



**BESA3-ALPHA1**  
**Manual de Usuario Programador**  
Pontificia Universidad Javeriana  
Departamento de Ingeniería de Sistemas  
Grupo de Investigación SIDRe  
Septiembre 20 de 2011

 <b>Pontificia Universidad JAVERIANA</b> Bogotá	Departamento de Ingeniería de Sistemas.	Manual de Usuario Programador BESA- ALPHA1.
	Septiembre 20 de 2011.	Versión 1.0.


## Historial de Revisiones

<b>Fecha</b>	<b>Descripción</b>	<b>Autor</b>
<b>20/09/11</b>	Creación del documento.	SIDRe

 <b>Pontificia Universidad JAVERIANA</b> Bogotá	Departamento de Ingeniería de Sistemas.	Manual de Usuario Programador BESA- ALPHA1.
	Septiembre 20 de 2011.	Versión 1.0.

## Tabla de contenido

Introducción.....	4
Modelo de Implementación.....	7
Componentes de Implementación.....	8
Creación de contenedor.....	8
Creación de un contenedor Centralizado.....	8
Creación de un contenedor Remoto.....	9
Creación de un agente BESA.....	9
Envío de Eventos.....	12
Movilidad de Agentes.....	13
Eliminación de Agentes y Contenedores.....	13
Ejemplo.....	14

 <b>Pontificia Universidad JAVERIANA</b> Bogotá	Departamento de Ingeniería de Sistemas.	Manual de Usuario Programador BESA- ALPHA1.
	Septiembre 20 de 2011.	Versión 1.0.

## Introducción


El presente documento consiste en el manual de usuario de BESA3-ALPHA1. En primera instancia se aborda el concepto de sistema multitagente (SMA) comenzando por el concepto de agente. Luego se describe la plataforma de agentes BESA, junto con el modelo de implementación y los componentes de implementación.

## Agente

Un agente es una entidad autónoma que actúa para cumplir sus objetivos en un ambiente de trabajo. Las características necesarias para que un agente pueda ser considerado como tal, y por lo tanto, debe cumplir, son las siguientes:

- Situado: el agente debe interactuar con su ambiente de trabajo, donde el ambiente de trabajo es comprendido como el medio dentro del cual se encuentra inmersos los agentes, entidades externas y objetos pasivos.
- Proactivo: el agente siempre debe tratar de lograr sus intenciones o metas.
- Autónomo: el agente debe poder actuar sin la necesidad de la intervención humana y su conducta debe estar definida por su propia experiencia.
- Social: El agente debe poder establecer relaciones con otros agentes para comunicarse, compartir recursos y habilidades con el fin de lograr de manera conjunta, objetivos comunes.

El ambiente de trabajo se compone de: agentes cooperativos (agentes que comparten los mismos objetivos) y entidades externas (son aquellas entidades que no comparten los objetivos del agente). Las entidades externas las hay de dos tipos: actores (son aquellas entidades con la capacidad de modificar su estado interno o situación en el ambiente autónomamente, ejemplo: agentes externos al sistema, usuarios humanos o informáticos) y objetos pasivos (son aquellas entidades que carecen de la capacidad de modificar su estado interno o situación en el ambiente autónomamente, ejemplo: un balón, un archivo).

	Departamento de Ingeniería de Sistemas.	Manual de Usuario Programador BESA-ALPHA1.
	Septiembre 20 de 2011.	Versión 1.0.

Los agentes se componen de adaptadores (son elementos genéricos por medio de los cuales los agentes son capaces de interactuar con las entidades externas). Los adaptadores se categorizan en dos grupos. En el primer grupo se encuentran los actuadores (componentes que permiten al agente afectar el estado del ambiente de trabajo). Y en el segundo grupo se encuentran los sensores (componentes que le permiten al agente percibir los eventos provenientes del medio). Un evento es la ocurrencia de un suceso en el ambiente de trabajo. Cada vez que hay un cambio en el estado del ambiente de trabajo se dice que ocurre un evento.


Según Russell, existen dos tipos generales de agentes: el agente reactivo y el agente racional. El agente reactivo frente a la presencia de un estímulo actúa inmediatamente, sin la mediación de la reflexión de que acción adoptar. Mientras que un agente racional frente a la presencia de un estímulo, reflexiona acerca de lo que conoce y de lo que percibe, en busca de determinar cuál será la decisión o acción a adoptar.

La principal función de un agente es de decidir o deducir cual es la acción más adecuada para alcanzar una meta específica. Esta funcionalidad en la plataforma de agentes BESA se realiza por medio de una función de toma de decisiones denominada función de mapeo, la cual puede ser expresada de diferentes maneras.

## Sistemas Multiagente (SMA)

En vista de la necesidad de plantear soluciones reales a problemas complejos, en donde sistemas estructurados difícilmente pueden resolver. Las organizaciones de agentes, gracias a su capacidad de poder simular situaciones en donde interviene el no determinismo. Se presenta como un instrumento para enfrentar problemas complejos, a través de la descomposición del problema general en problemas suficientemente pequeños para que un agente pueda resolver. La suma de las pequeñas tareas realizadas por los agentes, a través de la comunicación y la cooperación, dan como resultado un sistema denominado sistema multi-agente (SMA).

Para Ferber, un SMA es un conjunto organizado de agentes que interactúa de forma cooperativa para lograr de manera colectiva un objetivo global, donde SMA se

	Departamento de Ingeniería de Sistemas.	Manual de Usuario Programador BESA-ALPHA1.
	Septiembre 20 de 2011.	Versión 1.0.


compone de un ambiente, objetos, agentes, relaciones entre objetos, operaciones (acciones de percibir, producir, consumir, transformar y manipular).

La complejidad de la función de mapeo es directamente proporcional a la complejidad del problema a la que responde la meta del agente. Para problemas muy complejos la utilización de un solo agente no es suficiente, por lo que se precisa de varios agentes que contribuyen para alcanzar el objetivo en común, de modo que un SMA se puede definir como una colección de agentes que cooperan para alcanzar una meta.

## Plataforma Agentes BESA

BESA es una plataforma desarrollada en el Departamento de Ingeniería de Sistemas de la Pontificia Universidad Javeriana, que facilita la implementación y ejecución de sistemas multi-agentes (SMA). El acrónimo: BESA toma su nombre en ingles y encapsula los conceptos: Behavior-Oriented, Event-Driven, Social-Based, Agent Framework; los cuales son los principios sobre los que se fundamenta las soluciones creadas a partir de la plataforma. A continuación se da una idea básica de cada principio:

- **Control Orientado a Comportamientos:** un agente se compone de un conjunto de comportamientos concurrentes, en donde un comportamiento se entiende como una entidad en la que se encapsula lo necesario para garantizar la realización de un propósito bien definido.
- **Dinámica Basada en Eventos:** un agente debe estar listo para reaccionar a múltiples señales no determinísticas, tal que para el agente, lo relevante no sea la información de la señal, sino el hecho de haber recibido un estímulo, el cual debe reaccionar para recibir una respuesta. Esta señal se puede interpretar como un evento en el que puede incluir información sobre lo que ha sucedido en el ambiente.
- **Mecanismos Sociales de Cooperación:** para reducir la complejidad de un problema, se parte de la descomposición y asignación de tareas, que ejecutadas bajo un orden coherente, posibilitan la obtención de la solución. El orden coherente se sustenta en una comunicación adecuada y de un manejo racional de los recursos. Para ello se hace uso del concepto de micro-

 <b>Pontificia Universidad JAVERIANA</b> Bogotá	Departamento de Ingeniería de Sistemas.	Manual de Usuario Programador BESA- ALPHA1.
	Septiembre 20 de 2011.	Versión 1.0.

sociedad, el cual, una sociedad de agentes puede ser vista como un agente, debido a que cada sociedad, tiene asociado un agente con el rol de administrar y publicar las tareas asignadas a los agentes de la micro-sociedad. Lo anterior significa que la unidad más simple del sistema es el agente y más compleja es el sistema de micro-sociedades.


## Modelo de Implementación

BESA3-ALPHA1 esta Implementado en JAVA y funciona desde la maquina virtual 1.4. Por estar hecho en java BESA es portable por lo que le permite funcionar en multiplataforma. BESA a nivel de procesos consta de un canal y unos comportamientos por agente que son implementados por hilos concurrentes (todo transparente para el usuario). Lo que permite que el usuario se desentienda de la implementación de hilos y solo se concentre en la lógica de negocio del SMA. A continuación se enlistan las clases abstractas de la plataforma:

## Clases Abstractas

- Agente
- Estado
- Guarda
- Adaptador
- Agente Social

Las clases abstractas determinan un modelo de implementación para el desarrollo de los SMA de usuario, es decir, el modelo de desarrollo es por extensión. Por ejemplo, si deseo crear un agente BESA debo primero crear una clase que extienda de AgentBESA. De esta forma mi clase hereda las características de lo que la hace un agente BESA. En el siguiente capítulo se explica con mayor detalle los mecanismos de implementación de cada componente requerido por la plataforma.

 <b>Pontificia Universidad JAVERIANA</b> Bogotá	Departamento de Ingeniería de Sistemas.	Manual de Usuario Programador BESA- ALPHA1.
	Septiembre 20 de 2011.	Versión 1.0.

## Componentes de Implementación

En el presente capítulo se describe por cada componente de implementación el modo y el conjunto de instrucciones requerido para la implementación de un SMA con la plataforma de agentes BESA.

### Creación de contenedor

La plataforma BESA comprende dos tipos de contenedores de acuerdo a la naturaleza del entorno en el que se desenvuelven los agentes. El primer tipo corresponde a un SMA que se despliega de forma centralizada, es decir, que basta con un computador para su funcionamiento. Y el segundo tipo es al que corresponde a un SMA que se despliega de forma distribuida con el propósito de establecer comunicación con otros contenedores BESA alojados en diferentes máquinas. A nivel de código la forma de crear un contenedor es promedio de la siguiente instrucción:


```
AdmBESA adm = AdmBESA.getInstance();
```

La creación de un contenedor BESA centralizado o distribuido se especifica por medio del archivo 'confbesa.xml' de configuración que se debe situar en el directorio del proyecto en curso. Pero de igual forma, de no existir el archivo, por defecto siempre se crea un contenedor centralizado. A continuación se describe los modos de configuración para obtener los tipos de contenedores.

### Creación de un contenedor Centralizado

Para obtener un contenedor centralizado el archivo 'confbesa.xml' debe mantener la siguiente estructura:



 <b>Pontificia Universidad JAVERIANA</b> Bogotá	Departamento de Ingeniería de Sistemas.	Manual de Usuario Programador BESA- ALPHA1.
	Septiembre 20 de 2011.	Versión 1.0.

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
  <container alias = "MAS_1" password = "0.91" ipaddress= "127.0.0.1">
    <environment seneventattempts = "10" sendeventtimeout = "1" rmitimeout
= "1000"/>
  </container>
</config>
```

En donde la etiqueta 'config' se define la etiqueta 'container' la cual por propiedades se especifica el alias del contenedor, la contraseña y la dirección IP. Luego dentro se define la etiqueta 'environment' la cual por propiedades se especifica los parámetros de configuración de la comunicación del entorno.

### Creación de un contenedor Remoto


Para obtener un contenedor centralizado el archivo 'confbesa.xml' debe mantener la siguiente estructura:

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
  <container alias = "MAS_1" password = "0.91" ipaddress= "127.0.0.1">
    <environment seneventattempts = "10" sendeventtimeout = "1" rmitimeout
= "1000">
      <remote rmiport = "1099" mcaddress = "230.0.0.1" mcport = "2222"/>
    </environment>
  </container>
</config>
```

Que consta de las mismas etiquetas que la configuración anterior pero con una adicional que es la etiqueta 'remote' la cual por propiedades se especifica el puerto de registro rmi, la dirección y puerto multidifusión.

### Creación de un agente BESA

De acuerdo al modelo de implementación por herencia, para crear un agente BESA basta con extender de la clase AgentBESA e implementar sus métodos abstractos: setupAgent y shutdownAgent y conformar el súper constructor:

 <b>Pontificia Universidad JAVERIANA</b> Bogotá	Departamento de Ingeniería de Sistemas.	Manual de Usuario Programador BESA- ALPHA1.
	Septiembre 20 de 2011.	Versión 1.0.

```

/**
 * This class represents the odometer agent.
 *
 * @author SIDRe - Pontificia Universidad Javeriana
 * @author Takina - Pontificia Universidad Javeriana
 * @version 3.0, 20/09/11
 * @since JDK1.4
 */
public class AgentOdo extends AgentBESA {

```

```

/**
 * Creates a new instance of AgentOdo.
 *
 * @param alias Agent alias.
 * @param state Agent state.
 * @param structOdo Agent struct.
 * @param passwd Password.
 */
public AgentOdo(String alias, StateOdo state, StructBESA structOdo,
double passwd) throws KernellAgentExceptionBESA {
    super(alias, (StateBESA) state, structOdo, passwd);
}

/**
 * Setups agent.
 */
@Override
public void setupAgent() {
    StateOdo stateOdo = (StateOdo) this.state;
    stateOdo.setCounter(new CounterOdo(INT_COUNTER));           //
    Initialized the counter.
    stateOdo.setState(StateOdo.State.Init);                     //Sets the
    initial state.
    ArrayList v = new ArrayList();
    v.add(new Integer(1));
    stateOdo.initState(v);
}

/**
 * Shutdown agen.
 */
@Override
public void shutdownAgent() {
}
}

```

El método `setupAgent` permite inicializar y reservar los recursos utilizados por el agente (en el código se aprecia como inicializa el estado del agente) y `shutdownAgent` permite liberar los recursos en el momento en que se elimina del contenedor al agente. Como se puede observar el constructor de la clase está obligado a recibir cuatro parámetros que constituyen la creación del agente: el alias del agente, la contraseña, la estructura y el estado. A continuación se describe su modo de instanciación siguiendo el modelo de implementación:

### ***Creación del Estado de Agente***


Para crear el estado de un agente es necesario extender tan solo de la clase `StateBESA` como se aprecia:

```
public class StateOdo extends StateBESA {  
  
    /**  
     * Enumeration that indicates the posible states of the odometer.  
     */  
    public enum State {  
        Init, Start, Stop  
    }  
    .  
    .  
    .  
}
```

El está por lo regular mantiene una referencia a todos tipos de datos concernientes a la lógica de negocio del agente.

### ***Creación de la Estructura del Agente***

La estructura del agente comprende de una colección de duplas <comportamiento, guarda>, que en castellano significa la especificación de la asociación de los comportamientos del agente y de las guardas asociadas a estos comportamientos. Para tal efecto se sigue lo siguiente:

 <b>Pontificia Universidad JAVERIANA</b> Bogotá	Departamento de Ingeniería de Sistemas.	Manual de Usuario Programador BESA- ALPHA1.
	Septiembre 20 de 2011.	Versión 1.0.

```

StructBESA structOdo = new StructBESA();
try {
    structOdo.addBehavior("BehaviorOdo");
    structOdo.bindGuard("BehaviorOdo", GuardOdoModify.class);
    structOdo.bindGuard("BehaviorOdo", GuardMoveOdo.class);
} catch (ExceptionBESA ex) {
    ReportBESA.error(ex);
}

```

## Envió de Eventos


Un evento se envía de una agente a otro por medio del método 'sendEvent' que se obtiene por medio del manejador del agente. Por lo tanto, para enviar un evento es preciso obtener primero el manejador del agente y luego se procede con el envío del evento:

```

AgHandlerBESA ah = null;
EventBESA ev = new EventBESA(GuardOdoModify.class.getName(),
datOdo);
try {
    ah = AdmBESA.getInstance().getHandlerByAlias("Odometro_1");
    ah.sendEvent(ev);
} catch (Exception ex) {
    ReportBESA.error(ex.getMessage());
}

```

Como se puede observar el manejador de un agente se puede obtener por medio del administrador del contenedor. En este caso se obtuvo el manejador a través del alias del agente. Para el envío de un evento es preciso crear el evento, un evento BESA consta del tipo de evento (el nombre de la clase de la guarda asociada) y los datos, que para tal efecto la plataforma dispuso la clase DataBESA para su extensión:

 <b>Pontificia Universidad JAVERIANA</b> Bogotá	Departamento de Ingeniería de Sistemas.	Manual de Usuario Programador BESA- ALPHA1.
	Septiembre 20 de 2011.	Versión 1.0.

```

public class DataOdo extends DataBESA {

    /**
     * Message.
     */
    private String message;
    /**
     * Value.
     */
    private String valueString;

    /**
     * Creates a new instance of DataOdo.
     */
    public DataOdo(String mensaje) {
        this.message = mensaje;
    }
    .
    .
    .
}

```

## Movilidad de Agentes

La movilidad de agentes consiste en mover un agente de un contenedor a otro. Esta función solo está permitida para SMA distribuidos. Para mover una agente tan solo hay que seguir lo siguiente:

```


try {
    AdmBESA.getInstance().moveAgent("AgentB", "MAS_1", 0.71);
} catch (ExceptionBESA ex) {
    ReportBESA.error(ex);
}

```

Es decir, especificar el agente a mover por medio del alias, el destino del agente y la contraseña del agente lo que me convalida la acción.

## Eliminación de Agentes y Contenedores

La muerte de los agentes se efectúa por medio del administrador del contenedor. Para eliminar un agente se especifica su ID y la contraseña del agente:

 <b>Pontificia Universidad JAVERIANA</b> Bogotá	Departamento de Ingeniería de Sistemas.	Manual de Usuario Programador BESA- ALPHA1.
	Septiembre 20 de 2011.	Versión 1.0.

```
try {
    admLocal.killAgent(ah.getAgId(), 0.91);
} catch (ExceptionBESA ex) {
    ReportBESA.error(ex);
}
```


Y para eliminar el contenedor se sigue de:

```
try {
    admLocal.kill(0.91);
} catch (ExceptionBESA ex) {
    ReportBESA.error(ex);
}
```

Donde 0.91 corresponde a la contraseña del contenedor, lo que me convalida la acción.

## Ejemplo


A continuación se muestra el Main completo del el proyecto Odómetro:

 <b>Pontificia Universidad JAVERIANA</b> Bogotá	Departamento de Ingeniería de Sistemas.	Manual de Usuario Programador BESA- ALPHA1.
	Septiembre 20 de 2011.	Versión 1.0.

```

public static void main(String[] args) {
    //-----//
    // Creates and starts the BESA container.          //
    //-----//
    AdmBESA adm = AdmBESA.getInstance();
    //-----//
    // Creates the display agent.                      //
    //-----//
    DisplayAgent displayAgent = null;
    String alias = "Display";
    DisplayState displaySate = new DisplayState();
    StructBESA displayStruct = new StructBESA();
    double passwdAg = 0.91;
    try {
        displayStruct.bindGuard(DisplayGuard.class);           //Binds
        GuardOdoRequest: Displays the progress the odometer counter
        displayStruct.bindGuard(StopGuard.class);             //Binds
        StopOdoGuard: Time out guard that control the live cycle of odometer.
        displayAgent = new DisplayAgent(alias, displaySate, displayStruct,
        passwdAg);
    } catch (ExceptionBESA ex) {
        ReportBESA.error(ex);
    }
    displayAgent.start();
    //-----//
    // Creates the odometer agent.                      //
    //-----//
    AgentOdo agOdo = null;
    alias = "Odometro_1";
    StateOdo stateAgOdo = new StateOdo();
    StructBESA structOdo = new StructBESA();
    passwdAg = 0.92;
    try {
        structOdo.addBehavior("BehaviorOdo");
        structOdo.bindGuard("BehaviorOdo", GuardOdoModify.class); //
        Binds GuardOdoModify: Simulates the progress the odometer counter.
        structOdo.bindGuard("BehaviorOdo", GuardMoveOdo.class); //
        Binds GuardMoveOdo: Periodic guard that directed the progress the odometer
        counter.
        agOdo = new AgentOdo(alias, stateAgOdo, structOdo, passwdAg);
    } catch (ExceptionBESA ex) {
        ReportBESA.error(ex);
    }
}

```

 <b>Pontificia Universidad JAVERIANA</b> Bogotá	Departamento de Ingeniería de Sistemas.	Manual de Usuario Programador BESA- ALPHA1.
	Septiembre 20 de 2011.	Versión 1.0.

```

agOdo.start();

//-----//
// Starts the odometer, creates a PeriodicDataBESA guard and start //
// command (for starts the process the increments the counter). The //
// PeriodicDataBESA is a data type predefine for the configuration of //
// the periodic guard. //
//-----//
AgHandlerBESA ah = null;
PeriodicDataBESA periodicData = new
PeriodicDataBESA(PERIODIC_TIME,
PeriodicGuardBESA.START_PERIODIC_CALL);//Creates the start message
for periodic guard.
EventBESA startPeriodicEv = new
EventBESA(GuardMoveOdo.class.getName(), periodicData);//Creates the
event and sends to the GuardMoveOdo.
try {
    ah = adm.getHandlerByAid(agOdo.getAid());
    ah.sendEvent(startPeriodicEv);
} catch (ExceptionBESA e) {
    e.printStackTrace();
}
//-----//
// Creates a event with a start message for starts time-out //
// GuardBESA. //
//-----//
EventBESA startTimeOut = new
EventBESA(StopGuard.class.getName(), null);
try {
    ah = adm.getHandlerByAid(displayAgent.getAid());
    ah.sendEvent(startTimeOut);
} catch (ExceptionBESA e) {
    e.printStackTrace();
}
}

```