JADE PROGRAMMING (LANJUTAN)

9.1 Pembahasan

Pemrograman Agent menggunakan JADE pada modul 9 ini membahas tentang komunikasi agent serta layanan *yellow pages*.

9.1.1 Komunikasi Agent

Komunikasi antar agent merupakan fitur penting dalam JADE. Untuk komunikasi antar *platform,* pesan akan diubah dari representasi java internal JADE, menjadi sintak-sintak, kode-kode, dan protokol-protokol *transport* yang mengikuti aturan FIPA (*Foundation for Intellegent Physical Agent*). FIPA sendiri adalah sebuah lembaga internasional yang mengembangkan standar-standart terkait dengan teknologi agen. Komunikasi antar agent dapat tercapai ketika terdapat pengiriman dan penerimaan pesan antara dua atau lebih agent.

Pertukaran pesan oleh agent JADE menggunakan format ACL (*Agent Communication Language*) yang didefinisikan oleh FIPA. Format ini mencangkup beberapa bidang, khususnya:

- Pengirim pesan (***sender***)
- Daftar ***receiver***
- Maksud dari komunikasi (***performative***), mengindikasikan tujuan dari pengirim mengirimkan pesan. Permormatif dapat berupa REQUEST jika pengirim mengiginkan penerima melakukan suatu tindakan, INFORM jika pengirim menginginkan penerima mengetahui suatu fakta, CFP (call for proposal), PROPOSE, ACCEPT_PROPOSAL, REJECT_PROPOSAL jika pengirin dan penerima terlibat dalam negoisasi, dan masih banyak lagi jenis permormatif lain.
- ***Content***, yaitu isi dari pesan
- Content ***language***, yaitu syntax yang digunakan untuk mengungkapkan content
- ***Ontology***, yaitu kosakata symbol yang digunakan dalam content dan makna yang tercangkup di dalamnya.
- Selain itu, ada beberapa cakupan lain yang diguankan untuk mengontrol percakapan dan menentukan timeout untuk menerima balasan, seperti ***conversation-id***, ***reply-with***, ***in-reply-to***, ***reply-by***

Pesan di dalam JADE diimplementasikan sebagai sebuah objek dari kelas `jade.lang.acl.ACLMessage` yang menyediakan method `set` dan `get` untuk menangani setiap bidang pesan. Code berikut ini menunjukkan pesan yang menginformasikan kepada sebuah agent bernama *peter* bahwa *today it's raining* :

```
ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
msg.addReceiver(new AID("Peter", AID.ISLOCALNAME));
msg.setLanguage("English");
msg.setOntology("Weather-forecast-ontology");
msg.setContent("Today it's raining");
send(msg);
```

Pengiriman Pesan

Tabel 9.1 berikut ini menunjukkan method yang umum digunakan untuk pengiriman pesan.

Tabel 9.1 Method pada JADE untuk pengiriman pesan

Method / Procedure	**Deskripsi**

`ACLMessage msg = new ACLMessage(ACLMessage.INFORM)`
`thows`
`jade.lang.acl.ACLMessage`
	Memanggil Class Object ACLMessage untuk tempat dari pesan yang akan dikirim. Nama dari Class Object ACLMessage yaitu "msg"

`Public void setContent()`
`throws`
`jade.lang.acl.ACLMessage`
	Memanggil Methode dari kelas ACLMessage untuk pengisian pesan

`Public void addReceiver()`
`throws`
`jade.lang.acl.ACLMessage`
	Methode yang digunakan untuk menambah nama agent sebagai penerima pesan

`Public void send()`
`throws`
`jade.lang.acl.ACLMessage`
	Mengirimkan pesan sesuai dengan agent tujuan

Penerimaan Pesan

Tabel 9.2 berikut ini menunjukkan method dan konstraktor yang umum digunakan pada penerima pesan.

Tabel 9.2 Method pada JADE untuk menerima pesan

Method / Procedure	**Deskripsi**

`Public void receive()`
`throws`
`jade.lang.acl.ACLMessage`
	Methode untuk penerima kiriman

`Public void getContent()`
`throws`
`jade.lang.acl.ACLMessage`
	Methode untuk membaca isi dari pesan yang didapat

`Public void createReply()`
`throws`
`jade.lang.acl.ACLMessage`
	Methode untuk membalas kiriman pesan secara otomatis.

`ACLMessage msg =new ACLMessage()`
`throws`
`jade.lang.acl.ACLMessage`
	Memanggil Class Object ACLMessage untuk memanggil methode yang digunakan untuk penerimaan dan membalas pesan otomatis

Seleksi Pesan menggunakan Message Template

Message Template digunakan untuk menjamin bahwa pesan yang diterima oleh suatu agent, adalah yang tepat dan di proses dengan behaviour yang tepat pula. Oleh karena itu dengan adanya *Message Template* pada setiap behaviour yang menerima pesan, maka penyaringan pesan yang masuk akan lebih mudah. **Tabel 9.3** berikut ini menunjukkan method dan konstraktor dasar penggunaan Message Template

Tabel 9.3 Method dasar penggunaan Message Template

Method / Procedure	**Deskripsi**

`MessageTemplate nama = MessageTemplate.and();`
`thows`
`jade.lang.acl.*;`
	Memanggil Class Object MessageTemplate untuk tempat dari template. Nama dari Class Object MessageTemplate yaitu “nama”

`Public voidMatchSender(new AID("broker", AID.ISLOCALNAME))`
`throws`
`jade.lang.acl.*;`
	Memanggil Method dari kelas MessageTemplate untuk penyeleksian agen pengirim pesan. nama agen yang diperbolehkan : broker

`Public void MatchPerformative(ACLMessage.INFORM)`
`throws`
`jade.lang.acl.*;`
	Methode yang digunakan untuk pengecekkan performativenya

Sebagai contoh, berikut ini ditunjukkan pada behavior agent yang bersifat cyclic dilakukan penyaringan pesan dengan tipe pervormatif CFP sehingga aksi dilakukan ketika agent menerima pesan CFP.

```
public void action() {
 MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.CFP);
 ACLMessage msg = myAgent.receive(mt);
 if (msg != null) {
       // CFP Message received. Process it
       ...
 }
 else {
       block();
 }
}
```

Contoh 9.1. Komunikasi Agent

Kelas Pengirim

```
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.TickerBehaviour;
import jade.lang.acl.ACLMessage;

public class sender extends Agent {

    @Override
    protected void setup() {
        addBehaviour(new TickerBehaviour(this, 1000) {

            @Override
            protected void onTick() {
                System.out.println(myAgent.getLocalName() + " mengirim pesan");
                ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
                msg.addReceiver(new AID("AgentPenerima", AID.ISLOCALNAME));
                msg.setContent("Ini pesan dari " + myAgent.getLocalName());
                send(msg);
            }
        });
    }

}
```

Kelas Penerima

```
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.CyclicBehaviour;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;

public class receiver extends Agent {
    protected void setup() {
        addBehaviour(new CyclicBehaviour(this) {
            public void action() {
                //seleksi pesan berdasarkan performative
                MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.INFORM);
                //seleksi pesan berdasarkan nama agent pengirim dan permormative
                MessageTemplate mt2 = MessageTemplate.and(MessageTemplate.MatchSender(
                        new AID("pengirim", AID.ISLOCALNAME)),
                        MessageTemplate.MatchPerformative(ACLMessage.INFORM));

                ACLMessage msg = myAgent.receive(mt);
                String content;
                if (msg != null) {
                    content = msg.getContent();
                    System.out.println("Pesan yang diterima: " + content);
                } else {
                    block();
                }
            }
        });
    }
}
```

Konfigurasi untuk Menjalankan Agent

Untuk menjalankan kedua kelas tersebut secara bersamaan maka lakukan pengaturan konfigurasi dengan menuliskan **jade.Boot** pada main class, sedangkan pada argurment tuliskan:

–gui namaAgent:packace.namaKelas

misal:

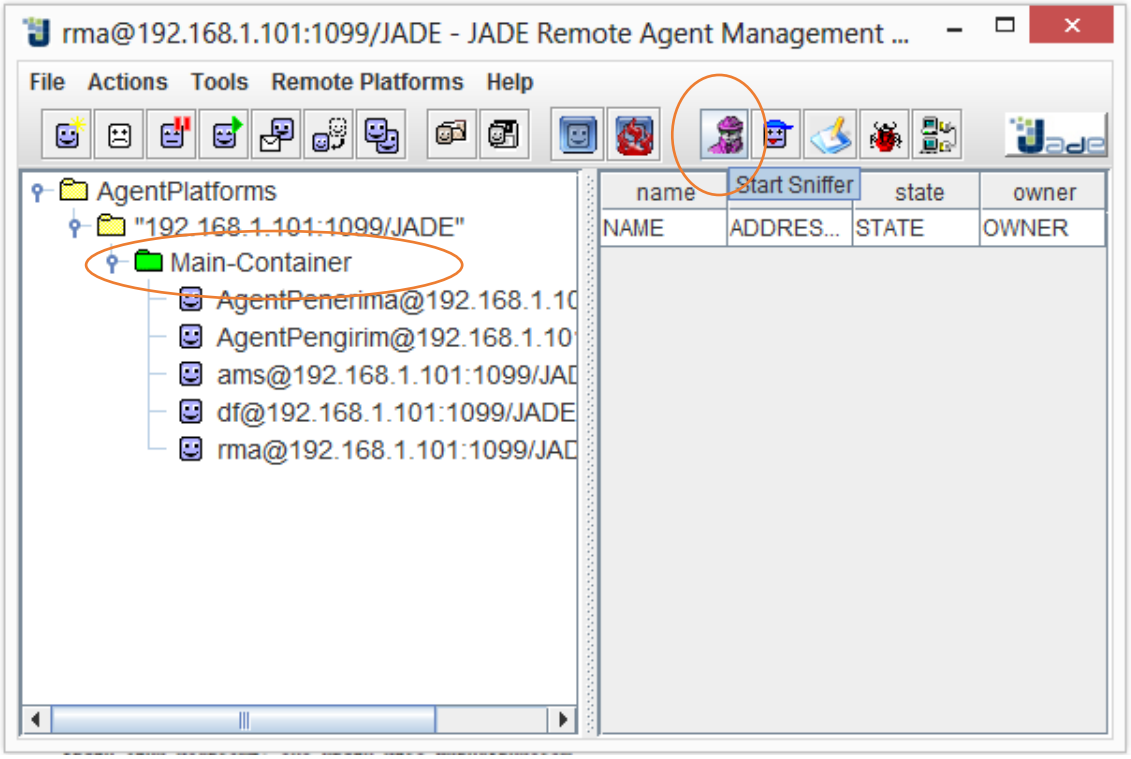
-gui AgentPenerima:JADE.receiver;AgentPengirim:JADE.sender

Begitu dijalankan akan tampil output:

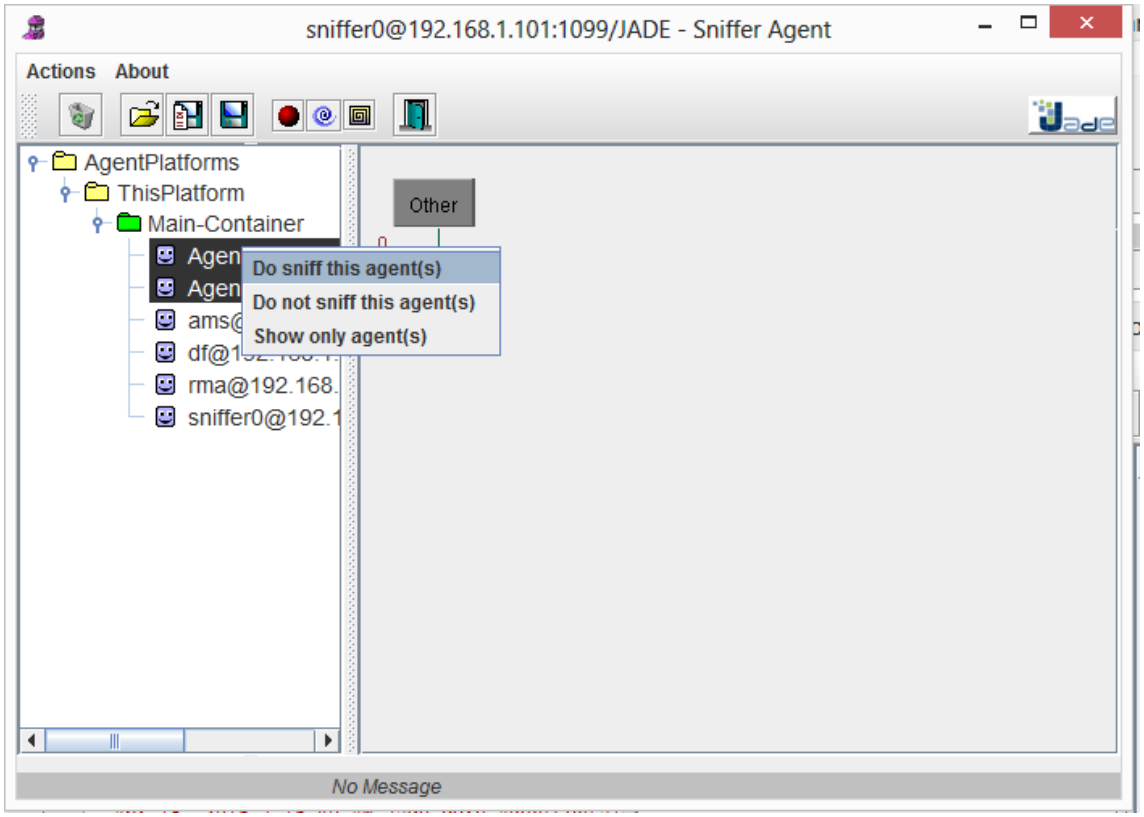
```
----------------------------------------
Apr 18, 2016 7:16:40 AM jade.imtp.leap.LEAPIMTPManager initialize
INFO: Listening for intra-platform commands on address:
- jicp://192.168.1.101:1099

Apr 18, 2016 7:16:40 AM jade.core.BaseService init
INFO: Service jade.core.management.AgentManagement initialized
Apr 18, 2016 7:16:40 AM jade.core.BaseService init
INFO: Service jade.core.messaging.Messaging initialized
Apr 18, 2016 7:16:40 AM jade.core.BaseService init
INFO: Service jade.core.resource.ResourceManagement initialized
Apr 18, 2016 7:16:40 AM jade.core.BaseService init
INFO: Service jade.core.mobility.AgentMobility initialized
Apr 18, 2016 7:16:40 AM jade.core.BaseService init
INFO: Service jade.core.event.Notification initialized
Apr 18, 2016 7:16:40 AM jade.mtp.http.HTTPServer <init>
INFO: HTTP-MTP Using XML parser com.sun.org.apache.xerces.internal.jaxp.SAXParserImpl$JAXPSAXParser
Apr 18, 2016 7:16:40 AM jade.core.messaging.MessagingService boot
INFO: MTP addresses:
http://Khiznia:7778/acc
Apr 18, 2016 7:16:40 AM jade.core.AgentContainerImpl joinPlatform
INFO: ----------------------------------------
Agent container Main-Container@192.168.1.101 is ready.
--------------------------------------------
AgentPengirim mengirim pesan
Pesan yang diterima: Ini pesan dari AgentPengirim
AgentPengirim mengirim pesan
Pesan yang diterima: Ini pesan dari AgentPengirim
AgentPengirim mengirim pesan
Pesan yang diterima: Ini pesan dari AgentPengirim
AgentPengirim mengirim pesan
Pesan yang diterima: Ini pesan dari AgentPengirim
AgentPengirim mengirim pesan
```

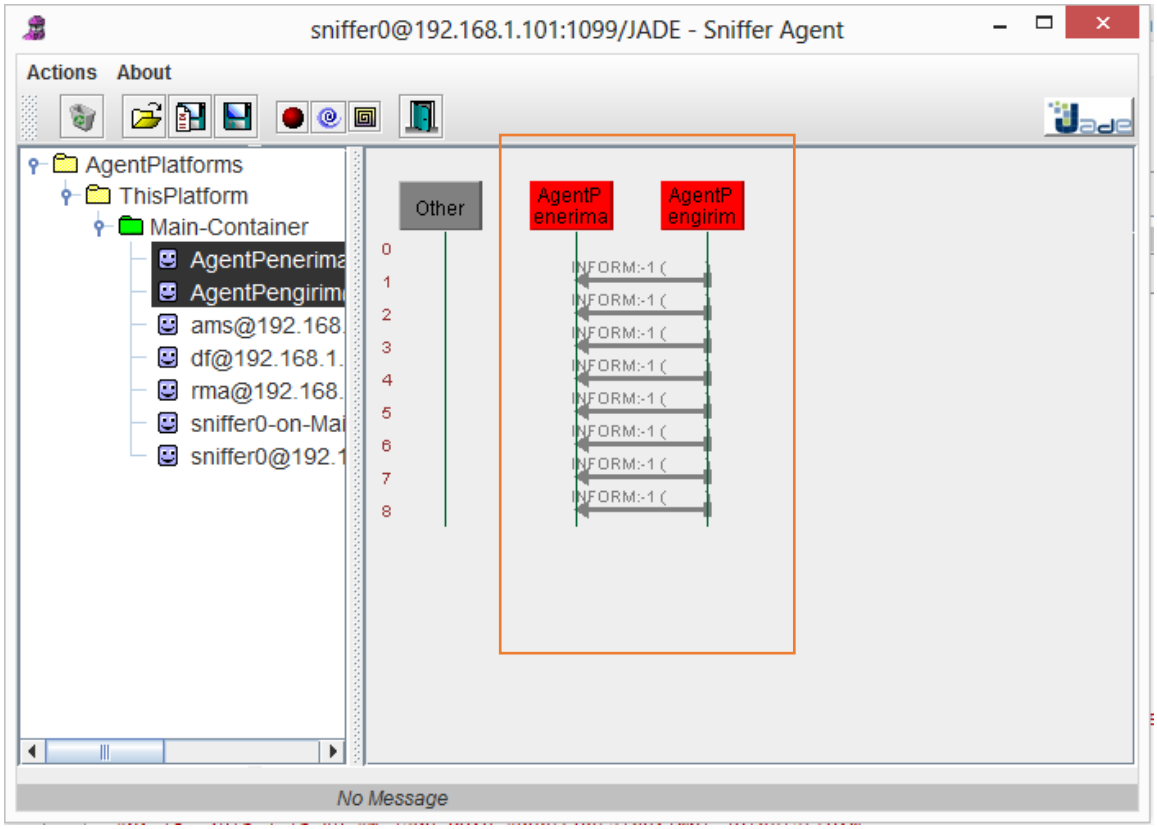
Sniffing

Untuk melihat komunikasi antar agent, dapat menggunakan *sniff* pada RMA. Pastikan kursor aktiv pada main-container kemudian klik icon "*sniffer*" untuk melakukan *sniffing*.



Klik kanan pada agent kemudian pilih “Do sniff this agent(s)” sehingga tampak arah komunikais dan jenis pesan yang dikirim.

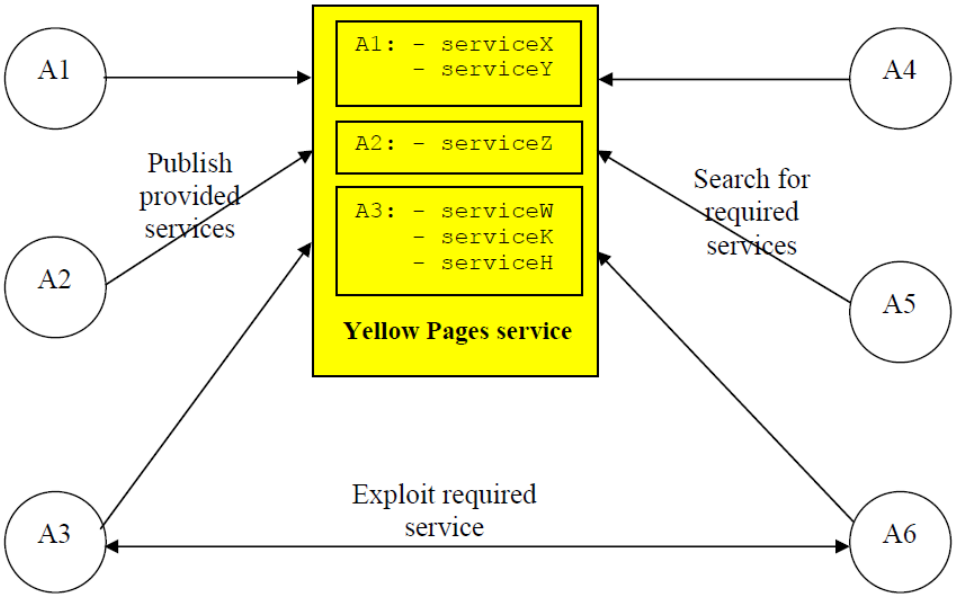




9.1.2 Yellow Pages

Layanan *yellow pages* memungkinkan agen untuk mempublikasikan deskripsi dari satu atau lebih layanan yang mereka berikan agar agen lain dapat dengan mudah menemukan mereka. Layanan yellow pages di JADE, sesuai dengan spesifikasi Agen Manajemen FIPA, disediakan oleh agen khusus yang disebut DF (*Directory Fasilitator*).

Setiap agen bisa mendaftar (*publish*) layanan dan mencari (*discover*) jasa. Pendaftaran, *deregistrations*, modifikasi dan pencarian dapat dilakukan setiap saat selama agen masih hidup. Hal ini digambarkan dalam gambar berikut:



Gambar 9.1 Layanan Yellow Pages

Selama DF adalah agen, hal tersebut memungkinkan untuk berinteraksi dengan agen lainnya dengan bertukar *ACLMessage* menggunakan bahasa konten yang seperti yang didefinisikan dalam spesifikasi FIPA.

Contoh 9.2. Yellow Pages

Pada contoh ini terdapat tiga kelas yang ditulis terpisah, yaitu: (1) kelas “kirim” sebagai pengirim informasi, (2) kelas “broker” sebagai perantara yang mengirimkan informasi dari pengirim ke beberapa penerima serta sebagai pencari agent penerima yang masih hidup, dan (3) kelas “terima” sebagai penerima pesan.

Kelas Kirim

```
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.CyclicBehaviour;
import jade.lang.acl.ACLMessage;

public class kirim extends Agent{
protected void setup(){
    addBehaviour(new CyclicBehaviour(this) {

        @Override
        public void action() {
            ACLMessage msg= new ACLMessage(ACLMessage.SUBSCRIBE);
            msg.setConversationId("kirim");
            msg.setContent("dari "+myAgent.getLocalName()+" apa kaabr? \n");
            msg.addReceiver(new AID("broker", AID.ISLOCALNAME));
            myAgent.send(msg);
            block(2000);
        }
    });
}
}
```

Kelas Broker

```
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.CyclicBehaviour;
import jade.core.behaviours.TickerBehaviour;
import jade.domain.DFService;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.domain.FIPAException;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import java.util.Vector;
import java.util.logging.Level;
import java.util.logging.Logger;

public class broker extends Agent {

    Vector aTerimaBaru = new Vector();

    @Override
    protected void setup() {
        addBehaviour(new behav(this, aTerimaBaru));
        addBehaviour(new cari_df(this, 10000, aTerimaBaru));
    }

    class behav extends CyclicBehaviour {...40 lines }

    class cari_df extends TickerBehaviour {...28 lines }
}
```

```
class behav extends CyclicBehaviour {
    MessageTemplate mt_Kirim;
    boolean StaKirim = false, StaTerima = false;
    Vector vTerima;
    int iTerima = 0;
    public behav(broker aThis, Vector aTerimaBaru) {
        vTerima = aTerimaBaru;
        mt_Kirim = MessageTemplate.MatchConversationId("kirim");
    }

    public void action() {
        //terima pesan dari terima
        ACLMessage msgKrmPesan = myAgent.receive(mt_Kirim);
        // terima pesan dari kirim
        if (msgKrmPesan != null) {
            if (vTerima.size() > 0) {
                ACLMessage Krmbalas = new ACLMessage(ACLMessage.PROPOSE);
                Krmbalas.setContent(msgKrmPesan.getContent());
                Krmbalas.addReceiver((AID) vTerima.elementAt(iTerima++));
                myAgent.send(Krmbalas);
                if (iTerima >= vTerima.size()) {
                    iTerima = 0;
                }
                StaKirim = true;
                System.out.print("size ok\n");
            }
            System.out.print("msg ok\n");
        } else {
            StaKirim = false;
        }
        if (!(StaKirim || StaTerima)) {
            block(50);
        }
    }
}
```

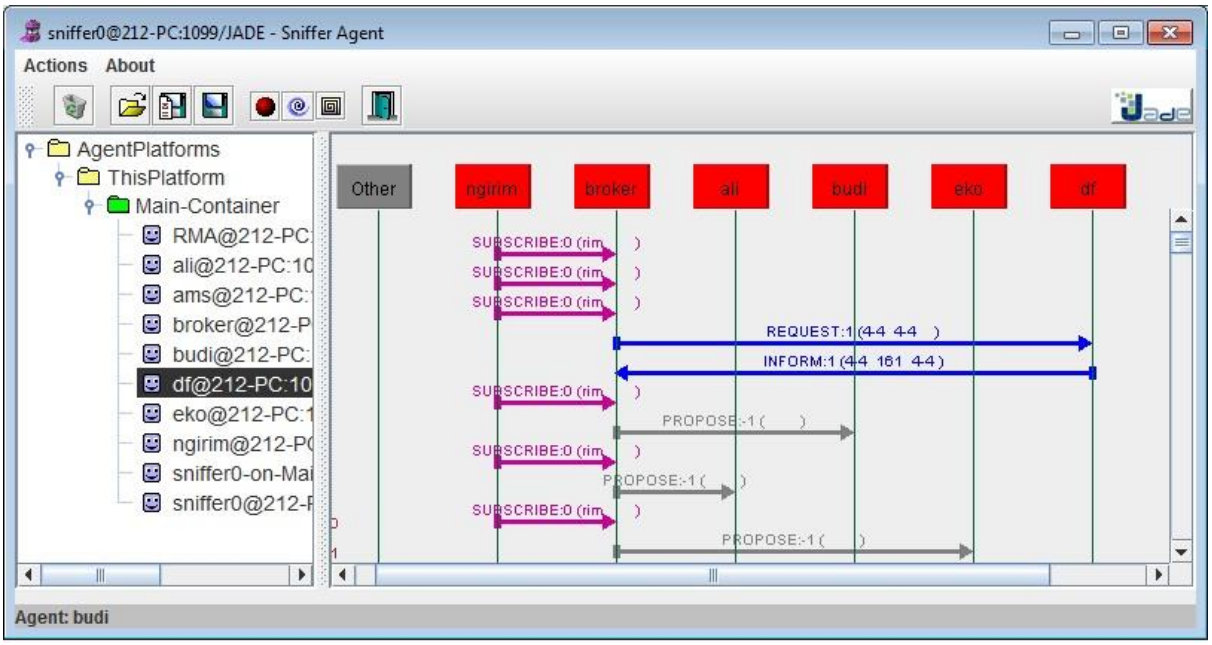
```
class cari_df extends TickerBehaviour {

    String[] aTerimaBaru;
    DFAgentDescription tm = new DFAgentDescription();
    ServiceDescription sd = new ServiceDescription();
    Vector vTerima;

    public cari_df(Agent aThis, int i, Vector aTerimaBaru) {
        super(aThis, i);
        sd.setType("terima");
        tm.addServices(sd);
        vTerima = aTerimaBaru;

    }
```

```
        @Override
        protected void onTick() {
            try {
                DFAgentDescription[] dfPenerima = DFService.search(myAgent, tm);
                vTerima.clear();
                for (int i = 0; i < dfPenerima.length; i++) {
                    vTerima.addElement(dfPenerima[i].getName());
                }
            } catch (FIPAException ex) {
                Logger.getLogger(broker.class.getName()).log(Level.SEVERE, null, ex);
            }
        }
    }
```

Kelas Terima

```
public class terima extends Agent {
    protected void setup() {
        //registrasi agent
        DFAgentDescription dfad = new DFAgentDescription();
        dfad.setName(getAID());
        ServiceDescription sd = new ServiceDescription();
        sd.setType("terima");
        sd.setName(getLocalName());
        dfad.addServices(sd);
        try {
            DFService.register(this, dfad);
        } catch (FIPAException ex) {
            Logger.getLogger(terima.class.getName()).log(Level.SEVERE, null, ex);
        }

        addBehaviour(new CyclicBehaviour(this) {
            public void action() {
                ACLMessage pesan = myAgent.receive();
                if (pesan != null) {
                    String msg = pesan.getContent();
                    System.out.print("terima : " + myAgent.getLocalName() +
                            " : " + msg + "\n");
                } else {
                    block(200);
                }
            }
        });
    }
}
```

Hasil dari kode program diatas jika dijalankan dengan isi dari argumentnya: '**-gui ngirim:JADE.kirim;broker:JADE.broker;budi:JADE.terima;eko:JADE.terima;ali:JADE.terima**', sebagai berikut :



Berdasarkan hasil tersebut dapat di jabarkan, bahwa broker me-*request* ke DF (*Directory Fasilitator*) tentang pencarian agent penerima dengan type "terima". Kemudian DF mengirim kembali kepada broker dan broker melakukan pekerjaannya yaitu sebagai perantara informasi dari 1 pengirim ke 3 penerima sekaligus.

9.2 Praktikum

1. Tulis dan jalankan **Contoh 9.1** sesuai langkah pada modul ini dan pahami tiap barisnya!
2. Tulis dan jalankan **Contoh 9.2** sesuai langkah pada modul ini dan pahami tiap barisnya!

9.3 Tugas

1. Pelajari contoh book trading untuk menjawab pertanyaan berikut:
   a. Jelaskan baris program yang menunjukkan komunikasi agent. Jelaskan ACLMessage dan MessageTemplate yang digunakan.
   b. Jelaskan alur kumunikasi agent dengan menjalankan Sniff
   c. Jelaskan layanan DF pada book trading dan jelaskan baris programnya
2. Buatlah sebuah aplikasi travel berbasis agent!
3. Bagaimana konfigurasi project agar aplikasi yang anda buat pada tugas 2 dapat dijalankan di beberapa device? Simulasikan jalannya aplikasi menggunakan 4 device (minimal)