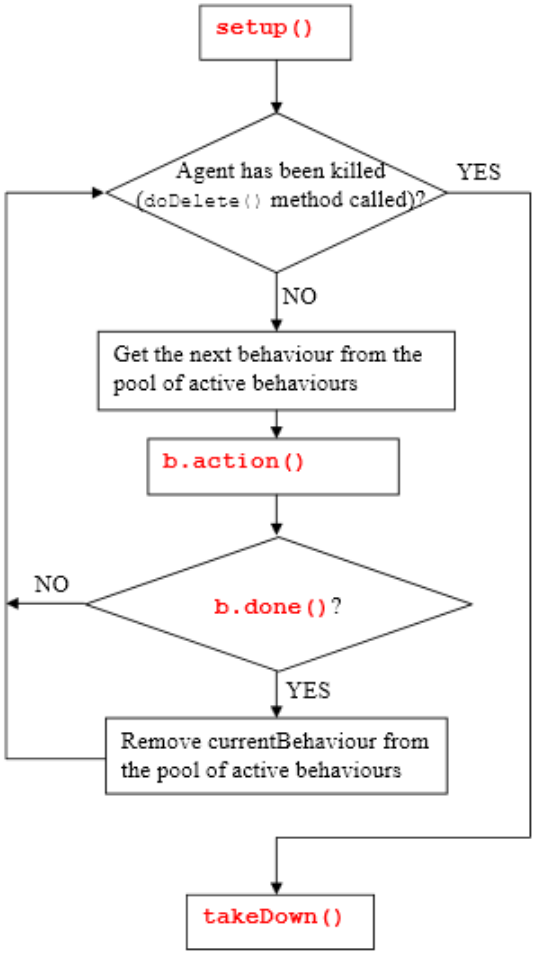
jade.Boot

Z argumentami:

Program arguments:

-gui jade.Boot

2. Cykl życia agenta:



3. Podstawowe rodzaje zachowań

- OneShotBehaviour – wykonuje akcję raz
- CyclickBehaviour – wykonuje akcję cały czas (“w kółko”)
- WakerBehaviour – wykonuje akcję po upływie podanego czasu
- TickerBehaviour – wykonuje akcję „w kółko” z odstępami czasowymi

4. **Zaimplementuj agenta, który będzie co 2s wypisywał “dziala”.**

```
public class NewAgent extends Agent {

    @Override
    protected void setup() {
        super.setup();

        addBehaviour(new TickerBehaviour(this,2000){

            @Override
            protected void onTick() {
                System.out.println("DZIALA");
            }

        });
    }

}
```

5. **Zaimplementuj agenta, który po 2s napisze „witaj”.**

```
public class NewAgent extends Agent {

    @Override
    protected void setup() {
        super.setup();

        addBehaviour(new WakerBehaviour(this, 2000) {
            @Override
            protected void handleElapsedTimeout() {
                System.out.println("witaj");
            }
        });
    }
}
```

6. Jak pobrać argumenty w agencie?

```
Object args[] = getArguments();
```

7. Napisz kod agenta, który wyśle wiadomość do innego agenta.

```
public class NewAgent extends Agent {

    @Override
    protected void setup() {
        super.setup();

        addBehaviour(new OneShotBehaviour() {

            @Override
            public void action() {

                ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
                msg.addReceiver(new AID("agent007", AID.ISLOCALNAME));
                msg.setContent("Wiadomosc testowa");
                send(msg);
            }
        });
    }

}
```

Wiadomość zostanie wysłana do agenta, którego nazwa lokalna to „agent001”

8. Przyjmij i wypisz wiadomość.

```
//...
addBehaviour(new CyclicBehaviour() {

    @Override
    public void action() {
        ACLMessage msg = receive();

        if(msg!=null){
            System.out.println(msg.getContent());
        }
        else {
            block();
        }
        /* block() "blokuje" zachowanie przez co
         * wypada ono z zadań zakolejkowanych do wykonania przez agenta.
         * Wszystkie zablokowane zachowania zostaja odblokowane gdy agent
         * dostaje wiadomość.
         */
    }
});
//...
```

9. Zarejestruj się w Yellow Pages.

```
//...
DFAgentDescription dfd = new DFAgentDescription();
dfd.setName(getAID());

ServiceDescription sd = new ServiceDescription();
sd.setName("Nazwa_serwisu");
sd.setType("Typ_serwisu");

dfd.addServices(sd);

try {
    DFService.register(myAgent, dfd);
} catch (FIPAException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
//...
```

10. Gdzie powinno się wyrejestrować z Yellow Pages?

Dobrą praktyką jest wyrejestrowywanie się z Yellow Pages gdy agent ginie (patrz cykl życia agenta).

```
@Override
protected void takeDown() {
    super.takeDown();

    try {
        DFService.deregister(this);
    } catch (FIPAException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

11. Wypisz wszystkich agentów, którzy zarejestrowali w Yellow Pages podaną usługę.

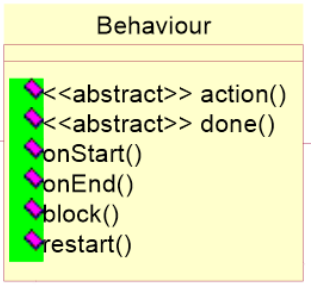
```
// ...
DFAgentDescription dfd = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
sd.setType("Typ_serwisu");

dfd.addServices(sd);

try {
    DFAgentDescription results[] = DFService.search(myAgent,
            dfd);
    for (int i = 0; i < results.length; i++) {
        System.out.println(results[i].getName().getLocalName());
    }
} catch (FIPAException e) {
    e.printStackTrace();
}
// ...
```

12. Metody onStart(), onEnd().

Znajdują się w zachowaniu i są jego „prologiem” i „epilogiem”. Są puste i można je nadpisać w klasach potomnych. Przed wykonaniem akcji z zachowania przez agenta zostanie wykonana metoda onStart() (jeden raz nawet dla CyclicBehaviour). Analogicznie jest na onEnd() tylko, że wykona się na koniec wykonywanej akcji. onEnd zwrana int’a – kod zakończenia zachowania.



13. Jak klonować i migrować agenta?

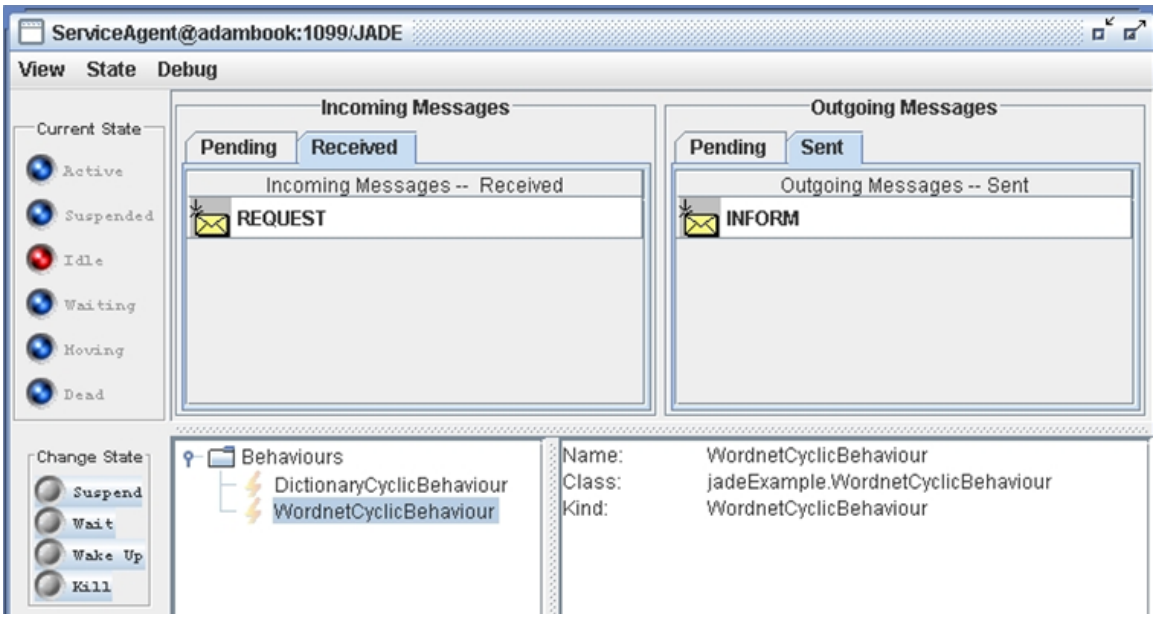
Metody w agencie: doClone(Location arg0, String arg1) i doMove(Location arg0).

14. Czy agent może zamknąć platformę?

AMS (Agent Management System) jest agentem specjalnym i tylko on posiada możliwość tworzenia i usuwania innych agentów, kontenerów lub zamknięcia platformy. Agent może jednak zażądać od AMS zamknięcia platformy.

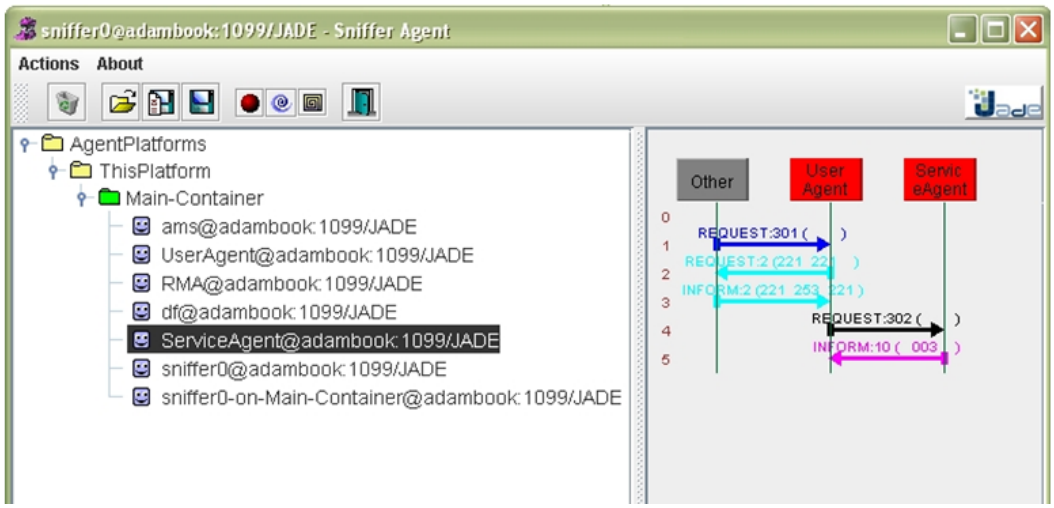
15. Inspector agent.

IntrospectorAgent pozwala na wybranie agenta i monitorowanie jego zachowań oraz wiadomości jakie do niego przychodzą i są przez niego wysyłane, możliwy jest zarówno podgląd kolejki wiadomości agenta (oczekujących na odebranie) jak i tych odebranych.



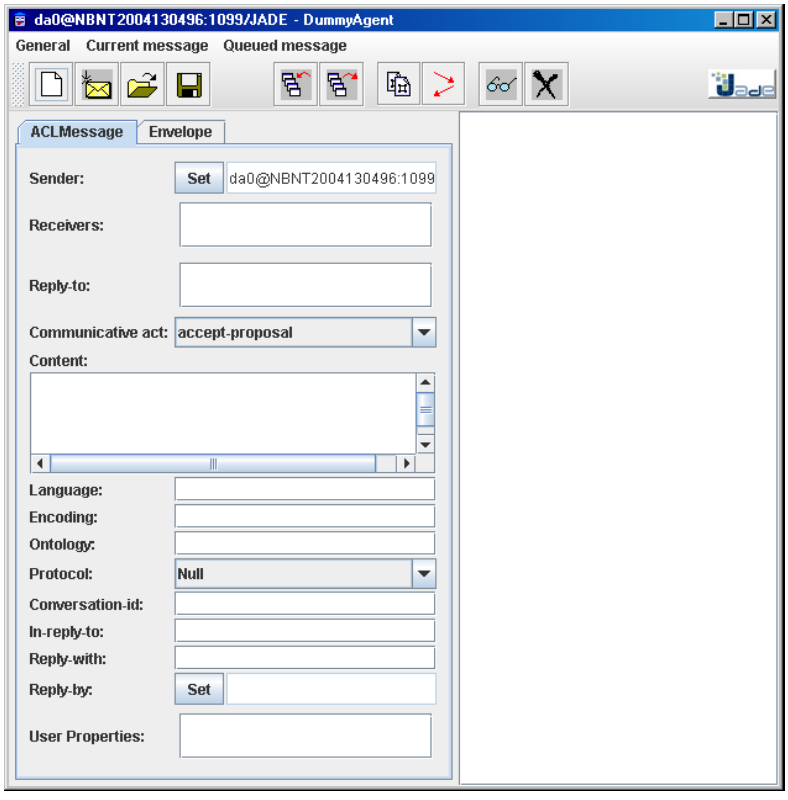
16. Sniffer agent.

Sniffer pokazuje przepływ wiadomości między agentami w perspektywie wielu agentów, które zostały wybrane jako przedmioty monitoringu. Można powiedzieć, że łapie on wiadomości w locie i pokazuje nam ich użycie. Tworzy diagramy zbliżone do UML. Przykładowy zapis interakcji pomiędzy agentami *rma* (gui), *UserAgent* oraz *ServiceAgent* wygląda następująco:



17. Dummy agent.

Jest narzędziem do monitorowania i debugowania. Tworzy graficzny interfejs. Używając GUI możemy tworzyć wiadomości ACL i wysyłać je do innych agentów. Można wyświetlić wszystkie wiadomości wysłane i otrzymane.



18. CompositeBehaviour

Ta klasa skomponowana jest z innych zachowań (dzieci). Operacje wykonywane są więc zdefiniowane w „dzieciach” natomiast sama klasa zajmuje się harmonogramowaniem tych operacji według określonych reguł. Sama klasa nie definiuje tych zasad a daje tylko interfejs. Zasady muszą być zdefiniowane w podklasach (SequentialBehaviour, ParallelBehaviour, FSMBehaviour).

19. Wymień możliwe tzw. performative wiadomości.

INFORM, REQUEST, AGREE, CANCEL, CONFIRM, DISCONFIM, FAILURE, UNKNOW, NOT_UNDERSTOOD, SUBSCRIBE itd.