# Game Oriented Multi Agent System, based on Jade

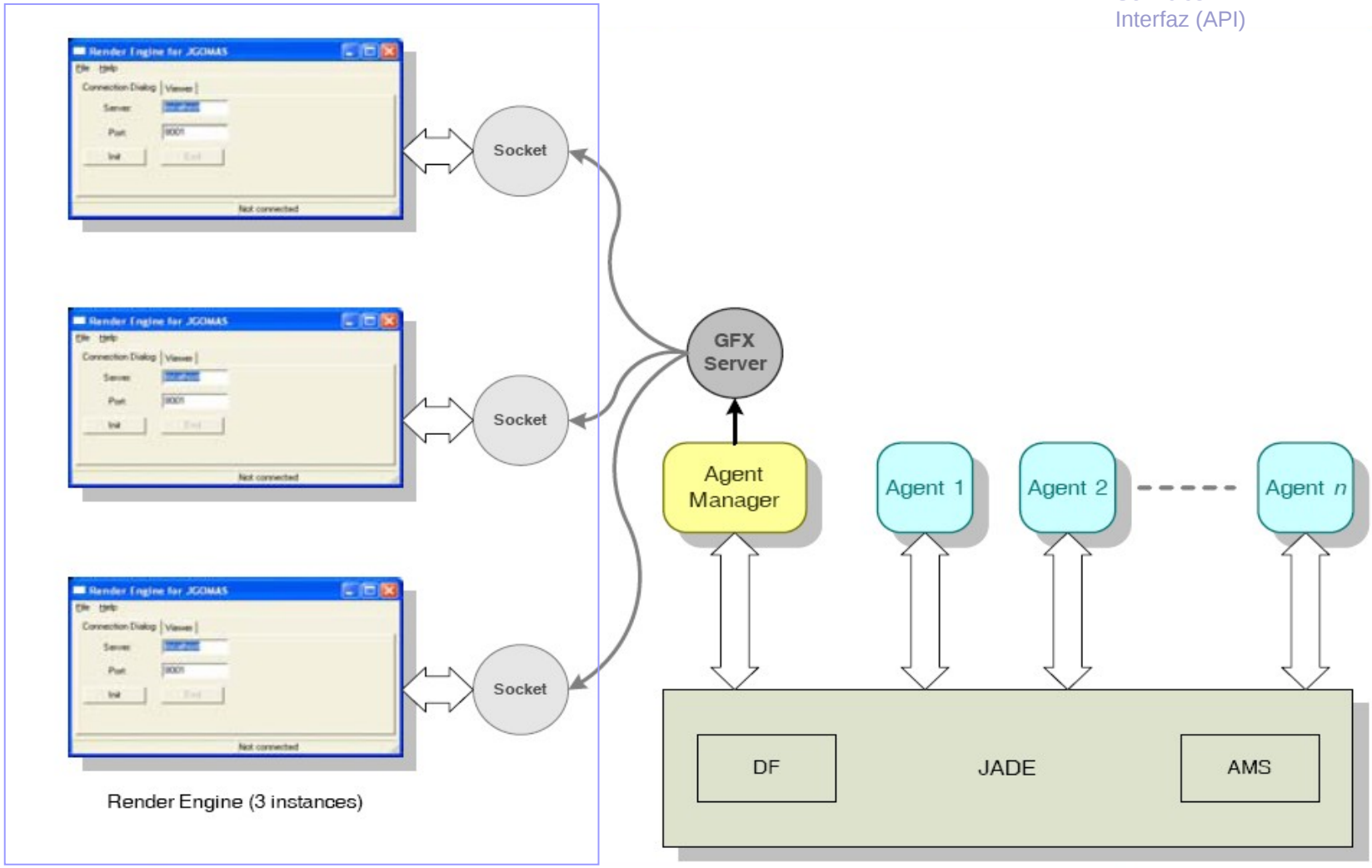
**Práctica TAIA , FISS**

*Toni Barella, DSIC – UPV*
***tbarella@dsic.upv.es***
***http://www.dsic.upv.es/users/ia/sma/tools/jgomas/***

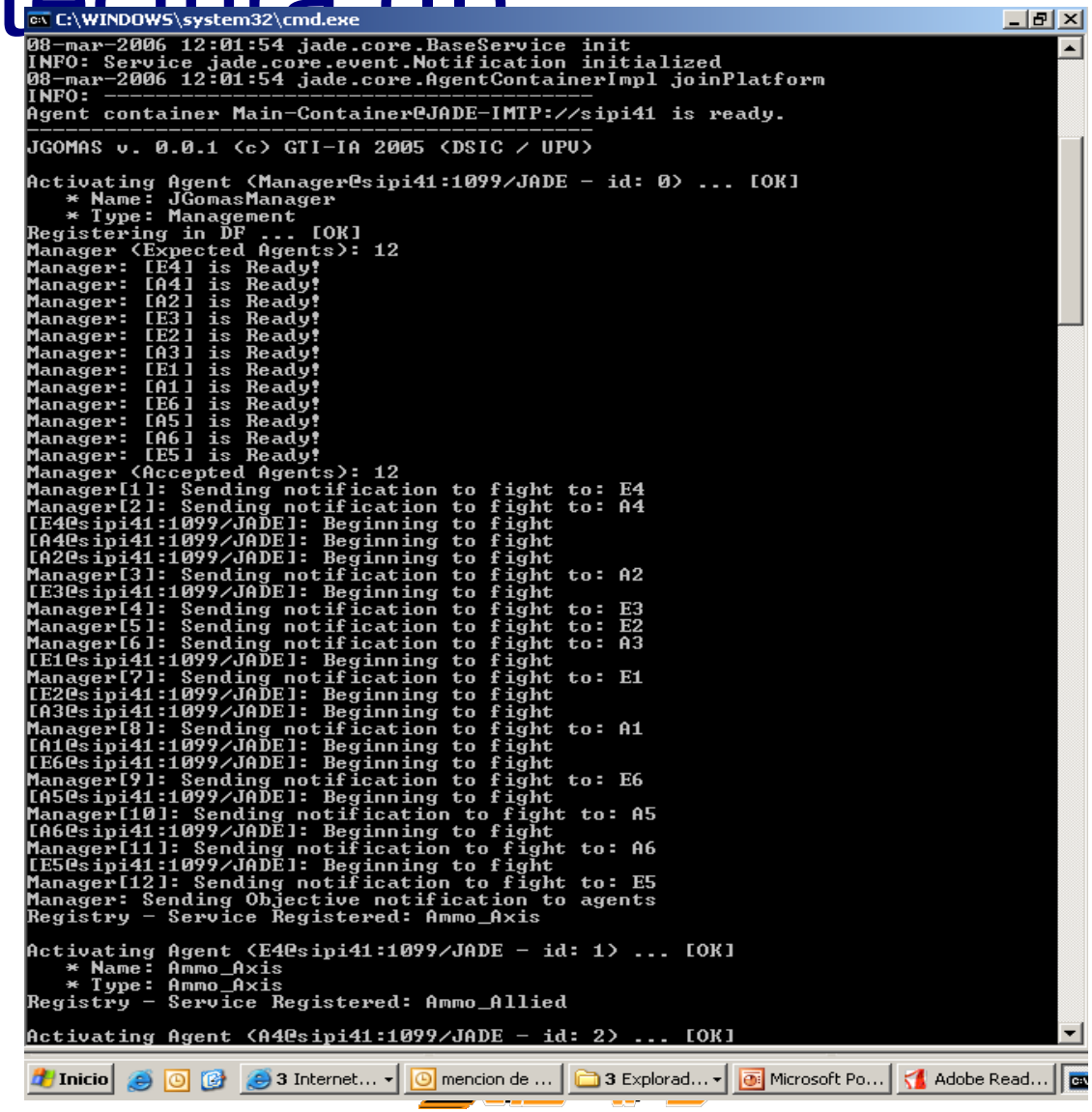
# Especificación JGOMAS

- **Arquitectura**
- Taxonomía de Agentes
- Tareas
- Bucle de Ejecución
- Comunicaciones
- Servicios
- Interfaz (API)

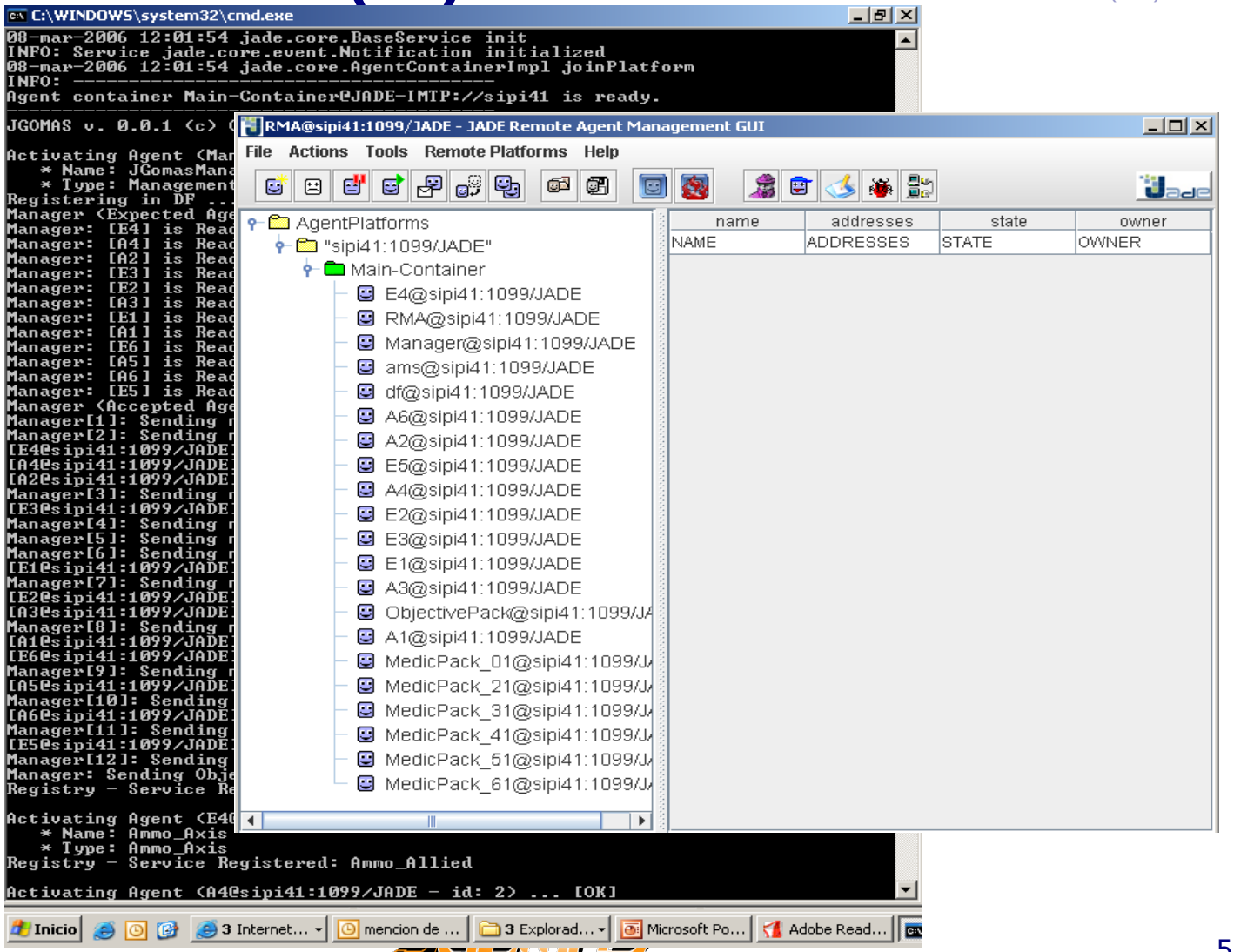
# Arquitectura (I)



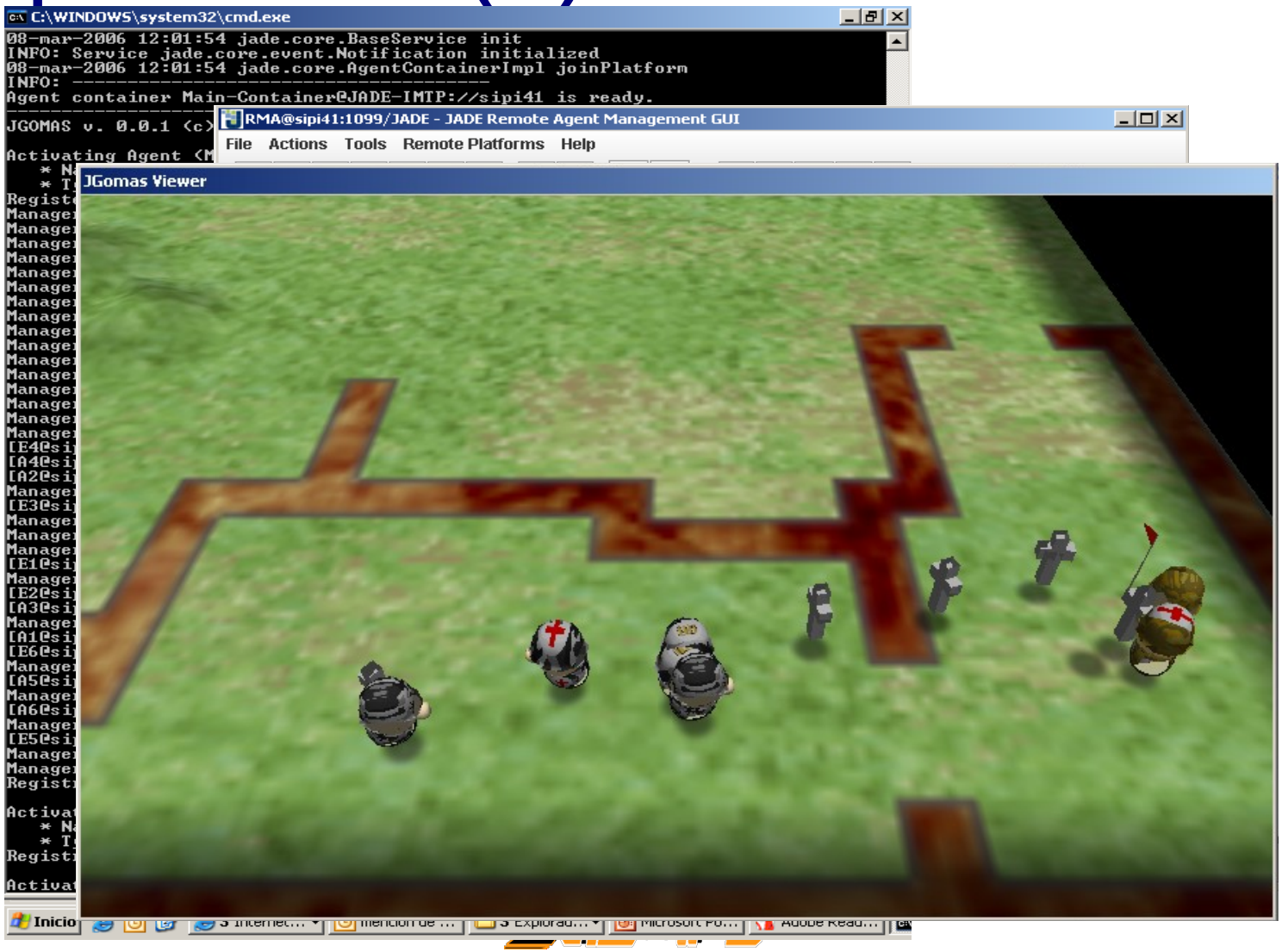
# Arquitectura (II)



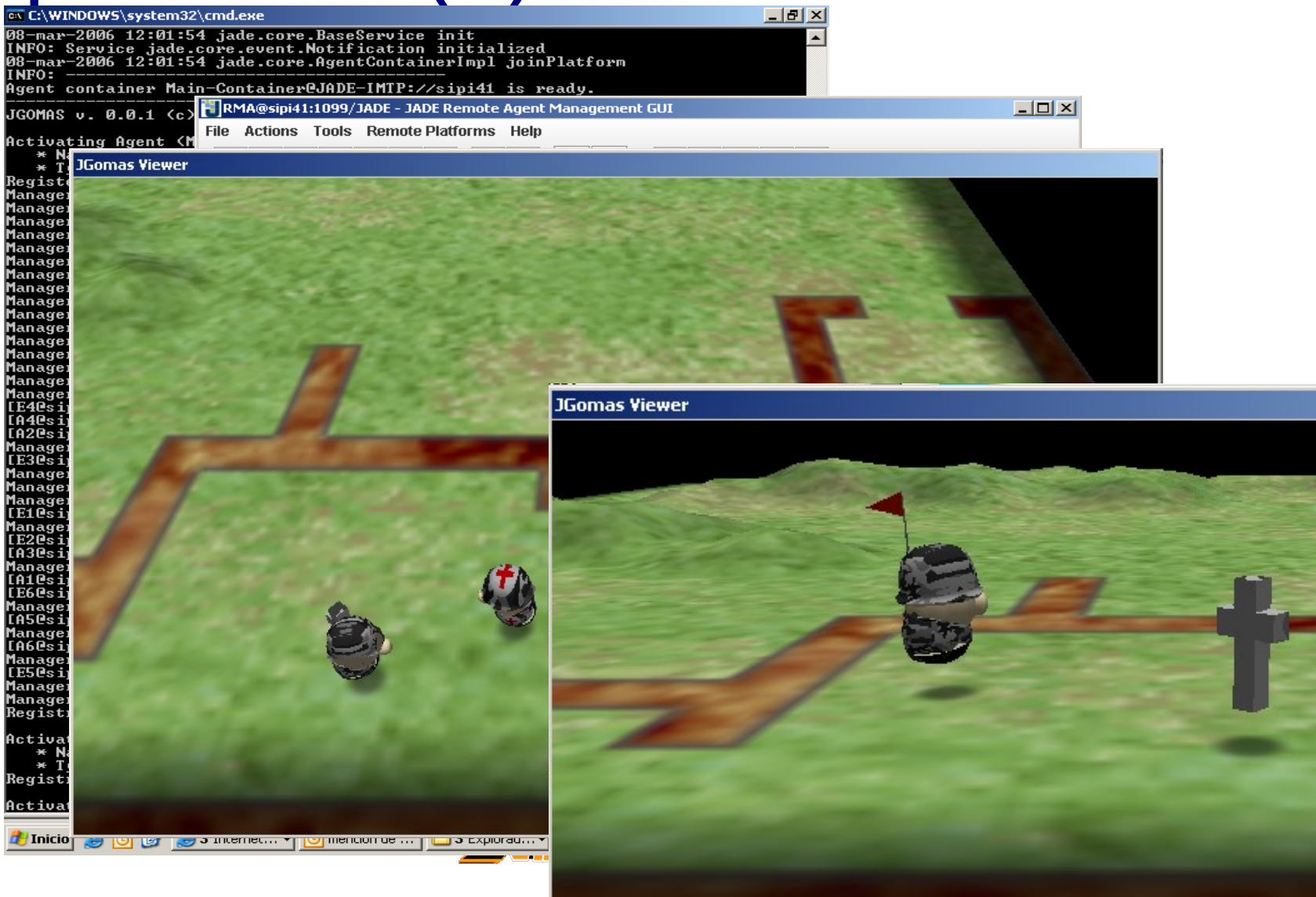
# Arquitectura (II)



# Arquitectura (II)



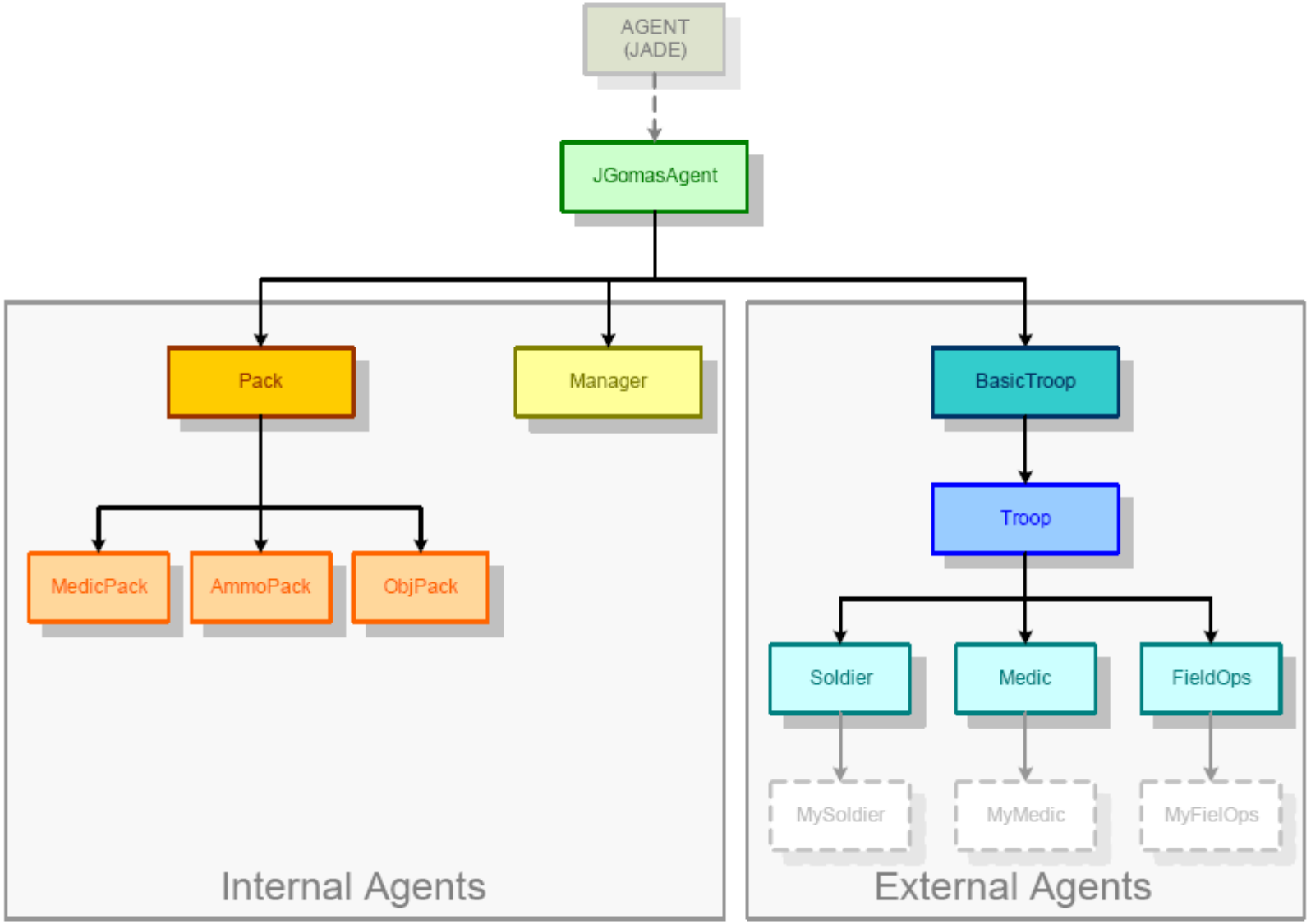
# Arquitectura (II)



# Especificación

- Arquitectura
- **Taxonomía de Agentes**
- Tareas
- Bucle de Ejecución
- Comunicaciones
- Servicios
- Interfaz (API)

# Taxonomía de Agentes



# Manager

- Controla el estado del juego:
  - Mantiene la información de todos los objetos y la envía al visualizador
- Gestión de la lógica del juego
  - Gestión del ciclo de vida
  - Coordinación y gestión de los servicios de los agentes
  - Estado actual del juego
  - Atención de peticiones de los agentes que tienen que ver con el entorno (disparar, mirar,…)
  - Estadísticas

- Hay definidos **tres tipos de roles**:
  - Soldier: acude a dar apoyo -- CallForBackup CFB
  - Medic: acude a curar -- CallForMedic CFM
  - FieldOps: acude a dar munición -- CallForAmmo CFA

- Un age
  toda la

- Cada r
  ofrece e
  un com e
  mejora

- es.upv.dsic.gti_ia.jgomas.CJGomasAgent
  - es.upv.dsic.gti_ia.jgomas.CBasicTroop
    - es.upv.dsic.gti_ia.jgomas.CTroop
      - es.upv.dsic.gti_ia.jgomas.CFieldOps
      - es.upv.dsic.gti_ia.jgomas.CMedic
      - es.upv.dsic.gti_ia.jgomas.CSoldier
  - es.upv.dsic.gti_ia.jgomas.CManager
  - es.upv.dsic.gti_ia.jgomas.CPack
    - es.upv.dsic.gti_ia.jgomas.CAmmoPack
    - es.upv.dsic.gti_ia.jgomas.CMedicPack
    - es.upv.dsic.gti_ia.jgomas.CObjPack

- Hay definidos **tres tipos de roles**:
  - Soldier: acude a dar apoyo -- CallForBackup CFB
  - Medic: acude a curar -- CallForMedic CFM
  - FieldOps: acude a dar munición -- CallForAmmo CFA

- Un agente asume un **único rol** durante toda la partida

- Cada rol tiene unas características y ofrece unos determinados servicios. Dispone de un comportamiento básico que el usuario puede mejorar.

# CJGomasAgent

```
package Classes;
public abstract class CJGomasAgent extends jade.core.Agent
{    //
     // Campos
     //
     protected int m_ID;
     protected java.util.List m_ServiceList;
     //
     // Constructores
     //
     public void CJGomasAgent() {. . . }
     //
     // Metodos
     //
     protected final void AddService(Classes.CService param1)    {. . . }
     protected void setup() {. . . }
     protected void takeDown() {. . . }
     private final void RegisterDF() {. . . }
     private final void DeregisterDF() {. . . }
}
```

# CBasicTroop

```
package Classes;
public abstract class CBasicTroop extends Classes.CJGomasAgent
{       //
        // Campos
        //
        protected java.util.Hashtable m_TaskList;
        protected Classes.CTask m_CurrentTask;
        protected int[] m_TaskPriority;
        private int m_iHealth , m_iPower , m_iAmmo;
        protected boolean m_bFighting , m_bEscaping;
        protected java.lang.String m_sMedicService , m_sAmmoService , m_sBackupService;
        //
        // Constructoras
        // Métodos
        //
        private final void Launch_MedicAmmo_RequestBehaviour() { . . . }
        private final void Launch_FSM_Behaviour() { . . . }
        protected final int Move() { . . . }
        protected final int GetHealth() { . . . }
        private final void IncHealth(int param1) { . . . }
        protected final void AddServiceType(java.lang.String param1) { . . . }
        protected final void AddTask(int param1, jade.core.AID param2, java.lang.String param3) { . . . }
        protected final void Look(){ . . . }
        protected final boolean Shot(int param1) { . . . }
        protected final boolean HaveAgentToShot() { . . . }
        protected void CallForMedic() { . . . }
        protected void CallForAmmo() { . . . }
        protected void CallForBackup() { . . . }
}
```

# CBasicTroop

```
package Classes;
public abstract class CBasicTroop extends Classes.CJGomasAgent
{       //
        // Campos
        //
        protected java.util.Hashtable m_TaskList;
        protected Classes.CTask m_CurrentTask;
```

```
/*
 * Request for medicine.
 * This method sends a <b> FIPA REQUEST </b> message to all agents who offers
 * the <tt> m_sMedicService </tt> service.
 * The content of message is: <tt> ( x , y , z ) ( health ) </tt>.
 * Variable <tt> m_iMedicsCount </tt> is updated.

protected void CallForMedic() {…
```

```
        protected final boolean HaveAgentToShot() { . . . }
        protected void CallForMedic() { . . . }
        protected void CallForAmmo() { . . . }
        protected void CallForBackup() { . . . }
}
```

# CSoldier

```
package Classes;
public class CSoldier extends Classes.CTroop
{    //
      // Campos
      //
      // Constructoras
      //
      public void CSoldier() { . . . }
      //
      // Métodos
      //
      protected void setup(){ . . . }
      protected void SetUpPriorities(){ . . . }
      private final void Launch_CFB_ResponderBehaviour(){ . . . }
      protected boolean checkBackupAction(java.lang.String param1) { . . . }
      private final boolean performBackupAction() { . . . }
      }
}
```

CS

```
/**
* Definition of priorities for each kind of task.
* This method can be implemented in CTroop's derived classes to define the task's

* priorities in agreement to the role of the new class. Priorities must be defined in the

* array <tt> m_TaskPriority</tt>.
*/

protected void SetUpPriorities() {
        m_TaskPriority[CTask.TASK_NONE] = 0;
        m_TaskPriority[CTask.TASK_GIVE_MEDICPAKS] = 2000;
        m_TaskPriority[CTask.TASK_GIVE_AMMOPACKS] = 0;
        m_TaskPriority[CTask.TASK_GIVE_BACKUP] = 0;
        m_TaskPriority[CTask.TASK_GET_OBJECTIVE] = 1000;
        m_TaskPriority[CTask.TASK_ATTACK] = 1000;
        m_TaskPriority[CTask.TASK_RUN_AWAY] = 1500;
        m_TaskPriority[CTask.TASK_GOTO_POSITION] = 750;
        m_TaskPriority[CTask.TASK_PATROLLING] = 500;
        m_TaskPriority[CTask.TASK_WALKING_PATH] = 750;
}
```

CS

```
/**
* Definition of priorities for each kind of task.
* This method can be implemented in CTroop's derived classes to define the task's

* priorities in agreement to the role of the new class. Priorities must be defined in the
```

```
////////////////////////////////////////////////////////////////////////////////////////
// Methods to overload inhereted from CMedic/CSoldier/CFieldOp class
//
////////////////////////////////////////////////////////////////////////////////////////
/**
 * Decides if agent accepts the CFM/CFA/CFB request
*
* This method is a decision function invoked when a CALL FOR MEDIC request has arrived.
* Parameter <tt> sContent</tt> is the content of message received in <tt> CFM</tt>
responder behaviour as
 * result of a <tt> CallForMedic</tt> request, so it must be: <tt> ( x , y , z ) ( health ) </tt>.
* By default, the return value is <tt> TRUE</tt>, so agents always accept all CFM requests.
*
 */
protected boolean checkMedicAction/checkAmmoAction/checkBackupAction
                                                        (String _sContent) {
        // We always go to help
        return ( true );
}
```

# CMedic

```
package Classes;
public class CMedic extends Classes.CTroop
{    //
     // Campos
     //
     protected static int m_iPacksDelivered;
     //
     // Constructoras
     //
     public void CMedic() {. . .}
     //
     // Metodos
     //
     protected void setup() {. . .}
     protected void SetUpPriorities() {. . .}
     protected boolean checkMedicAction(java.lang.String param1) {. . .}
     private final boolean performMedicAction() {. . .}
     protected final int CreateMedicPack() {. . .}
}
```

# Especificación

- Arquitectura
- Taxonomía de Agentes
- **Tareas**
- Bucle de Ejecución
- Comunicaciones
- Servicios
- Interfaz (API)

# Tareas (I)

- Definición

```
class CTask {
   AID          m_AID
   int          m_iType;
   int          m_iPriority;
   Vector3D     m_Position;
   . . .
}
```

- `m_AID` es el identificador del agente que *provoca* la creación de la tarea

# Tareas (II)

```
class CTask {
        AID         m_AID
        int         m_iType;
        int         m_iPriority;
        Vector3D    m_Position;
        . . .
                }
```

- `m_iType` es el tipo de tarea
- Algunos de los tipos son:
  - TASK_GIVE_MEDICPAKS
  - TASK_GIVE_AMMOPACKS
  - TASK_GIVE_BACKUP
  - TASK_GET_OBJECTIVE
  - TASK_GOTO_POSITION
  - …

```
class CTask {
        AID         m_AID
        int         m_iType;
        int         m_iPriority;
        Vector3D    m_Position;
        . . .  }
```

# Tareas (III)

- `m_iPriority` indica la prioridad **actual**
- Se lanza siempre la tarea de prioridad más alta
- Es posible redefinir la prioridad de cada tipo de tarea:

```
protected void SetUpPriorities() {
…
m_TaskPriority[TASK_GIVE_MEDICPAK] = 2000;
m_TaskPriority[TASK_GIVE_AMMOPACS] = 0;
m_TaskPriority[TASK_GIVE_BACKUP] = 0;
m_TaskPriority[TASK_GET_OBJECTIVE] = 1000;
m_TaskPriority[TASK_GOTO_POSITION] = 750;
…
}
```

# Tareas (IV)

- Declaración:

```
AddTask(int    _tTypeOfTask,
        AID    _owner,
        String _sContent)

AddTask(int    _tTypeOfTask,
        AID    _owner,
        String _sContent,
        int    _iPriority)
```

- Ejemplo de uso:

```
String sNewPosition =
    " ( " + x + " , " + y + " , " + z + " ) ";
AddTask(TASK_GOTO_POSITION, getAID(), sNewPosition);
```

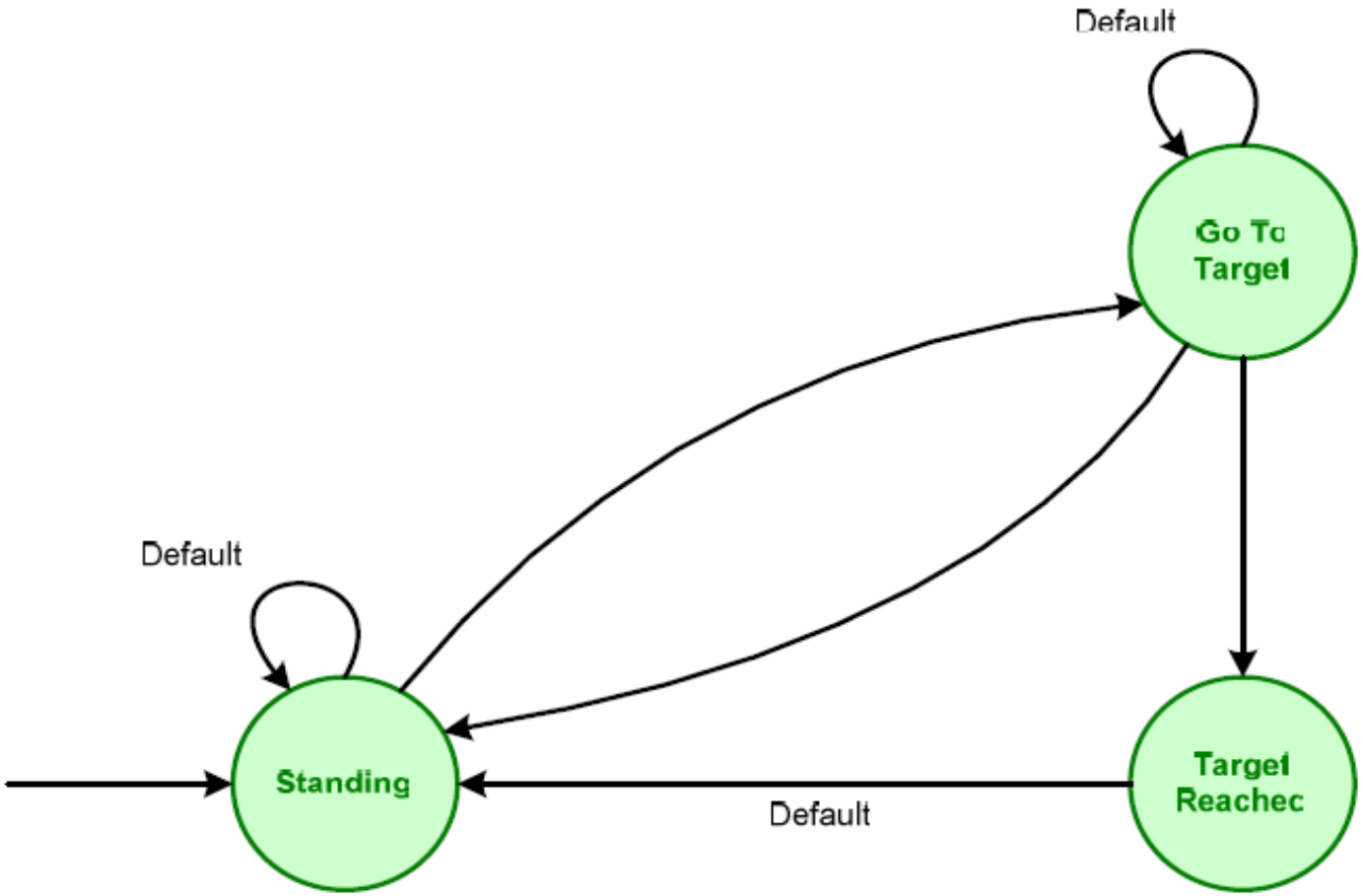
# Especificación

- Arquitectura
- Taxonomía de Agentes
- Tareas
- **Bucle de Ejecución**
- Comunicaciones
- Servicios
- Interfaz (API)

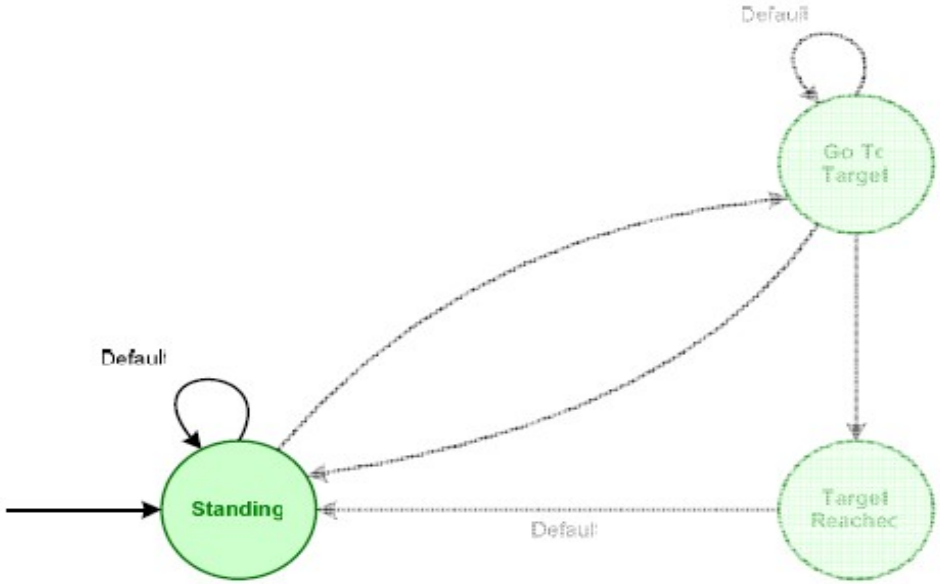
# Bucle de Ejecución (I)

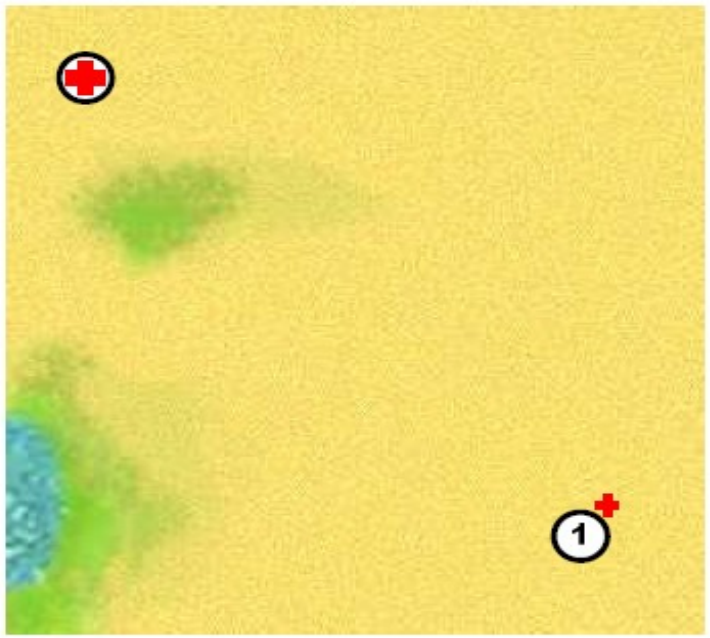
- Cada agente ejecuta una FSM:
  - STATE_STANDING
  - STATE_GOTO_TARGET
  - STATE_TARGET_REACHED
- FSM se utiliza para realizar tareas:
  - Inicio (Lanzamiento)
  - Desarrollo (Ejecución)
  - Final (Acción y Destrucción)
- Se lanza siempre la tarea de prioridad más alta

# Bucle de Ejecución (II)

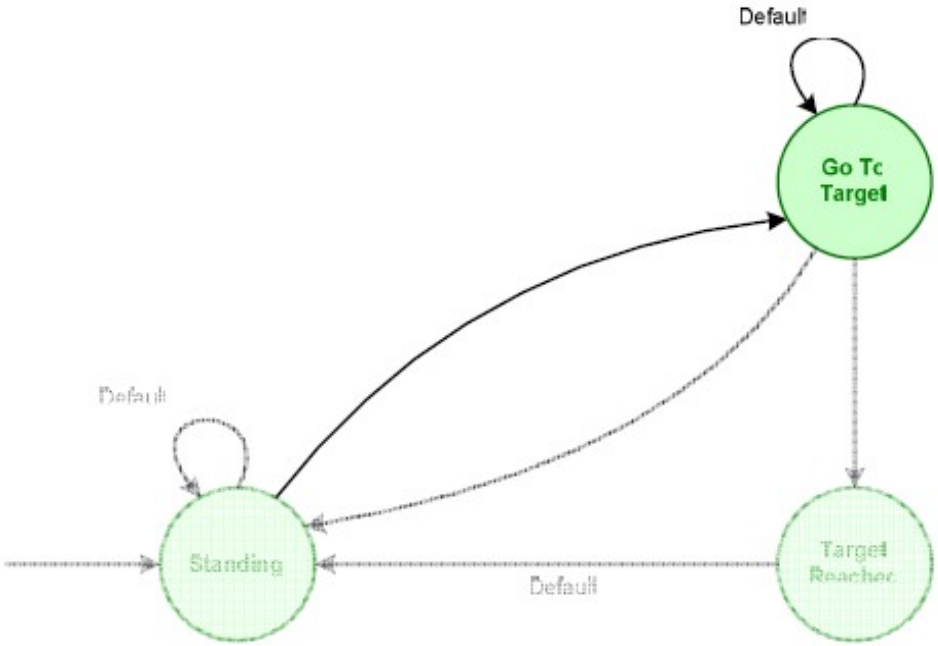


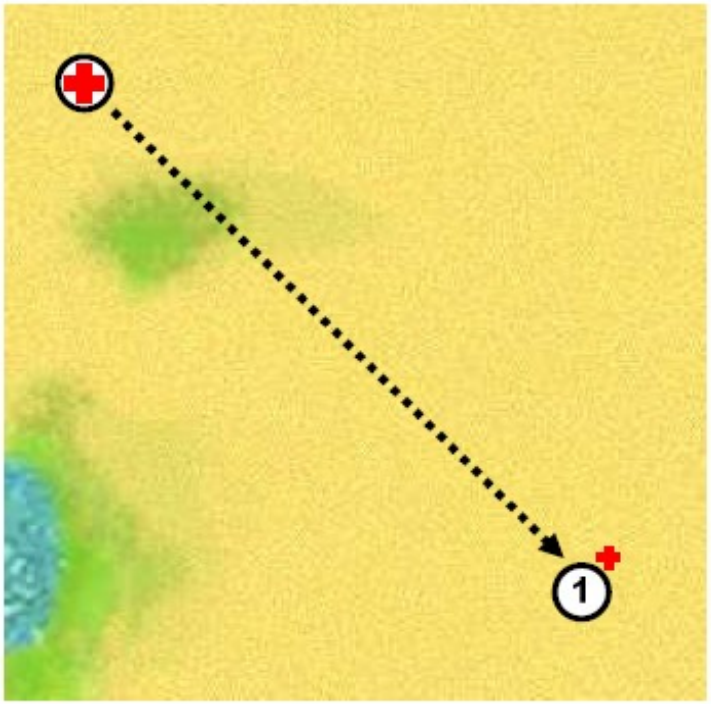
# Ejecución: Caso 1 (I)



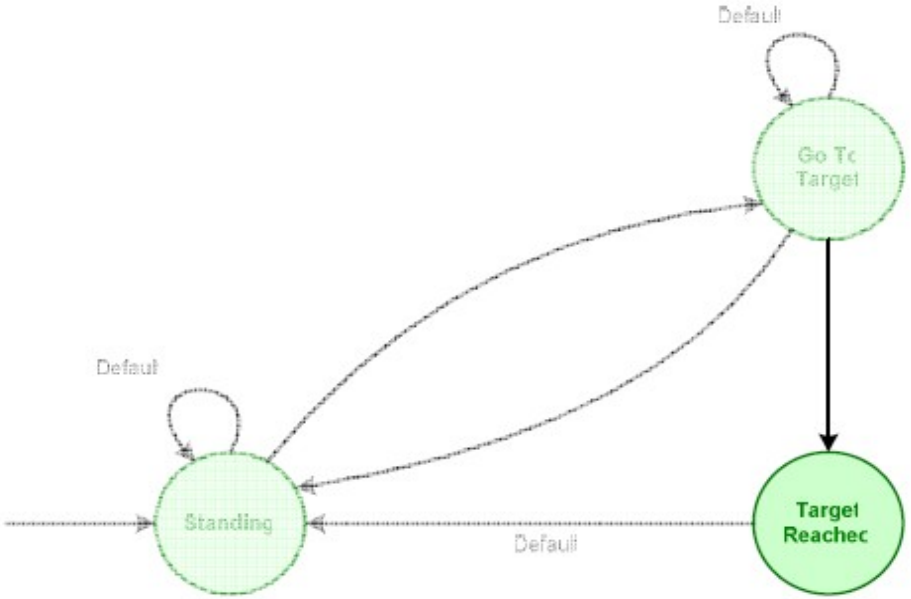



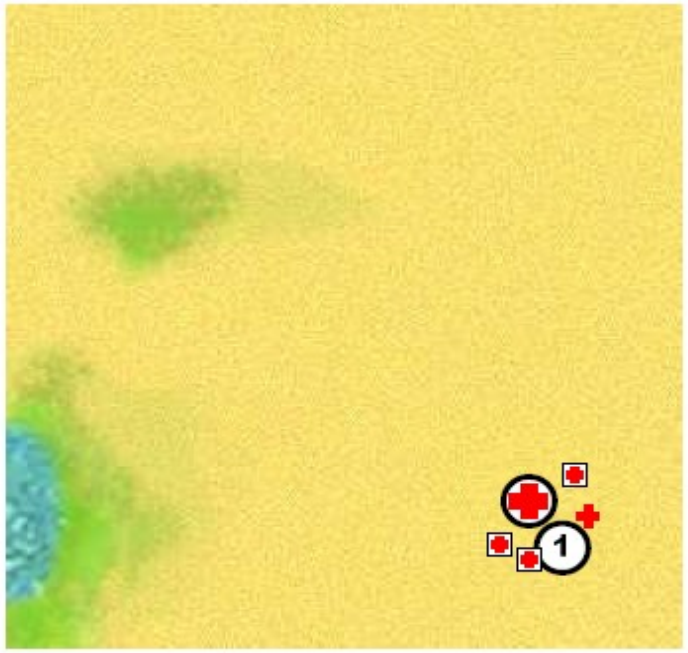
# Ejecución: Caso 1 (II)



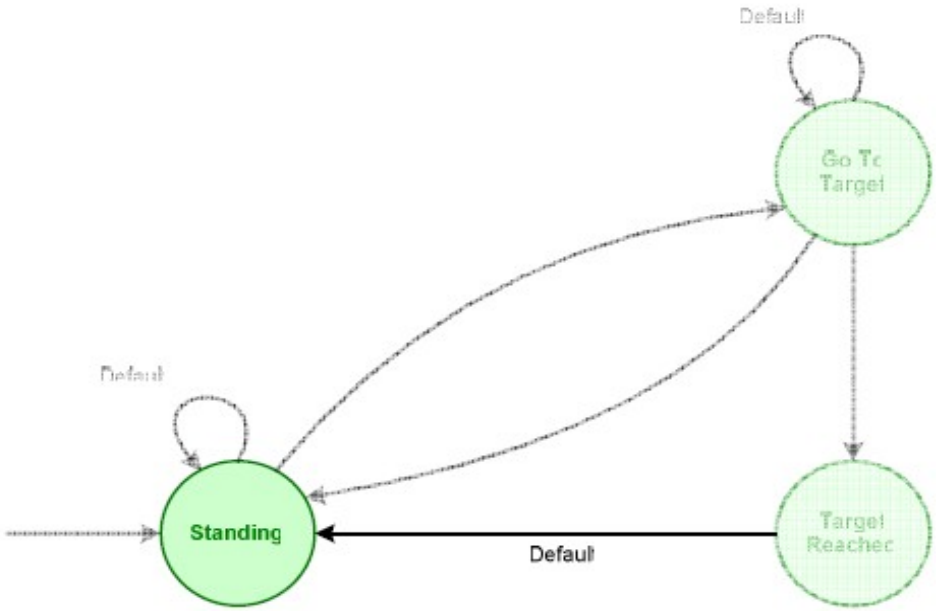



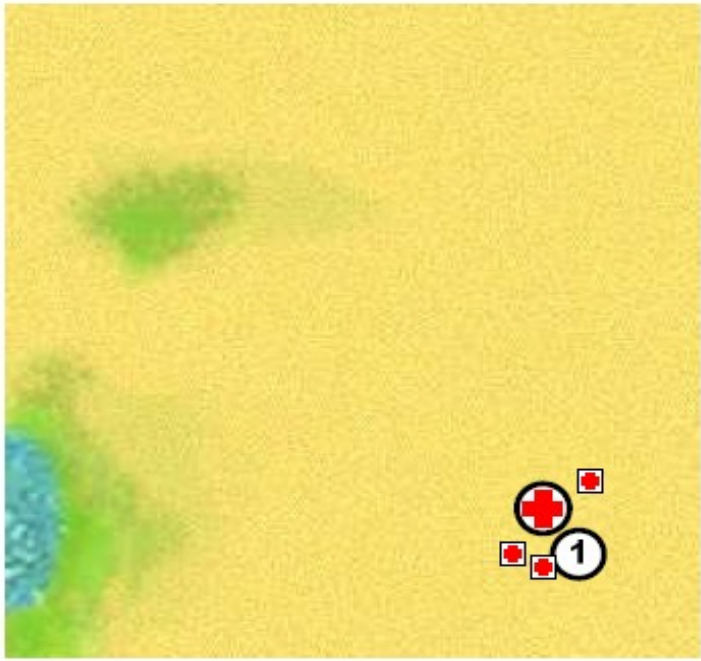
# Ejecución: Caso 1 (III)



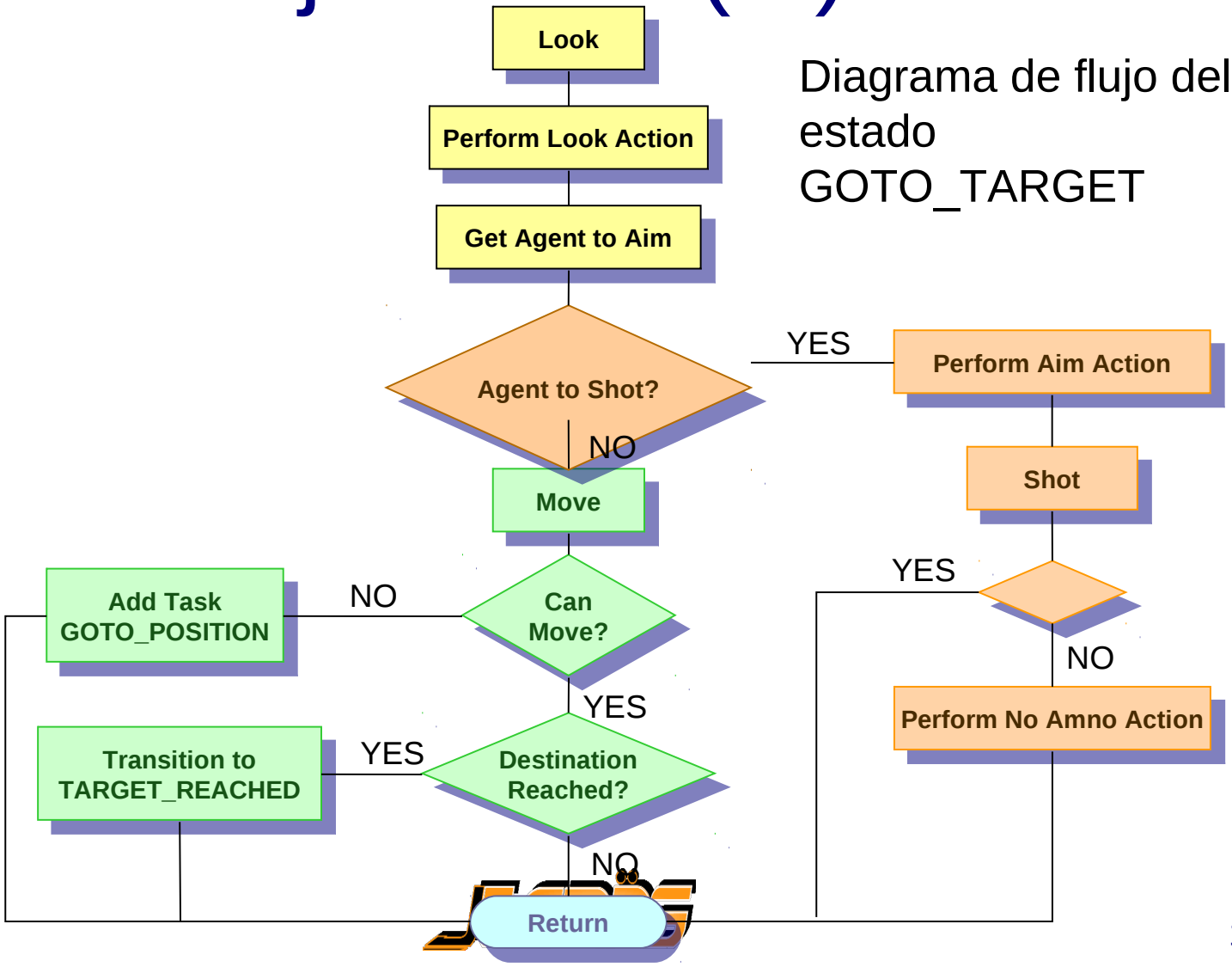
# Ejecución: Caso 1 (IV)



# Bucle de Ejecución (III)

Diagrama de flujo del estado GOTO_TARGET



# Especificación

- Arquitectura
- Taxonomía de Agentes
- Tareas
- Bucle de Ejecución
- **Comunicaciones**
- Servicios
- Interfaz (API)

# Comunicaciones

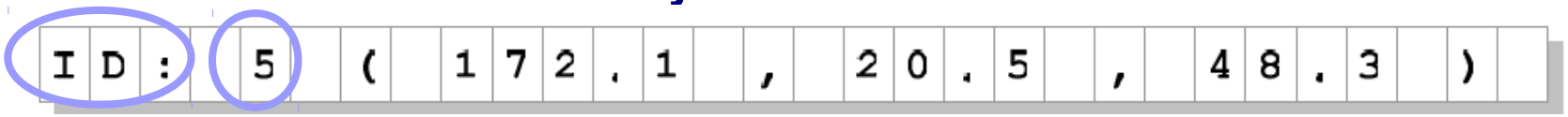
- Estándar FIPA
  - Partículas ilocutivas
- Formato de los mensajes
  - StringTokenizer
- Uso de templates y conversationID

# Comunicaciones (I)

- Estándar FIPA
  - Protocolos establecidos
    - Soportados por JADE
  - Partículas ilocutivas
    - INFORM
    - REQUEST
    - AGREE
    - REFUSE
    - CANCEL

# Comunicaciones (II)

- Formato de los mensajes




  - StringTokenizer

```
StringTokenizer tokens = new StringTokenizer(sContent);
tokens.nextToken(); // Get "ID:"
Integer id = Integer.parseInt(tokens.nextToken());
tokens.nextToken(); // Get "("
double x = Double.parseDouble(tokens.nextToken());
tokens.nextToken(); // Get ","
double y = Double.parseDouble(tokens.nextToken());
tokens.nextToken(); // Get ","
double z = Double.parseDouble(tokens.nextToken())
```

# Comunicaciones (III)

- Uso de templates y conversationID

```
MessageTemplate template = MessageTemplate.and(
      MessageTemplate.MatchPerformative(ACLMessage.INFORM),
      MessageTemplate.MatchConversationId("SHOT"));
ACLMessage msg = receive(template);
if ( msg != null ) {
          .
          .
         Agent does some actions
          .
          .
}
else
         block();
```

# Especificación

- Arquitectura
- Taxonomía de Agentes
- Tareas
- Bucle de Ejecución
- Comunicaciones
- **Servicios**
- Interfaz (API)

# Especificación

- Arquitectura
- Taxonomía de Agentes
- Tareas
- Bucle de Ejecución
- Comunicaciones
- **Servicio**
  - Registro
  - Solicitud
  - Respuesta
- Interfaz (API)

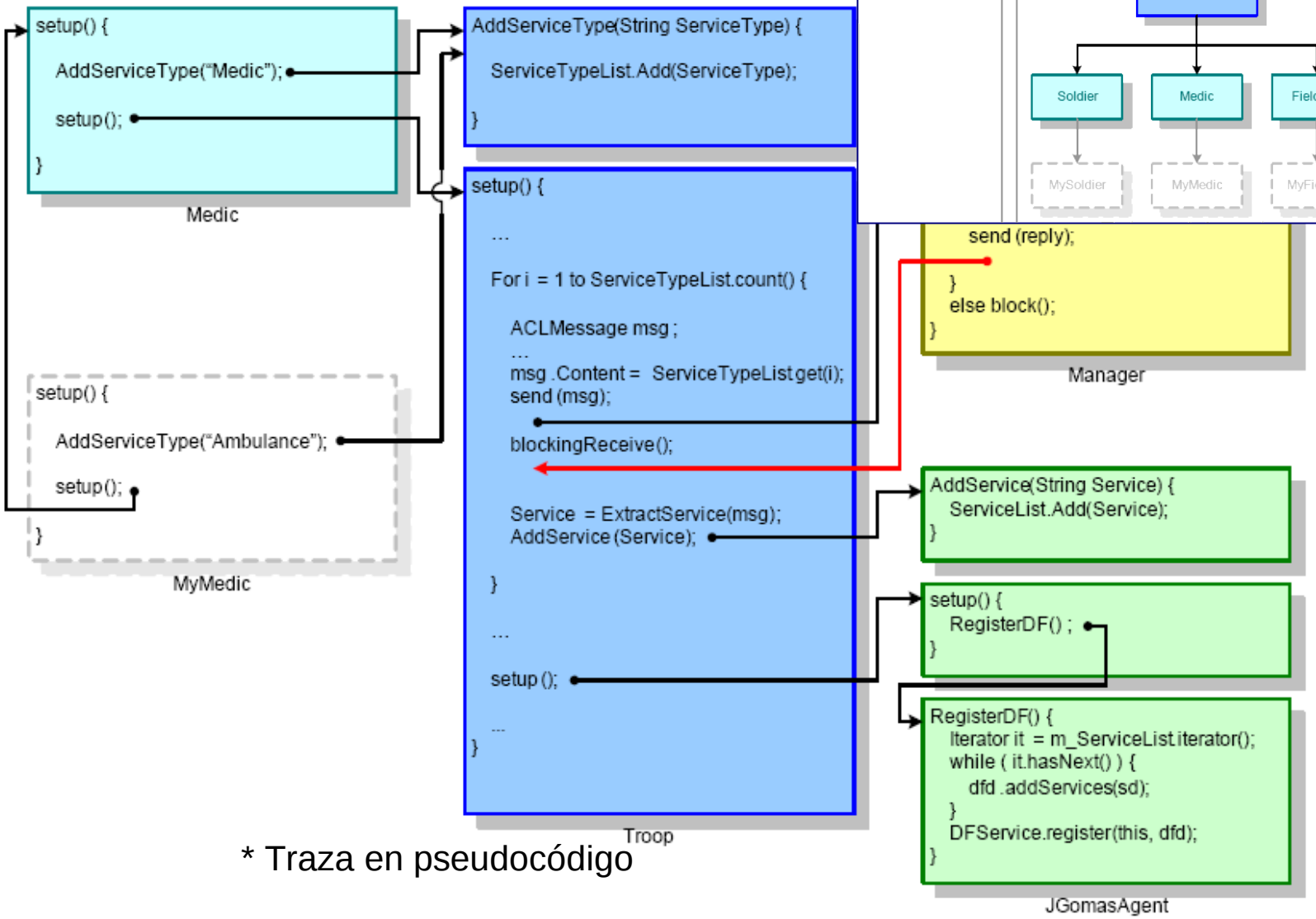
# Servicios: Registro (I)

- Un rol debe registrar un servicio para que el resto de roles puedan solicitarlo:

```
void setup() {
…
AddServiceType("Medic");
super.setup();
…
}
```

- Existen definidos tres tipos de servicios básicos:
  - **m_sMedicService;**
  - **m_sAmmoService;**
  - **m_sBackupService;**

# Servicios: Registro* (II




* Traza en pseudocódigo

# Servicios: Solicitud (I)

```
protected void CallForMedic() {
        try {
```

Solicitamos al DF (páginas amarillas) un listado de aquellos agentes que proporcionen el servicio "`m_sMedicService`"

Si hay alguno en la partida...

Creamos un mensaje ACL tipo REQUEST
Añadimos todos los agentes del listado recibido del DF como receptores del mensaje

ConversationID = "CFM"
Contenido = "( x , y , z ) ( vida )"

Enviamos el mensaje

Sino, sabemos que no hay medicos

```
        } catch (FIPAException fe)
                fe.printStackTrace();
}
```

# Servicios: Solicitud (II)

```
protected void CallForMedic() {
      try {
            DFAgentDescription dfd = new DFAgentDescription();
            ServiceDescription sd = new ServiceDescription();
            sd.setType(m_sMedicService);
            dfd.addServices(sd);
            DFAgentDescription[] result = DFService.search(this, dfd);

            if ( result.length > 0 ) {
                  m_iMedicsCount = result.length;
                  // Fill the REQUEST message
                  ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
                  for ( int i = 0; i < result.length; i++ ) {
                        DFAgentDescription dfdMedic = result[i];
                        AID Medic = dfdMedic.getName();
                        if ( ! Medic.equals(getName()) )
                              msg.addReceiver(dfdMedic.getName());
                        else
                              m_iMedicsCount--;
                  }
                  msg.setProtocol(FIPANames.InteractionProtocol.FIPA_REQUEST);
                  msg.setConversationId("CFM");
                  msg.setContent(" ( " + m_Movement.m_Position.x + " , "
                        + m_Movement.m_Position.y + " , "
                        + m_Movement.m_Position.z + " ) ( "
                        + m_iHealth + " ) ");
                  send(msg);
                  System.out.println(getLocalName()+ ": Need a Medic!");
            }
            else
                  m_iMedicsCount = 0;
      } catch (FIPAException fe)
            fe.printStackTrace();
}
```

# Servicios: Respuesta (I)

```
private void Launch_CFM_ResponderBehaviour() {
        // Behaviour to handle a Call For Medic request
        addBehaviour(new CyclicBehaviour() {
                public void action() {

                }
        });
}
```

Filtrado de Mensajes: (Tipo == REQUEST) and (ConversationID == CFM)

Recibimos una petición y ...

si decidimos aceptar la petición
crear tarea TASK_GIVE_MEDICPACKS
performativa = AGREE
sino
performativa = REFUSE

Contestamos al agente que solicita la petición

Bloqueamos el Comportamiento del Agente

# Servicios: Respuesta (II)

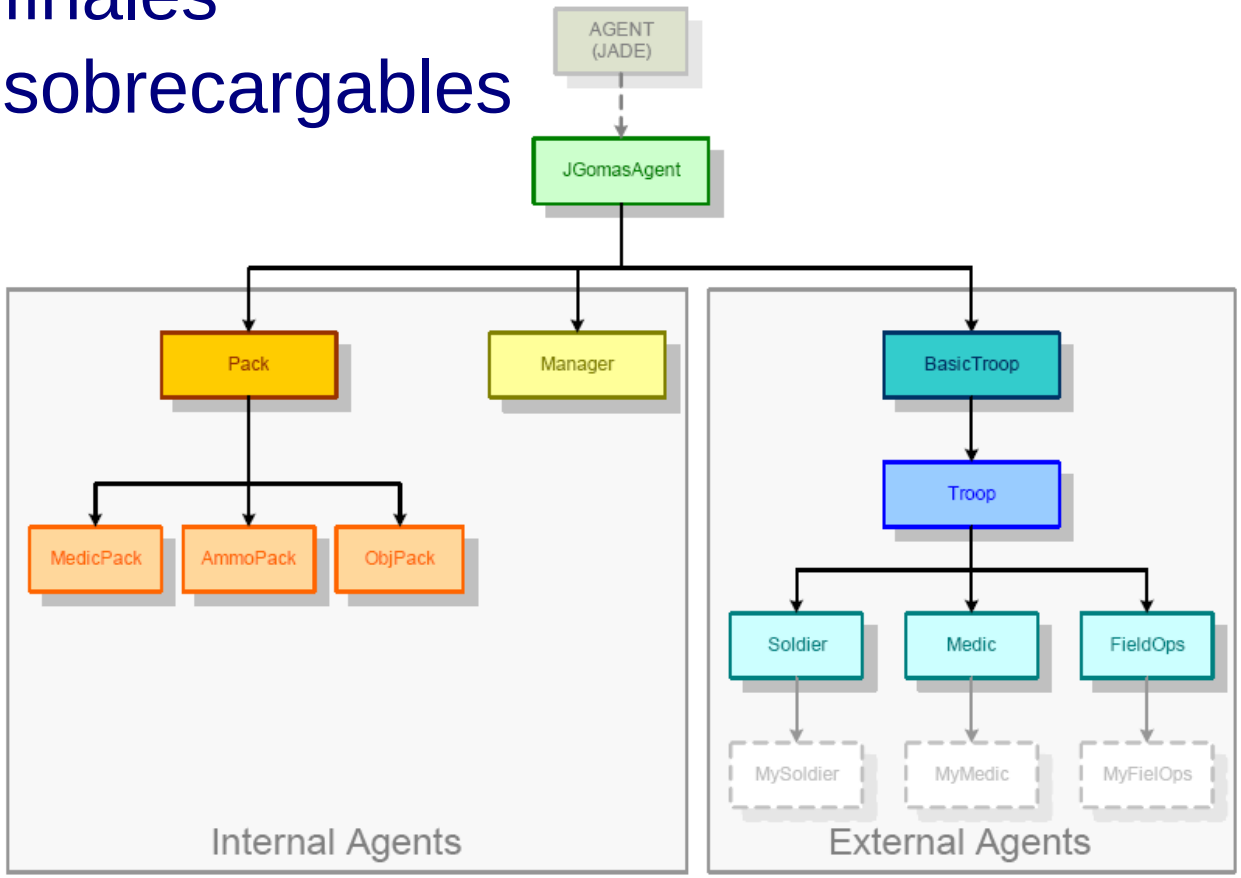
```
private void Launch_CFM_ResponderBehaviour() {
        // Behaviour to handle a Call For Medic request
        addBehaviour(new CyclicBehaviour() {
                        public void action() {
                                MessageTemplate template = MessageTemplate.and(
                                MessageTemplate.MatchPerformative(ACLMessage.REQUEST),
                                MessageTemplate.MatchConversationId("CFM"));

                                int iPerformative;
                                ACLMessage msgCFM = receive(template);
                                if ( msgCFM != null ) {

                                        AID owner = msgCFM.getSender();
                                        String sContent = msgCFM.getContent();

                                        if ( checkMedicAction(sContent) ) {
                                                AddTask(TASK_GIVE_MEDICPAKS, owner, sContent);
                                                iPerformative = ACLMessage.AGREE;
                                        }
                                        else {
                                                iPerformative = ACLMessage.REFUSE;
                                        }
                                        ACLMessage reply = msgCFM.createReply();
                                        reply.setContent(sContent);
                                        reply.setPerformative(iPerformative);
                                        send(reply);
                                }
                                else block();

                        }
        });
}
```

# Especificación

- Arquitectura
- Taxonomía de Agentes
- Tareas
- Bucle de Ejecución
- Comunicaciones
- Servicios
- **Interfaz (API)**

# Interfaz (API)

- CBasicTroop
  - Métodos finales
  - Métodos sobrecargables
  - Atributos



# API (I): Métodos finales

- `boolean` **`CheckStaticPosition ()`**
- `boolean` **`CheckStaticPosition (double _x, double _z)`**

- `void` **`AddTask (int _tTypeOfTask, AID _Owner, String _sContent)`**
- `void` **`AddTask (int _tTypeOfTask, AID _Owner, String _sContent, int _iPriority)`**

**Revisar la documentación de la web**

# API (II): Métodos sobrecargables

- `void` **`PerformNoAmmoAction ()`**
- `void` **`PerformTargetReached (CTask _CurrentTask)`**
- `boolean` **`GeneratePath ()`**
- `boolean` **`GetAgentToAim ()`**
- `void` **`PerformLookAction ()`**

# API (III): Atributos

- `CTask m_CurrentTask`
- `ArrayList m_FOVObjects`
- `CSight m_AimedAgent`
- `CMobile m_Movement`
- `CTerrainMap m_Map`

# Ejercicios a Realizar

- **Ejercicio 1** - Implementar un agente aliado que vaya mostrando por pantalla su posición y su distancia hasta la bandera y cuando la tenga su distancia hasta la base.
- **Ejercicio 2** - Implementar un agente aliado/defensor “loco” que vaya cambiando su posición aleatoriamente.
- **Ejercicio 3** - Implementar un agente aliado/defensor que localice al compañero “loco” y lo vaya siguiendo.

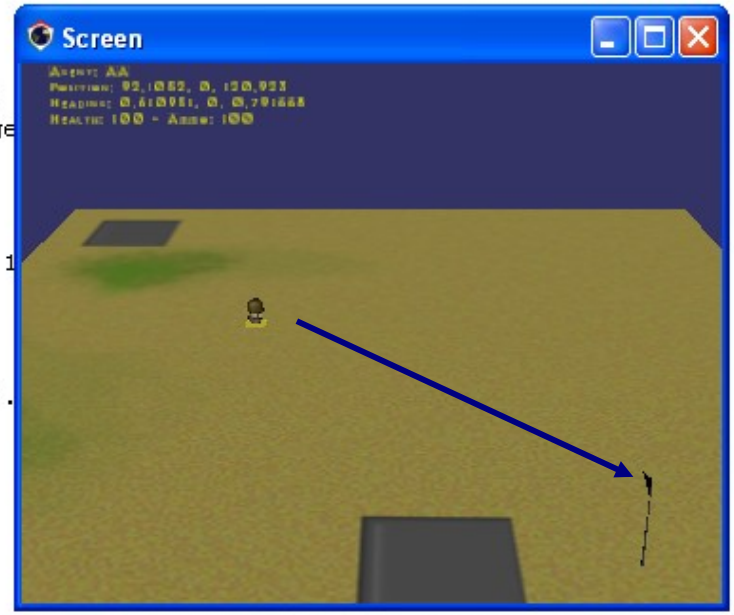
# *Ayuda*

- **Ejercicio 1** : Implementar un agente aliado que vaya mostrando por pantalla su posición y su distancia hasta el destino (bandera).
  - Atributo CMobile **m_Movement**:
    - → La clase *CMobile* contiene:
      - Atributos: Vector3D **m_Position**, Vector 3D **m_Destination**, etc.
      - Métodos de acceso: Vector3D **getPosition(),** …
      - → La clase *Vector3D* contiene los siguientes atributos: double **x**, double **y** y double **z**
  - - Atributo CTerrainMap **m_Map**
    - → La clase *CTerrainMap* contiene los métodos : double **GetTargetX**(), double **GetTargetY**(), double **GetTargetZ**(), …

# *Ayuda*

- **Ejercicio 1** : Implementar un agente aliado que vaya mostrando por pantalla su posición y su distancia hasta el destino (bandera).

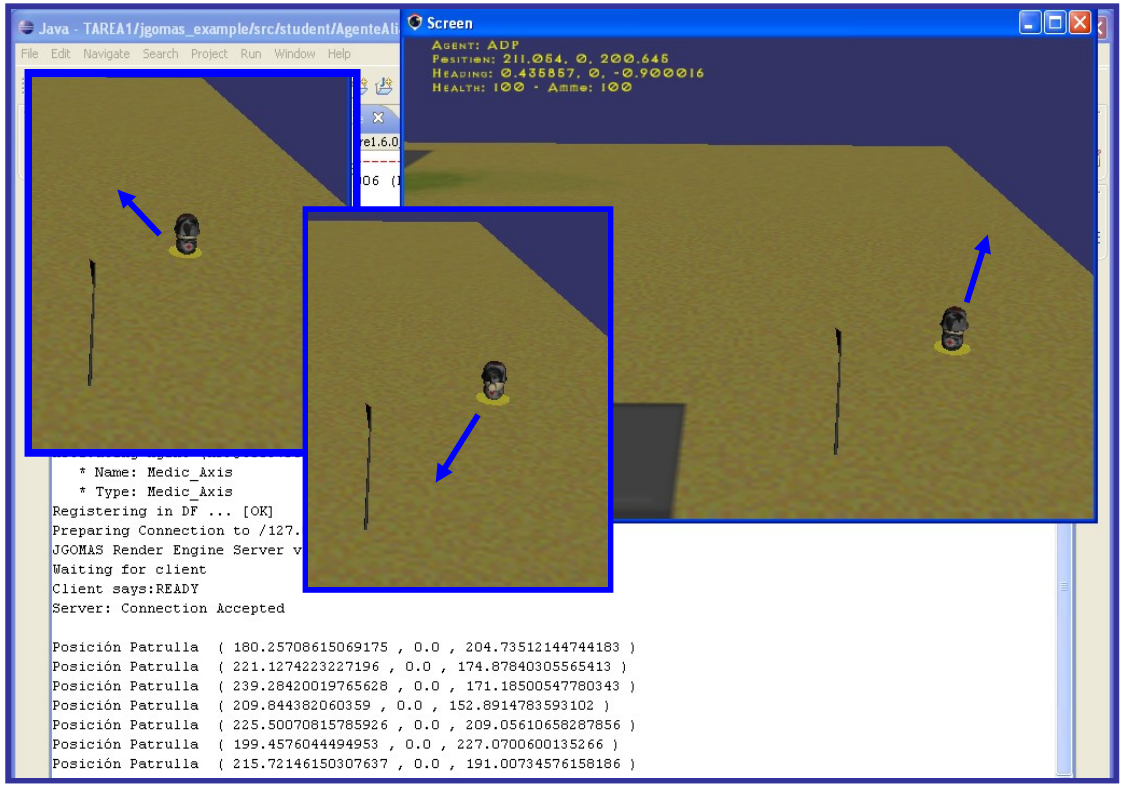
```
Manager (Expected Agents): 1
Manager: [AA] is Ready!
Manager (Accepted Agents): 1
Manager[1]: Sending notification to fight to:
Manager: Sending Objective notification to age
[AA@G003790:1099/JADE]: Beginning to fight
Registry - Service Registered: Medic_Allied

Activating Agent (AA@G003790:1099/JADE - id: 1
   * Name: Medic_Allied
   * Type: Medic_Allied
Registering in DF ... [OK]
La posición de la bandera (184.0, 0.0, 240.0).
Preparing Connection to /127.0.0.1:1247
JGOMAS Render Engine Server v. 0.1.0
Waiting for client
Client says:READY
Server: Connection Accepted

POSICION: (37.00469180687401, 0.0, 49.524000109486096) y DISTANCIA: 240.6007630184974
POSICION: (49.26159399862552, 0.0, 65.4064505715557) y DISTANCIA: 220.5387620211328
POSICION: (61.519718090562584, 0.0, 81.29048436756074) y DISTANCIA: 200.4747610271646
POSICION: (73.71919087345057, 0.0, 97.09851800496682) y DISTANCIA: 180.50676003403092
POSICION: (85.97731496538745, 0.0, 112.98255180097186) y DISTANCIA: 160.4427590400628
```



# *Ayuda*

- **Ejercicio 2**: Implementar un agente aliado/defensor “loco” que vaya cambiando su posición aleatoriamente.

**GenerateEscapePosition(), GeneratePath()** y **PerformThresholdAction()**



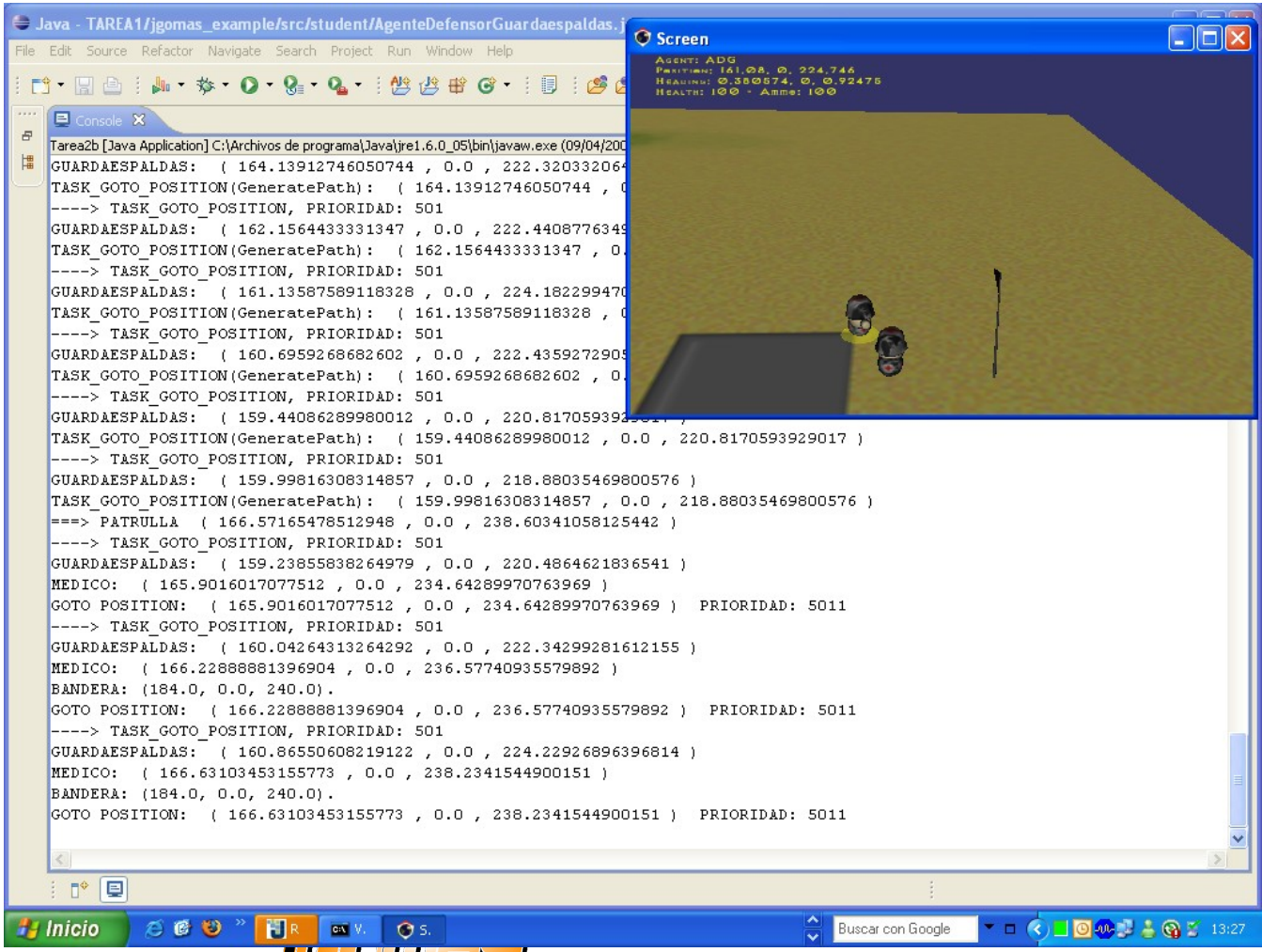
# *Ayuda*

- **Ejercicio 3:** Implementar un agente aliado/defensor que localice al compañero “loco” y lo vaya siguiendo.
  - Campo de visión: **Look()** => ArrayList **m_FOVObjects**.
  - **GetAgentToAim()** para conocer como tratar m_FOVObjects.
  - Los objetos de la lista son de la clases ***CSight***:
    - → La clase ***CSight*** contiene especialmente los métodos **getPosition()**, **getType()**, **getTeam()**, **getHealth()**, **…**

      **MIRAR CONSTANTES PARA TIPO, TEAM, …**

# Ayuda

- Implementar un agente aliado/defensor que localice al compañero “loco” y lo vaya siguiendo.



# *Ayuda*

- Otras pistas:
  - Añadir nuevas tareas:

    ***AddTask(CTask.Nombre_TAREA, getAID(), NewPosition, Priority)***

    ***AddTask(CTask.Nombre_TAREA, getAID(), NewPosition)***
  - Tipos de tareas pueden ser:
    **CTask.*TASK_PATROLLING*,**
    **CTask.*TASK_GOTO_POSITION*,**
    **CTask.*TASK_GIVE_BACKUP*,** …
  - Prioridad: ***m_CurrentTask.getPriority() + 1***