9. Ontologías

9.1 El Problema de la Comunicación

Para que dos agentes colaboren, no basta con que se envíen mensajes; deben **entenderse**. Tenemos 3 estrategias evolutivas para enviar datos dentro de un mensaje ACL.

9.1.1 Cadenas de Texto (Strings)

- **Método:** Escribir todo en un String plano. Ejemplo: `"valor=10;fecha=20/10"`.
- **Problema:** Es fácil para humanos, pero ineficiente para máquinas. Requiere *parsear* (analizar sintácticamente) el texto, lo cual es propenso a errores y computacionalmente costoso .

9.1.2 Objetos Serializables (Java Serialization)

- **Método:** Enviar objetos Java directamente (`implements Serializable`).
- **Problema:** Crea un **acoplamiento fuerte**. Todos los agentes deben estar programados en Java. Si un agente está en Python o C++, no podrá "deserializar" el objeto Java .

9.1.3 Ontologías(La Solución Estándar)

- **Método:** Definir un modelo formal del dominio (clases y jerarquías) independiente del lenguaje de programación.
- **Ventaja:** Proporciona **interoperabilidad real**. Un agente en Java y otro en C++ pueden entender el mismo concepto "Oferta" si comparten la misma ontología, independientemente de cómo lo representen internamente .

9.2 Definición y Teoría de Ontologías

Una ontología no es solo código; es un contrato semántico. Una ontología es una especificación explícita de una conceptualización. Define el vocabulario (tipos, propiedades y relaciones) de un dominio específico sin ambigüedad.

Qué ganamos usándolas?

1. **Eliminación de Ambigüedad:** Impide construir mensajes "mal formados". Si la ontología dice que una `Persona` tiene `Nombre` y `Edad`, no puedes enviar una `Persona` con `Marca` y `Modelo`.
2. **Validación Semántica:** Si un agente recibe un mensaje que viola la ontología, lo descarta automáticamente (en JADE lanza una excepción), protegiendo la lógica

del sistema.

3. **Desacoplamiento:** La ontología define el *qué* (estructura), separándolo del *cómo* (implementación en Java/C#).

9.3 Arquitectura de Ontologías en JADE

JADE implementa este estándar a través del paquete `jade.content`. Para entenderlo, debemos diferenciar tres tipos de elementos que componen cualquier ontología en este framework:

- **Concept (Concepto):**
  - Representa objetos o entidades pasivas del dominio
  - Ejemplo: `Persona, Vinilo, Coche`
  - *Nota técnica*: Suelen implementarse como Java Beans (getters/setters) y pueden contener otros conceptos.
- **Predicate (Predicado):**
  - Representa hechos o estados del mundo que pueden ser **Verdaderos o Falsos**
  - Ejemplo: `Trabajar`, `EsDueñoDe`
  - Relaciona conceptos entre sí
- **AgentAction (Acción del Agente):**
  - Es un tipo especial de Concepto que representa una orden o acción a realizar (Verbos imperativos)
  - Ejemplo: `Ofertar`, `Pedir`, `Vender`
  - *Importante*: A diferencia de los conceptos normales, estos sí tienen sentido como el "contenido principal" de un mensaje comunicativo.

> **Detalle de Implementación sobre Acciones:** Un `AgentAction` nunca viaja solo. En JADE, debe envolverse en una clase genérica llamada `Action` que incluye quién realiza la acción (`AID` del agente) y la acción en sí .

9.4 Estrucutra Interna: Esquema vs Beans

Para que JADE maneje las ontologías, divide el trabajo en dos capas. Es vital no confundirlas:

9.4.1 La Clase `Ontology` (El Mapa)

- Hereda de `jade.content.onto.Ontology`.
- Define la **estructura abstracta** (los esquemas). Dice: "En este mundo existe un concepto 'Persona' que tiene un slot 'nombre'".
- Es un **Singleton**: Solo debe haber una instancia de esta clase en toda la ejecución, ya que el esquema no cambia .

9.4.2 Los Beans (Los Datos)

- Son clases Java normales que implementan interfaces (`Concept`, `Predicate`, `AgentAction`).
- Contienen los **datos reales**: "Esta instancia específica es la Persona 'Juan'".
- Se crean y destruyen continuamente durante la ejecución .

9.5 El Proceso de Comunicación: El ContentManager

¿Cómo viaja un objeto Java por la red? Aquí entra el **Lenguaje de Contenido (Content Language)**.

9.5.1 Lenguajes Soportados

JADE traduce los objetos a una secuencia de bytes o caracteres usando un "Códec". Hay dos principales:

1. **SL (Semantic Language):** Legible por humanos (parecido a LISP). Es el estándar de facto para compatibilidad .
2. **LEAP:** Binario y comprimido. No es legible por humanos. Se usa en entornos de bajo rendimiento o móviles .

9.5.2. El Flujo de Trabajo (Pipeline)

El `ContentManager` es el componente del agente que gestiona esta traducción:

1. **Registro:** Al iniciar, el agente debe registrar qué Ontología y qué Lenguaje va a usar:

JAVA

```
getContentManager().registerLanguage(new SLCodec());
getContentManager().registerOntology(MiOntologia.getInstance());
```

2. **Enviar (Fill):** El agente crea sus objetos Java (Beans), y el ContentManager los "inyecta" en el mensaje, traduciéndolos al lenguaje elegido (SL/LEAP).
   - Método: `fillContent(ACLMessage msg, Element content)`.

3. **Recibir (Extract):** El receptor toma el mensaje y el ContentManager reconstruye los objetos Java a partir de los bytes recibidos.
   - Método: `extractContent(ACLMessage msg)`.

9.6. Herramientas de Desarrollo: Protégé

Escribir ontologías a mano es tedioso y propenso a errores. El documento recomienda usar herramientas gráficas .

- **Protégé:** Software de la Universidad de Stanford para dibujar ontologías gráficamente (clases, relaciones).

- **OntologyBeanGenerator (OBG):** Un plugin para Protégé que **genera el código Java automáticamente**. Crea tanto la clase `Ontology` (esquemas) como los Beans con sus getters y setters, listos para usar en JADE .
   - _Nota:_ Requiere versiones antiguas de Protégé (3.4).

9.7 Intento de Entederlo todo

Tienda de Discos,imagina esta situación:

- **Agente A (Comprador):** Quiere enviar un mensaje que diga: _"Quiero comprar el disco 'Thriller' de Michael Jackson"_.
- **Agente B (Vendedor):** Tiene que recibir ese objeto Java y entenderlo.

Paso 1: Definir las "Cosas" (Los Beans)

Primero creamos las clases Java que representan los datos. Estas son las piezas de LEGO que vamos a intercambiar.

1. El Concepto (El Sustantivo): Un `Vinilo` es un objeto pasivo. Implementa `Concept`.

```java
import jade.content.Concept;

public class Vinilo implements Concept {
```

```
    private String titulo;
    private String artista;

    // Es OBLIGATORIO tener un constructor vacío y getters/setters
    public Vinilo() {}

    public String getTitulo() { return titulo; }
    public void setTitulo(String titulo) { this.titulo = titulo; }

    public String getArtista() { return artista; }
    public void setArtista(String artista) { this.artista = artista; }
}
```

2. La Acción (El Verbo): `Pedir` es algo que un agente hace. Implementa `AgentAction`. Fíjate que contiene un `Vinilo` dentro.

JAVA

```
import jade.content.AgentAction;

public class Pedir implements AgentAction {
    private Vinilo viniloDeseado; // La acción lleva asociado el objeto

    public Pedir() {}

    public Vinilo getViniloDeseado() { return viniloDeseado; }
    public void setViniloDeseado(Vinilo vinilo) { this.viniloDeseado =
vinilo; }
}
```

Paso 2: El Mapa (La Clase Ontology)

Aquí es donde le damos nombre a las cosas para JADE. Esta clase une tus clases Java (`Vinilo.class`) con un nombre semántico ("Vinilo").

JAVA

```java
import jade.content.onto.*;
import jade.content.schema.*;

public class TiendaOntology extends Ontology {
    // Patrón Singleton: Solo una instancia para todos
    public static final String ONTOLOGY_NAME = "Ontologia-Tienda";
    private static Ontology instancia = new TiendaOntology();

    public static Ontology getInstance() { return instancia; }

    private TiendaOntology() {
        super(ONTOLOGY_NAME, BasicOntology.getInstance()); // Hereda tipos básicos (String, Integer)

        try {
            // 1. Definimos el Concepto "Vinilo"
            add(new ConceptSchema("Vinilo"), Vinilo.class);

            // 2. Definimos la estructura interna (Slots) del Vinilo
            ConceptSchema cs = (ConceptSchema) getSchema("Vinilo");
            cs.add("titulo", (PrimitiveSchema) getSchema(BasicOntology.STRING));
            cs.add("artista", (PrimitiveSchema) getSchema(BasicOntology.STRING));

            // 3. Definimos la Acción "Pedir"
            add(new AgentActionSchema("Pedir"), Pedir.class);

            // 4. Estructura de Pedir (contiene un concepto Vinilo)
            AgentActionSchema as = (AgentActionSchema) getSchema("Pedir");
            as.add("viniloDeseado", (ConceptSchema) getSchema("Vinilo"));

        } catch (OntologyException oe) {
            oe.printStackTrace();
        }
    }
}
```

Paso 3: El Agente Emisor (Cliente)

Ahora vamos a usarlo. El agente crea el objeto, lo rellena y el `ContentManager` lo convierte en mensaje.

JAVA

```java
// DENTRO DEL AGENTE COMPRADOR (setup o behaviour)

// 1. Registrar lenguaje y ontología (OBLIGATORIO)
ContentManager cm = getContentManager();
cm.registerLanguage(new SLCodec());
cm.registerOntology(TiendaOntology.getInstance());

// 2. Crear los datos (Java normal)
Vinilo v = new Vinilo();
v.setTitulo("Thriller");
v.setArtista("Michael Jackson");

Pedir accionPedir = new Pedir();
accionPedir.setViniloDeseado(v);

// 3. Preparar el mensaje ACL
ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
msg.addReceiver(new AID("Vendedor", AID.ISLOCALNAME));
msg.setLanguage(new SLCodec().getName());           // "FIPA-SL"
msg.setOntology(TiendaOntology.getInstance().getName()); // "Ontologia-
Tienda"

// 4. LA MAGIA: Rellenar el contenido
// Las acciones siempre van envueltas en un "Action" genérico que indica
QUIÉN lo hace
Action wrapper = new Action(msg.getReceiver().next(), accionPedir);

try {
    cm.fillContent(msg, wrapper); // Traduce Java -> Lenguaje SL
    send(msg);
} catch (Exception e) { e.printStackTrace(); }
```

Paso 4: El Agente Receptor (Vendedor)

El vendedor recibe un mensaje y extrae el objeto Java directamente.

JAVA

```java
// DENTRO DEL AGENTE VENDEDOR

// 1. Registrar lo mismo que el emisor
ContentManager cm = getContentManager();
cm.registerLanguage(new SLCodec());
cm.registerOntology(TiendaOntology.getInstance());

ACLMessage msg = receive();

if (msg != null) {
    try {
        // 2. Extraer el contenido (Bytes -> Java)
        ContentElement contenido = cm.extractContent(msg);

        // 3. Verificar qué es
        if (contenido instanceof Action) {
            Action wrapper = (Action) contenido;

            // Miramos qué acción hay dentro del sobre
            if (wrapper.getAction() instanceof Pedir) {
                Pedir peticion = (Pedir) wrapper.getAction();

                // ¡YA TENEMOS EL OBJETO! Acceso directo a los datos
                System.out.println("Me han pedido el disco: " +
                    peticion.getViniloDeseado().getTitulo());
            }
        }
    } catch (Exception e) { e.printStackTrace(); }
}
```

Resumen Visual

1. **Beans:** Creas `class Vinilo` (Java puro).
2. **Ontology:** Creas `class TiendaOntology` para enseñar a JADE que `class Vinilo` equivale al concepto "Vinilo".
3. **ContentManager:** Es el traductor.
   - **Emisor:** `fillContent()` (Java → Mensaje).
   - **Receptor:** `extractContent()` (Mensaje → Java).

No envías textos sueltos, envías estructuras definidas. Si intentaras enviar un `Coche` en el campo de `Vinilo`, JADE daría error antes de salir.