JADE – Comunicación entre agentes

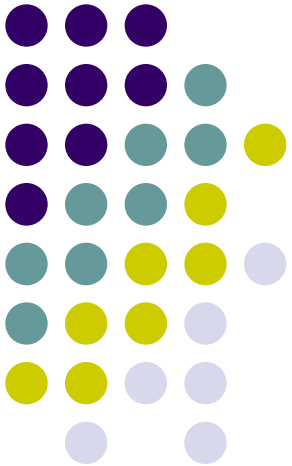
Taller de sistemas multiagentes

Prof. Dr. Ariel Monteserin

amontese@exa.unicen.edu.ar

ISISTAN –Fac. Cs. Exactas – UNICEN

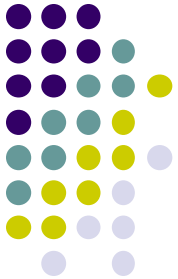
Tandil, Argentina



JADE Distribuido

```
import jade.wrapper.*;
…
AgentContainer c = getContainerController();
   try {
        AgentController a = c.createNewAgent( "X1",
        "MiAgente", null );
        a.start();
   }
   catch (StaleProxyException e){}
…
```

JADE Distribuido



- `jade.core.Runtime.instance()`
  - Instancia singleton de JADE Runtime.
- `Profile - ProfileImpl`
  - Mantiene las opciones de configuración.
  - `get/setParameters(String key, String value)`
    - `MAIN_HOST`: nombre o IP del host que contiene el contenedor principal.
    - `MAIN_PORT`: puerto donde el contenedor principal está escuchando registraciones de contenedores.

- `ContainerController Runtime.createAgentContainer(Profile)`

JADE Distribuido

```
public static void main(String[] args) {

   Runtime rt = Runtime.instance();

   Profile p = new ProfileImpl();
   p.setParameter(Profile.MAIN_HOST, "192.168.0.123");
   p.setParameter(Profile.MAIN_PORT, "1099");

   ContainerController cc = rt.createAgentContainer(p);

   try {
      AgentController r1 =
        cc.createNewAgent("R1","src.RemoteAgent", null);
      r1.start();
   }
   catch (StaleProxyException spe) {}
}
```

Instancia local de la plataforma

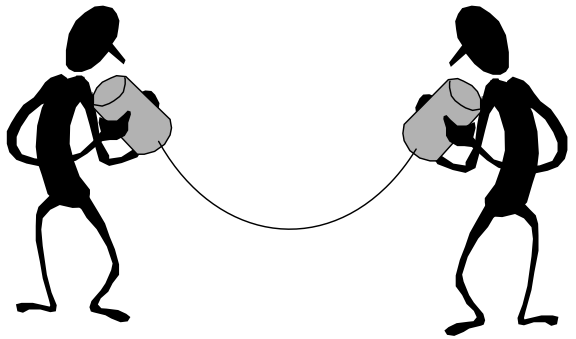
Ubicación del Main container

Plataforma distribuida desde línea de comandos

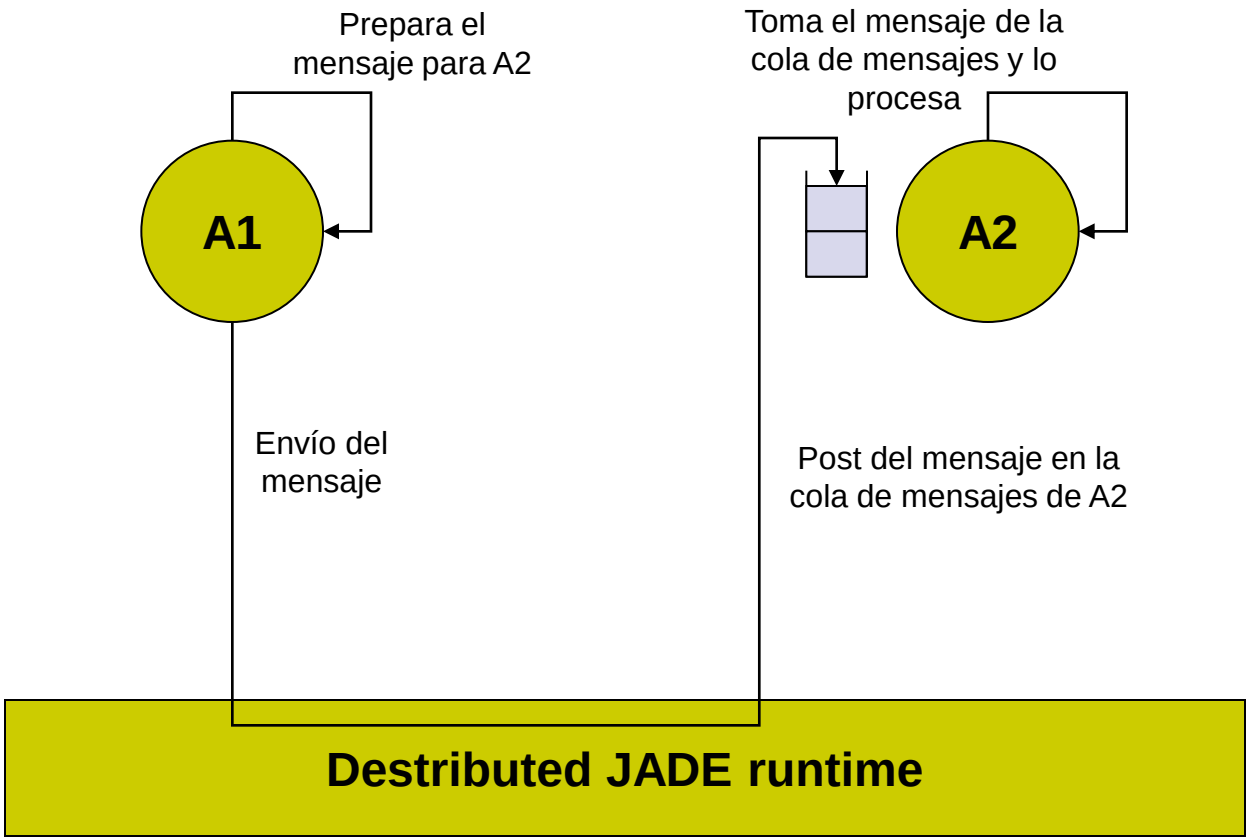
- Desde línea de comandos usar:
  - -container: especifica que la instancia de JADE es un contenedor.
  - -host: nombre o IP del host donde se ubica el main-container.
  - -port: puerto del host donde se ubica el main-container.
- Extras
  - -container-name: nombre del container.
- Crear GUI remotamente
  - RMA2:jade.tools.rma.rma

Comunicación entre agentes

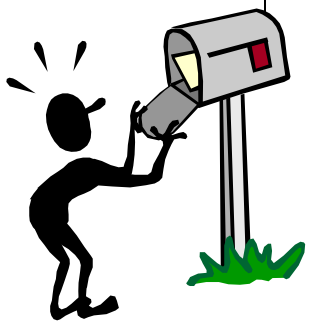
- Paradigma de intercambio de mensajes asincrónico.
- Estándar FIPA
- Utiliza FIPA ACL (Agent Communication Language)



Comunicación entre agentes

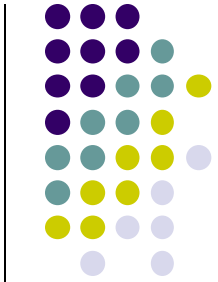


ACL (Agent Communication Language)



- Definido por FIPA
- El mensaje ACL posee los campos:
  - **sender**: agente que envía el mensaje.
  - **receivers**: lista de receptores.
  - **performative**: tipo de mensaje. Indica la intención que el emisor intenta lograr enviando el mensaje.
  - **content**: contenido principal del mensaje.
  - **language**: lenguaje usado en el contenido. P.e. la sintaxis utilizada para expresar el contenido.
  - **ontology**: ontología usada en el contenido. P.e. el vocabulario de símbolos usados en el contenido y su significado.
  - **conversation-id, reply-with, in-replay-to, reply-by**: para el control de conversaciones concurrentes.

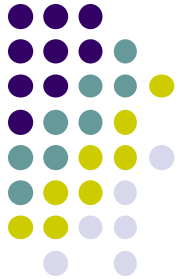
Performativas



- Cada mensaje representa un “acto de habla”
  - Precondiciones de factibilidad
  - Efecto racional

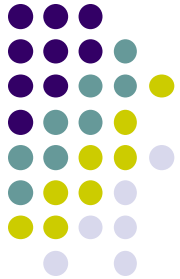
- Haber ejecutado el acto no garantiza el cumplimiento de efecto racional
  - Autonomía

ACL (Communicative Acts) I



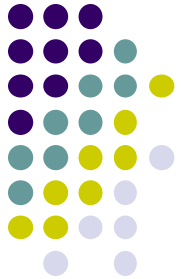
- Tipo de mensajes (performative)
  - `ACCEPT_PROPOSAL`: acepta una propuesta sobre la ejecución de una acción.
  - `AGREE`: expresa el acuerdo sobre la ejecución futura de una acción.
  - `CANCEL`: expresa desacuerdo del emisor sobre la ejecución de una acción por parte del receptor.
  - `CFP`: (call for proposal) solicita una propuesta.
  - `CONFIRM`: el emisor confirma al receptor la validez de una proposición.
  - `DESCONFIRM`: el emisor confirma al receptor que una proposición es falsa.
  - `FAILURE`: comunica que una acción que intentaba ejecutar falló.

ACL (Communicative Acts) II



- `INFORM`: informa que una proposición es verdadera.
- `NOT UNDERSTOOD`: el emisor E informa al receptor R que percibió que R ejecutó alguna acción, pero que E no la comprende.
- `PROPOSE`: envía una propuesta para ejecutar cierta acción, dada cierta precondición.
- `QUERY IF`: consulta a otro agente si una proposición es verdadera o no.
- `REFUSE`: rechaza ejecutar una acción, y explica los motivos del rechazo.
- `REJECT PROPOSAL`: rechaza una propuesta durante una negociación.
- `REQUEST`: solicita la ejecución de una acción.

Envío de mensaje



```
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);
```

- Dentro de un comportamiento:
  `myAgent.send(msg);`
  - `myAgent` es una referencia al agente que posee el comportamiento

Recepción de mensajes

- Usando `blockingReceive()`
  - `ACLMessage msg = myAgent.blockingReceive();`
  - Suspende todas las actividades del agente

Implementar un comportamiento que reciba mensajes desde la plataforma, utilizando blockingReceive() y los responda, y otro cíclico que muestre un contador.

Recepción de mensajes

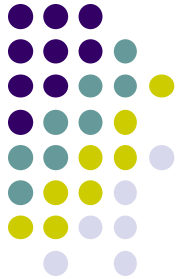
- Usando `receive()`

```
public void action() {
   ACLMessage msg = myAgent.receive();
   if (msg != null) {
       // Procesa el mensaje recibido
       ...
   }
   else {
       block();
   }
}
```

Implementar el ejercicio anterior utilizando receive().

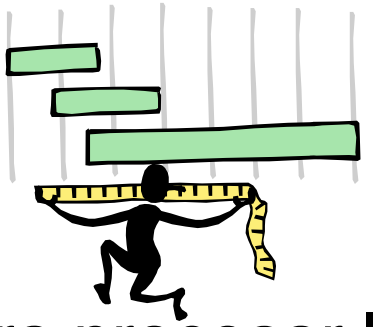
- Pattern sugerido para recibir mensajes dentro de un comportamiento

Conversaciones concurrentes



- **conversation-id**: identifica la conversación unívocamente.
- **reply-with**: identificador de la respuesta próxima.
- **in-replay-to**: reply-with del mensaje anterior.

Seleccionar mensajes específicos



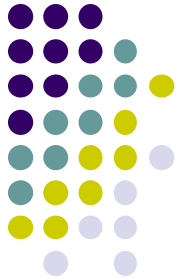
- Se pueden especificar template para procesar los mensajes
  - clase MessageTemplate
    - AND, OR, NOT, MatchContent(string), MatchSender(AID), MatchConversationID(string), MatchPerformative(int)
    - Se pasa como argumento de `receive(MessageTemplate)`

```
mt =
   MessageTemplate.and(MessageTemplate.MatchConversationId("123
   "),MessageTemplate.MatchInReplyTo(req.getReplyWith()));
```

ACLMessage posee constantes por cada Performativa

En el ejercicio anterior responder sólo a los mensajes de tipo REQUEST

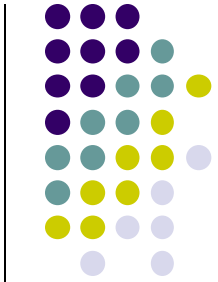
Seleccionar mensajes específicos



- Definir nuevos template

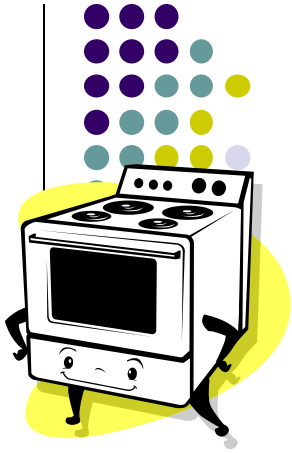
```
class myMatchExpression implements MessageTemplate.MatchExpression
{
   List senders;
   myMatchExpression(List l){
      senders = l;
   }
   public boolean match(ACLMessage msg){
      AID sender = msg.getSender();
      String name = sender.getName();
      Iterator it_temp = senders.iterator();
      boolean out = false;
      while(it_temp.hasNext() && !out){
         String tmp = ((AID)it_temp.next()).getName();
         if(tmp.equalsIgnoreCase(name))
          out = true;
      }
      return out;
   }
}
```

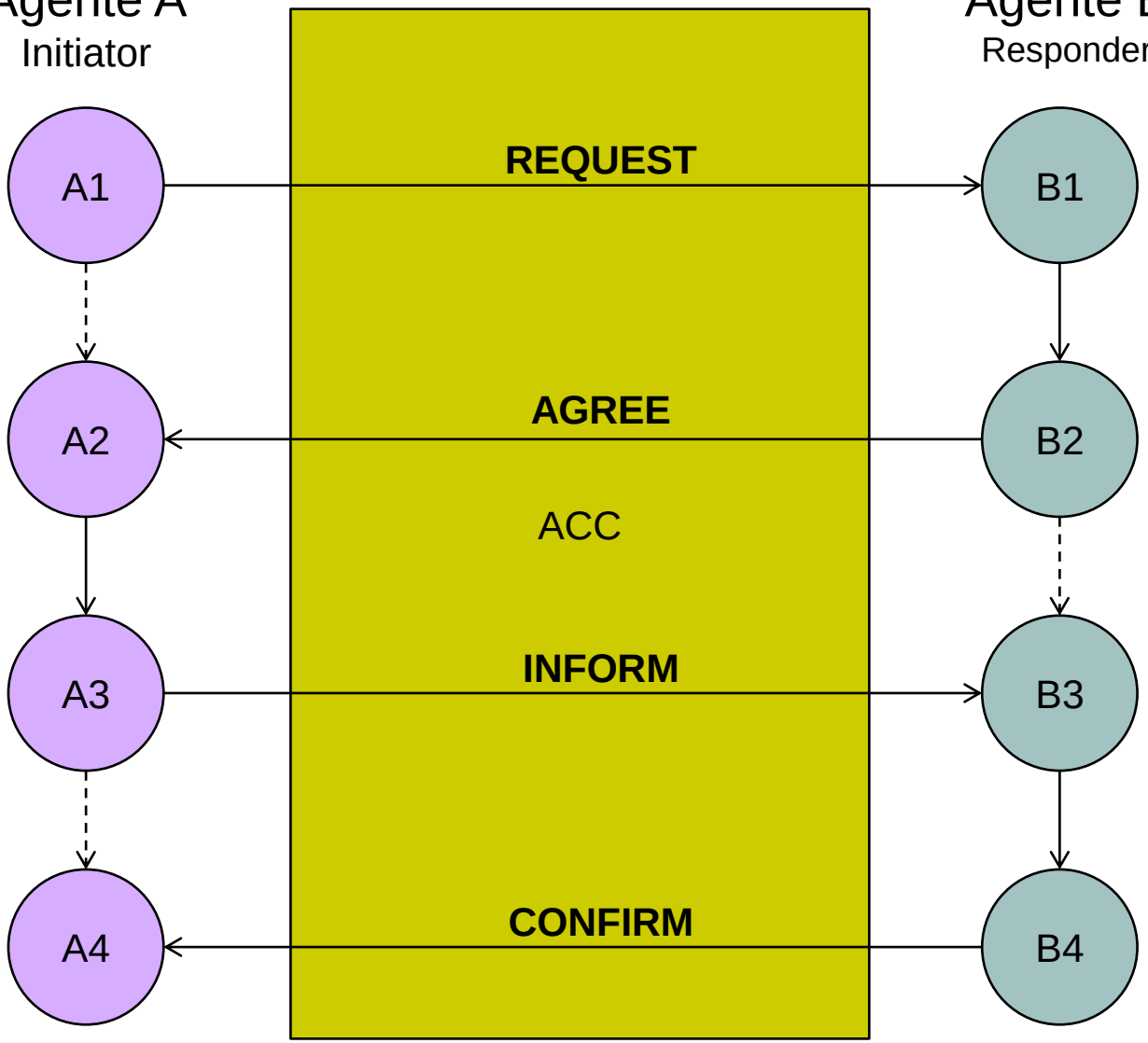
Crear una respuesta

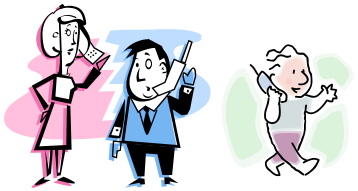


- ACLMessage.createReply()
  - crea un nuevo mensaje ACL en respuesta a este mensaje. Determina: `receiver`, `language`, `ontology`, `protocol`, `conversation-id`, `in-reply-to`, `reply-with`. El programador debe setear `communicative-act` y `content`.

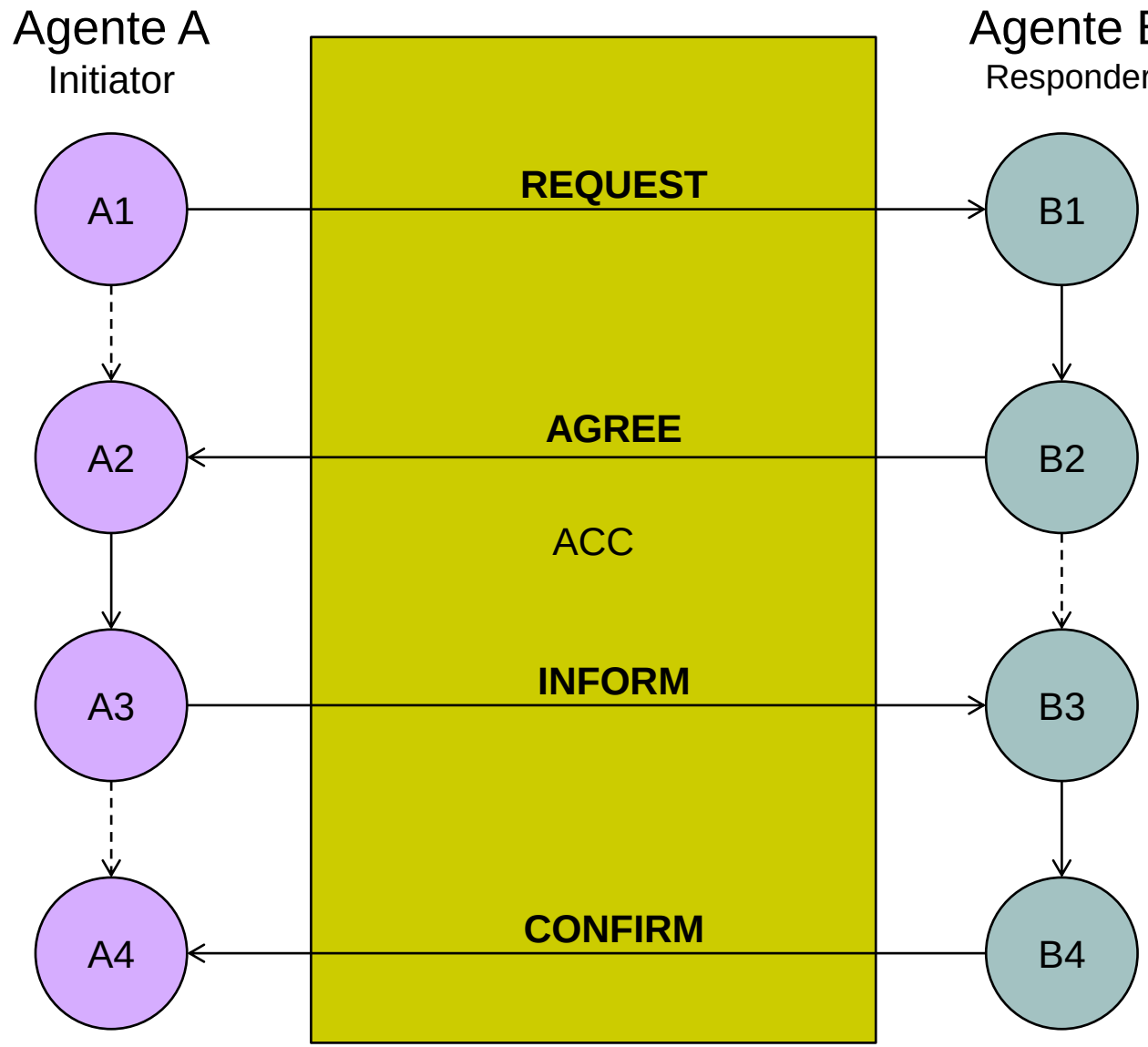
Ejemplo



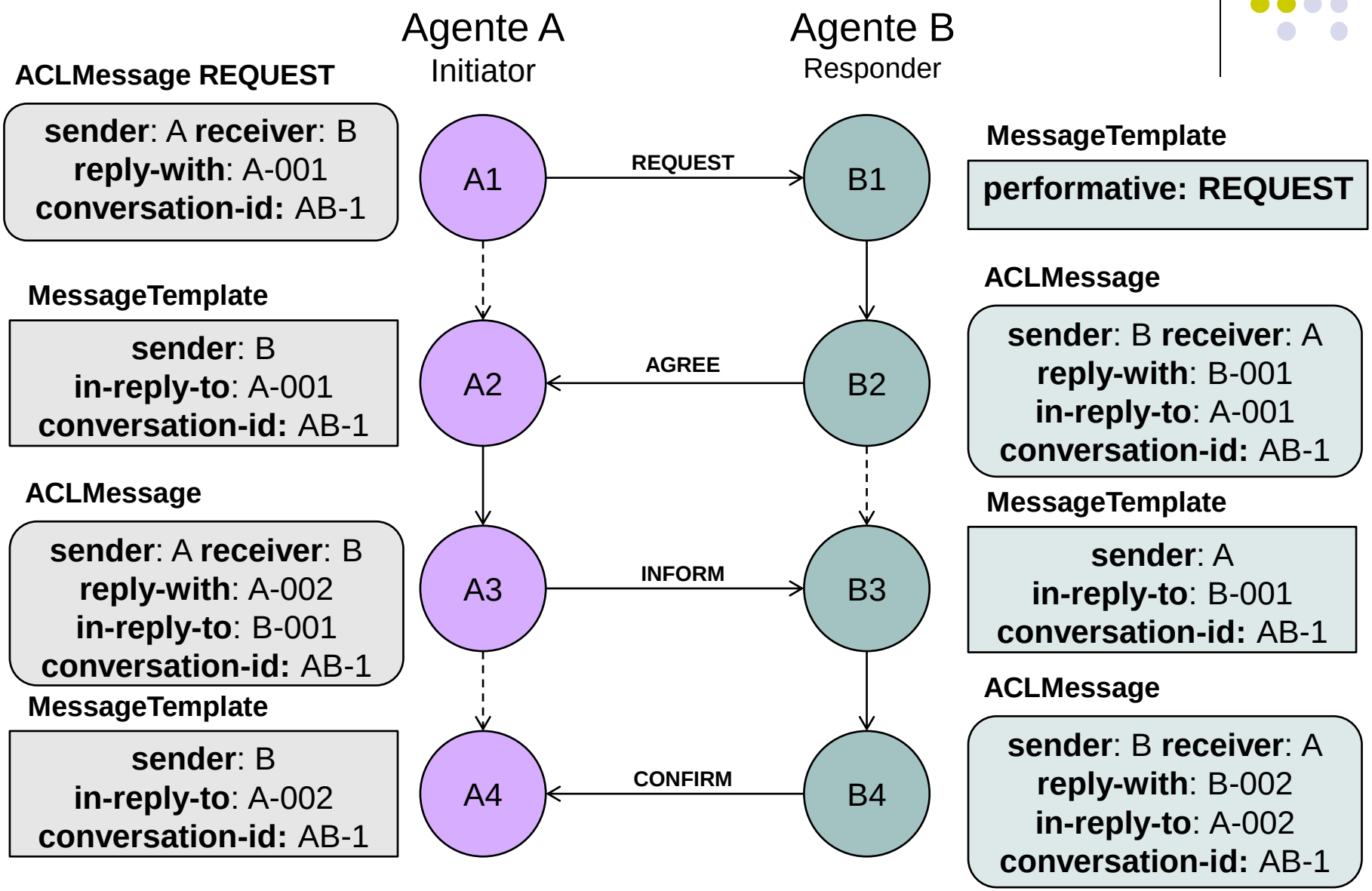

Agente A
Initiator
Agente B
Responder
A1
A2
A3
A4
B1
B2
B3
B4
REQUEST
AGREE
ACC
INFORM
CONFIRM



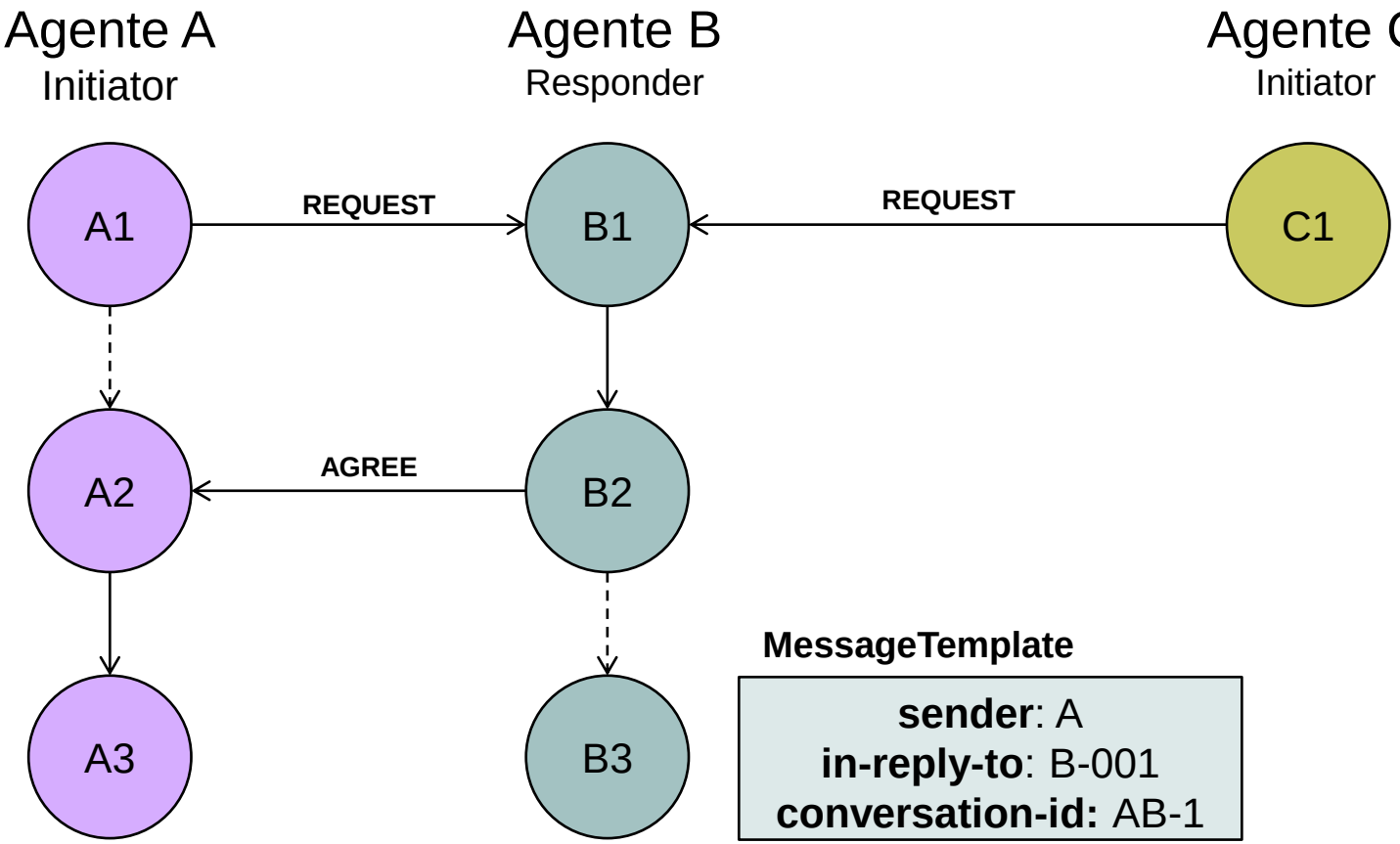
Ejemplo


Agente A
Initiator
Agente B
Responder
A1
A2
A3
A4
B1
B2
B3
B4
REQUEST
AGREE
ACC
INFORM
CONFIRM

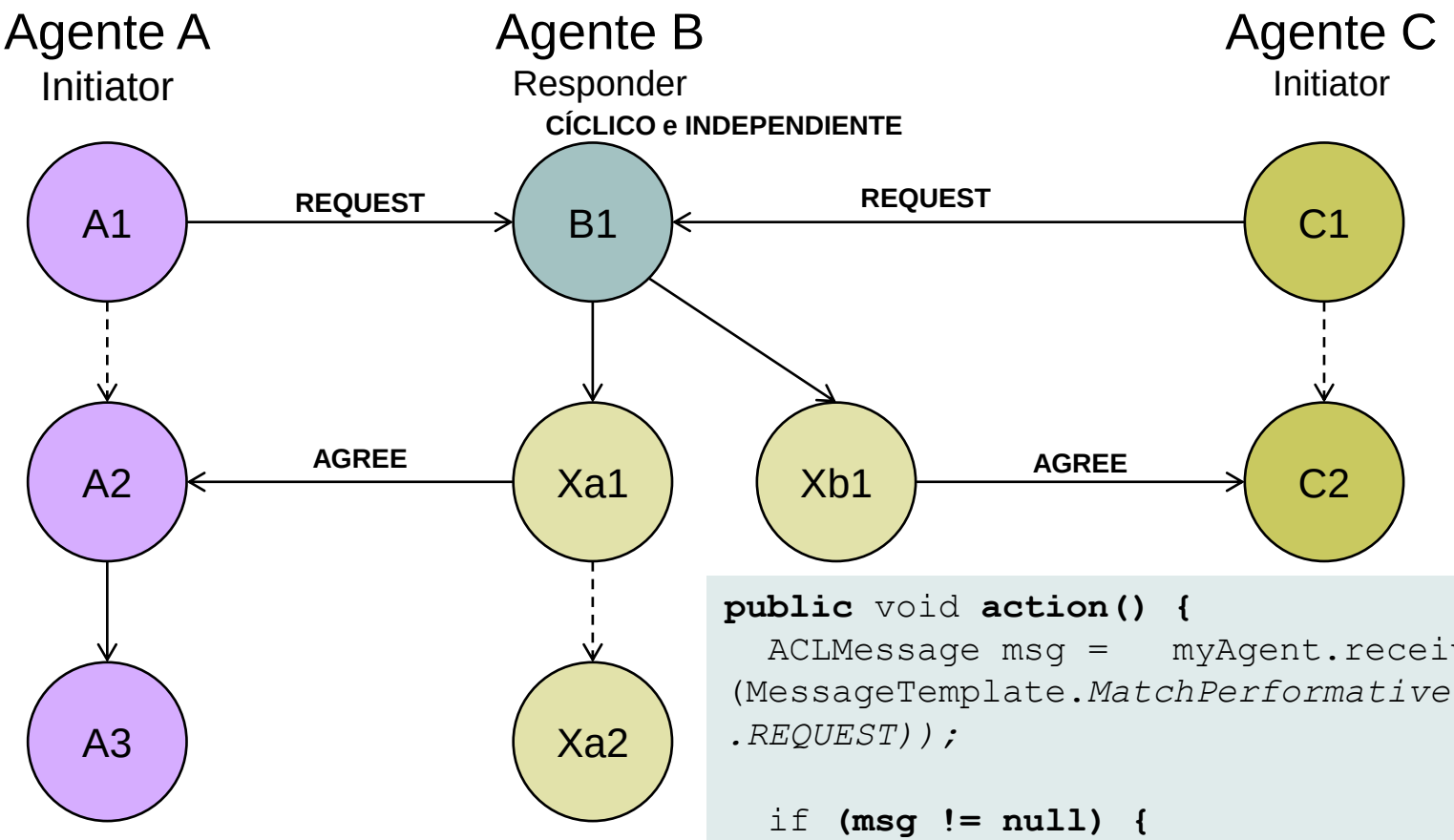
Ejemplo


Agente A
Initiator
Agente B
Responder
ACLMessage REQUEST
sender: A receiver: B
reply-with: A-001
conversation-id: AB-1
A1
REQUEST
B1
MessageTemplate
performative: REQUEST
MessageTemplate
sender: B
in-reply-to: A-001
conversation-id: AB-1
A2
AGREE
B2
ACLMessage
sender: B receiver: A
reply-with: B-001
in-reply-to: A-001
conversation-id: AB-1
ACLMessage
sender: A receiver: B
reply-with: A-002
in-reply-to: B-001
conversation-id: AB-1
A3
INFORM
B3
MessageTemplate
sender: A
in-reply-to: B-001
conversation-id: AB-1
MessageTemplate
sender: B
in-reply-to: A-002
conversation-id: AB-1
A4
CONFIRM
B4
ACLMessage
sender: B receiver: A
reply-with: B-002
in-reply-to: A-002
conversation-id: AB-1

Ejemplo

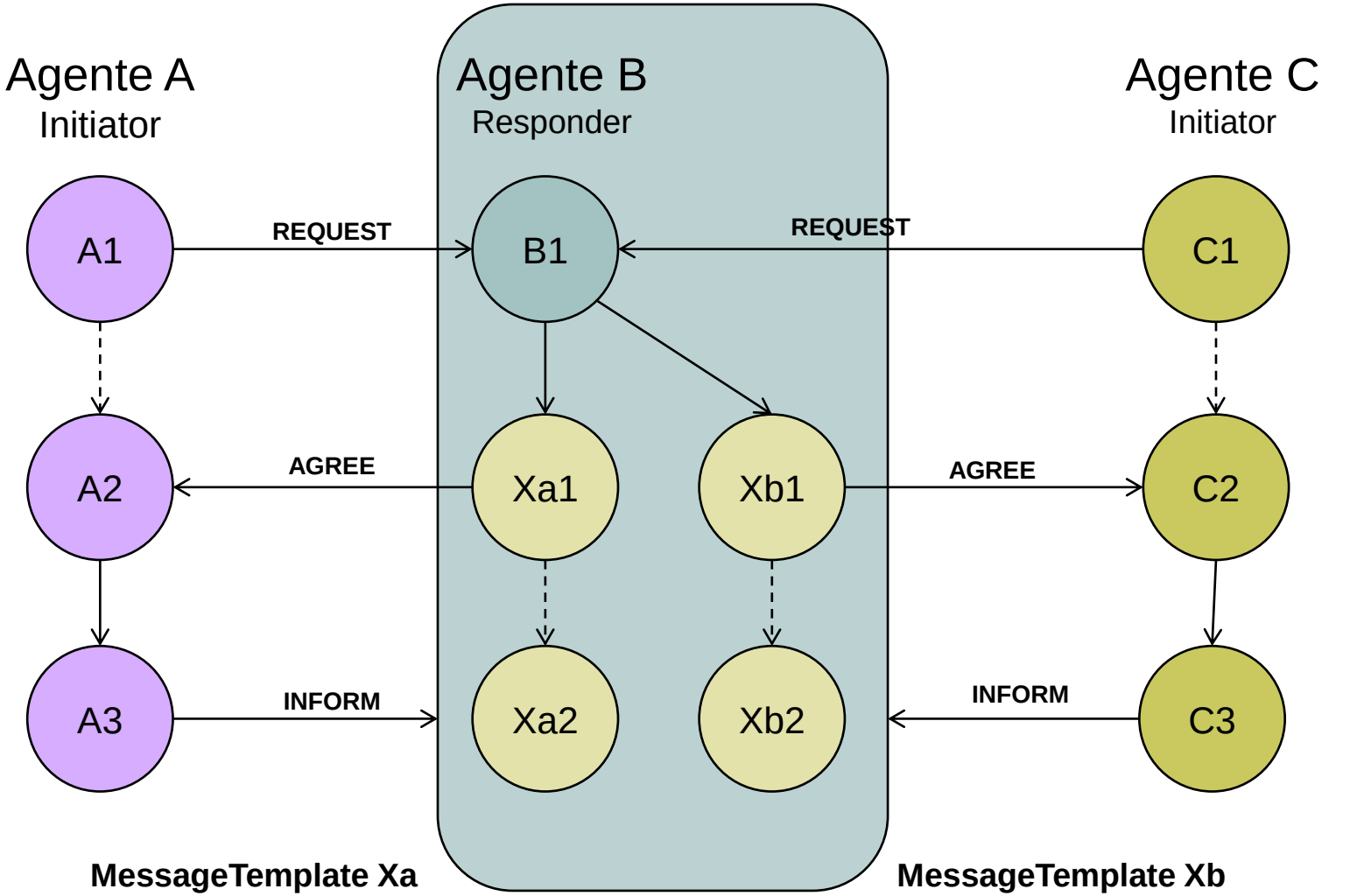

Agente A
Initiator
Agente B
Responder
Agente C
Initiator
A1
REQUEST
B1
REQUEST
C1
A2
AGREE
B2
A3
B3
MessageTemplate
sender: A
in-reply-to: B-001
conversation-id: AB-1

Ejemplo




```
public void action() {
  ACLMessage msg =   myAgent.receive
(MessageTemplate.MatchPerformative(ACLMessage
.REQUEST));

  if (msg != null) {
    myAgent.addBehaviour(new
                         XBehaviour(msg));
  }
  else
    block();
}
```

Ejemplo



MessageTemplate Xa

sender: A
in-reply-to: B-001
conversation-id: AB-1

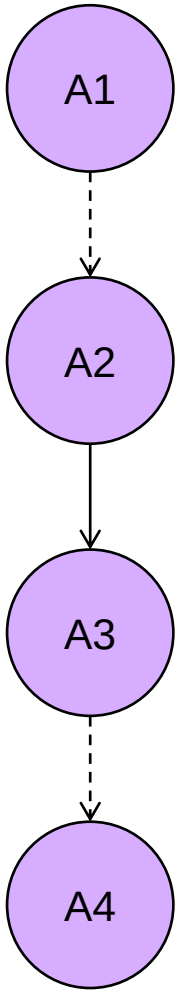
MessageTemplate Xb

sender: C
in-reply-to: B-201
conversation-id: CB-1

Ejemplo

Agente A

Initiator

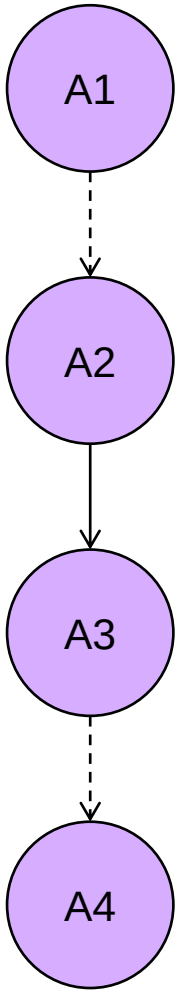



```
ACLMessage req = new ACLMessage(ACLMessage.REQUEST);
// Setear datos del mensaje
...
// Configura MessageTemaplate
mt =
MessageTemplate.and(MessageTemplate.MatchConversationId(req.g
etConversationId()),
MessageTemplate.MatchInReplyTo(req.getReplyWith()));

myAgent.send(req);
```

Ejemplo

Agente A

Initiator

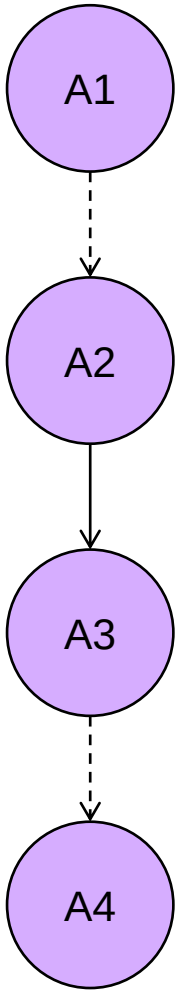



```
ACLMessage msg = myAgent.receive(mt);
if (msg != null) {
   // Pasa el control al siguiente comportamiento
}
else {
   block();
}
```

Ejemplo

Agente A

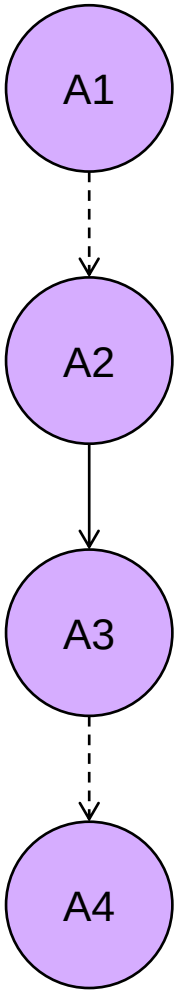
Initiator




```
// Procesa el mensaje recibido
...
// Prepara la respuesta
reply = msg.createReply();
mt =
MessageTemplate.and(MessageTemplate.MatchConversationId(reply
.getConversationId()),
MessageTemplate.MatchInReplyTo(reply.getReplyWith()));

myAgent.send(reply);
```

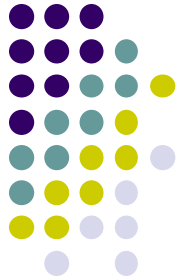
Ejemplo

Agente A
Initiator



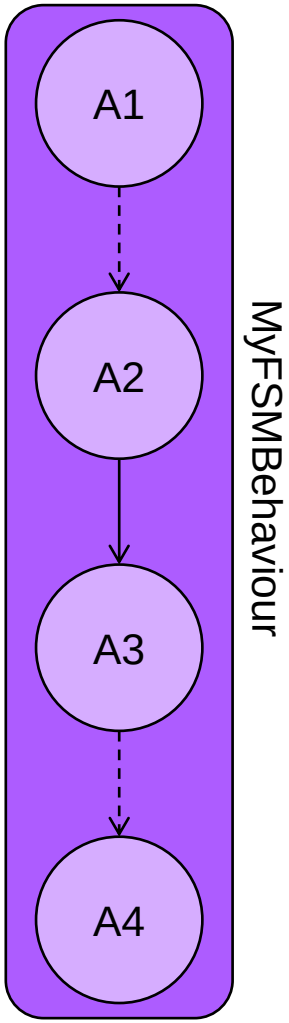

```
ACLMessage conf = myAgent.receive(mt);
if (conf != null) {
  // Procesa la confirmación y finaliza
}
else
   block();
```

Ejemplo



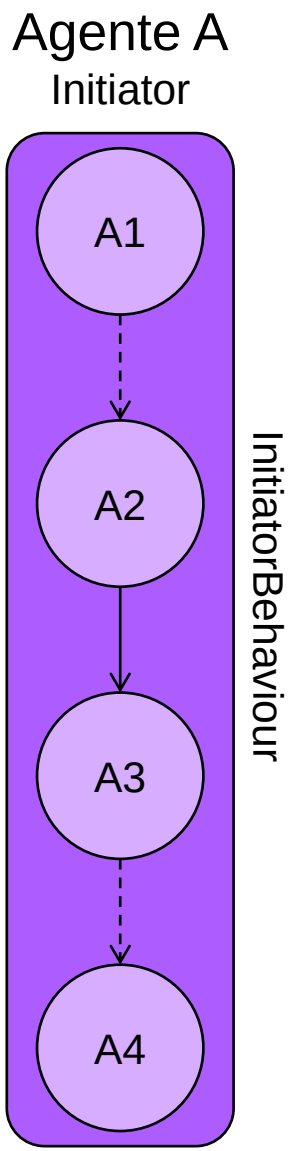
Agente A

Initiator




```
public MyFSMBehaviour() {
  DataStore ds = new DataStore();

  A1Behaviour a1 = new A1Behaviour();
  a1.setDataStore(ds);
  this.registerFirstState(a1,"a1");
  ...

  this.registerDefaultTransition("a1","a2");
  ...

}
```

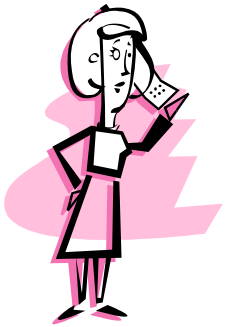
Ejemplo

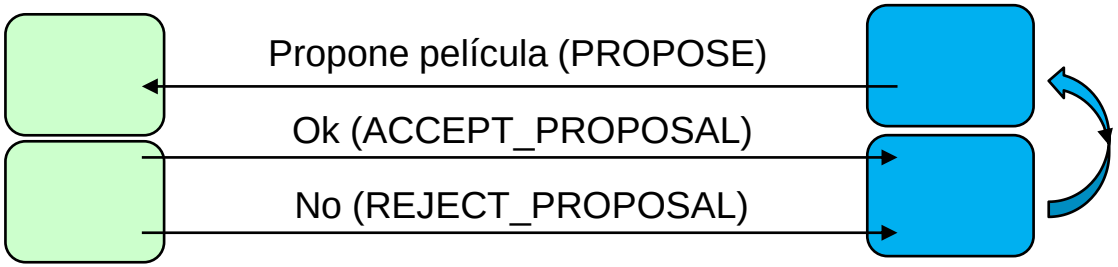



```
public InitiatorBehaviour(){
  step = 0;
}

public void action() {
  switch (step) {
  case 0:
    A1
    step = 1;
    break;
  case 1:
    A2
    step2
    break;

public boolean done() {
  return step == 4;
}
```

```
case 2:
    A3
    step = 3;
    break;
  case 3:
    A4
    step = 4;
    break;
  default:
    break;
  }
}
```

Implementando conversaciones





Agente RESPONDER
(procesar los pedidos concurrentemente)

Agente INITIATOR
(Uno o varios)

Implementar ambos agentes y testear con un RESPONDER y varios INITIATOR
(Trabajo de CURSADA)