

# Desarrollo de Sistemas Multi-Agente con INGENIAS

Jorge Gómez Sanz  
Juan Pavón Mestras

Dep. Sistemas Informáticos y Programación  
Universidad Complutense Madrid

<http://www.fdi.ucm.es/profesor/jpavon>



## INDICE

- Introducción a INGENIAS
  - Metodología
  - Herramientas: Ingenias Development Kit (IDK)
- INGENIAS
  - Notación
  - **Proceso de desarrollo**
  - **Entregas**
- IDK
  - Editor: Manejo
  - Módulos: propósito, estructura, funcionamiento
- Ejemplos de desarrollo: Juul

## Hipótesis

- Desarrollar SMA es complejo
- Soluciones tecnológicas: arquitecturas, plataformas de desarrollo
- Soluciones teóricas: lenguajes de agentes
- Soluciones metodológicas

# Ingeniería del software

## La metodología ayuda

No sé qué problema tengo que resolver → Análisis de sistema, guías

Desarrollar muchos agentes o agentes de elevada complejidad → Actividades, proceso de desarrollo, herramientas

Explicar a otros cómo es el sistema → Análisis justificado

Trabaja más de una persona → Proceso de desarrollo

Cómo evoluciona el proceso de desarrollo → Actividades, resultados, métricas

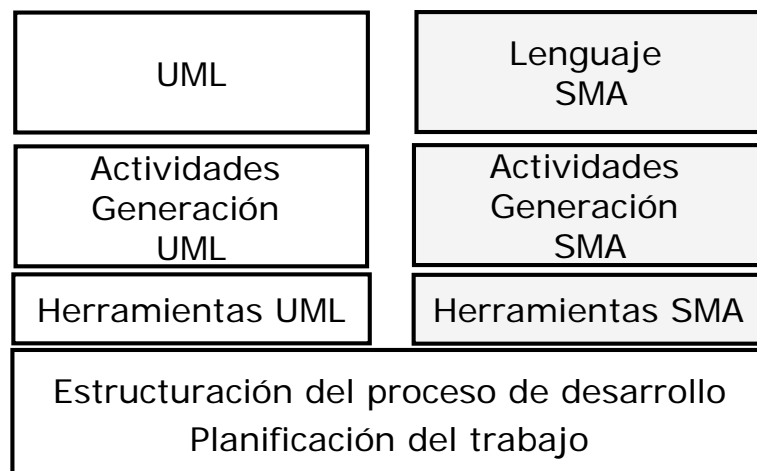
## Metodologías: Estado del arte

- No soportadas por herramientas:
  - BDI: representación del sistema con lógica modal
  - MAS-CommonKADS: sistemas basados en conocimiento
  - GAIA: descripciones con fichas del sistema
- Soportadas por herramientas:
  - Vowel Engineering: guías con vocales
  - ZEUS: descripción operacional
  - MaSE: diagramas UML
- Problemas Comunes
  - Proceso de desarrollo demasiados simples: desarrollos pequeños
  - Cómo se define el modelo de SMA
  - Integración con resultados en investigación

## Objetivo de INGENIAS

- Proporcionar soluciones de ingeniería para desarrollar SMA
  - Notación:
    - Lenguaje visual para expresar el diseño de SMA y agentes
    - Gramática & Semántica
  - Métodos:
    - Organización de entregas
    - Actividades y diagramas de actividades
  - Herramientas: Ingenias Development Kit (IDK)
    - Generación de especificación
    - Generación de código
    - Generación de documentación

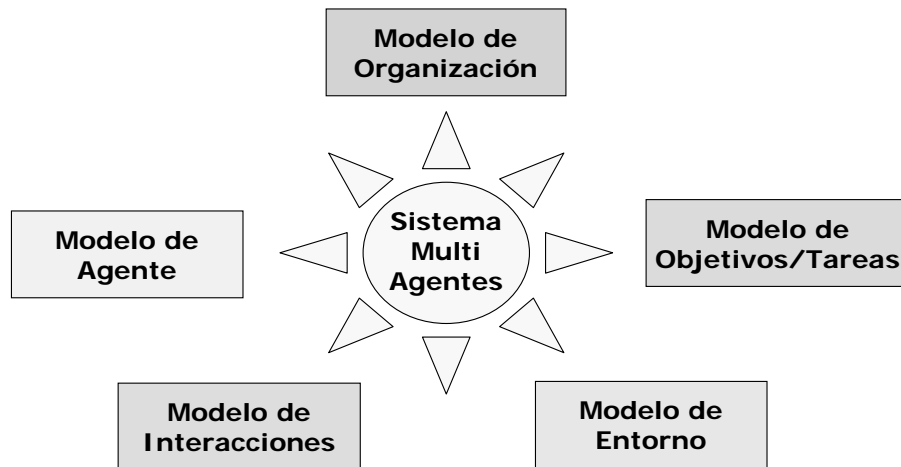
## Cómo se integra el lenguaje dentro del UP



## Notación de INGENIAS

- Lenguaje visual de especificación de SMA
  - Gomez-Sanz, J. J.: *Modelado de Sistemas Multi-Agente*. Tesis Doctoral, Facultad de Informática, UCM, 2002.
  - Gomez-Sanz, J. J., Pavon, J. y Garijo, F.: *Meta-modelling of Multi-Agent Systems*. Proc. ACM Symposium on Applied Computing 2002.
  - Garijo, F., Gomez-Sanz, J. J., Pavon, J. y Massonet, P.: *Multi-Agent System Organization. An Engineering Perspective*. Proc. MAAMAW 2001.

## Aspectos de un SMA



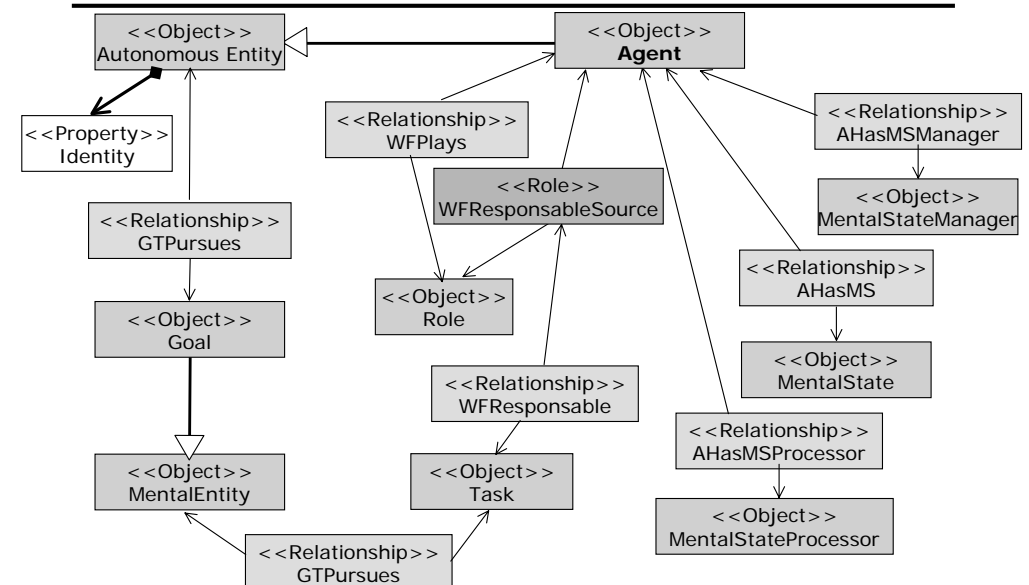
## Meta-modelado

- Lenguajes de meta-modelado
- Especificación de modelos: similar a una gramática
- Justificación
  - Compatible con otros formalismos
  - Extensible, incremental
  - Similitud con UML → Facilitar su aceptación en ingeniería
  - Número reducido de primitivas: Grafo, Objeto, Relación, Propiedad

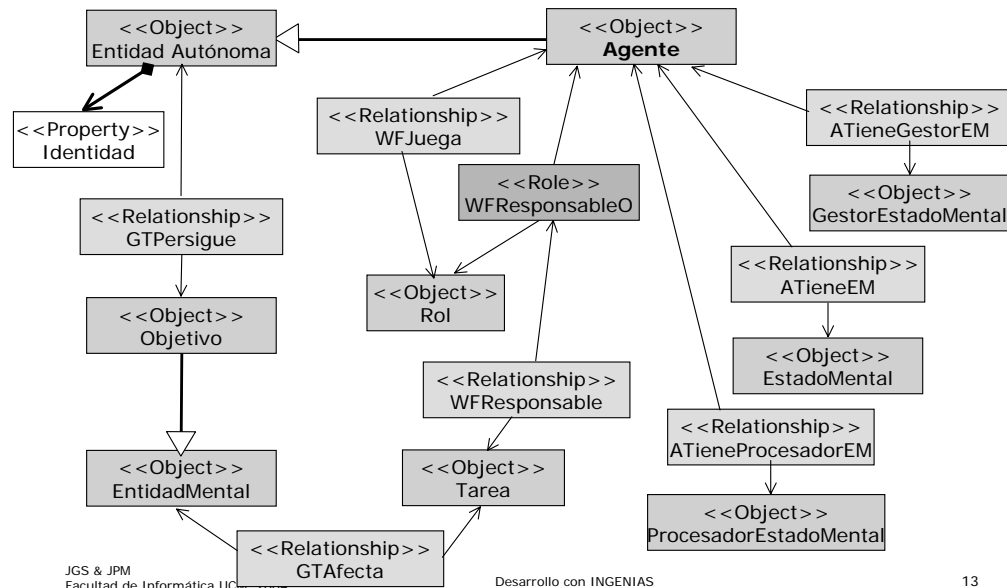
## ¿Qué se modela?

- Modelo de agente
  - Los agentes realizan tareas o persiguen objetivos
  - Responsabilidades, control y estado mental del agente
- Modelo de objetivos y tareas
  - Identificación de objetivos generales y descomposición en objetivos más concretos que se pueden asignar a agentes
  - Similarmente con tareas
  - Objetivos: motivación ⇔ Tareas: actividad
- Modelo de interacción
  - Qué interacciones existen entre agentes/roles
- Modelo de organización
  - Estructura del SMA, roles, relaciones de poder, workflows
- Modelo de entorno
  - Entidades y relaciones con el entorno del SMA

## Meta-modelo de agente

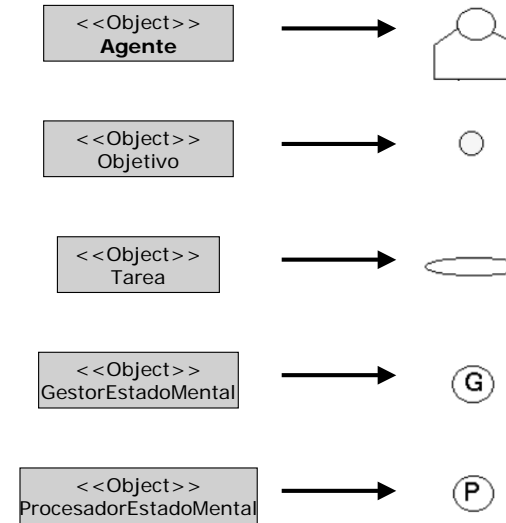


## Meta-modelo de agente



13

## Notación en los modelos



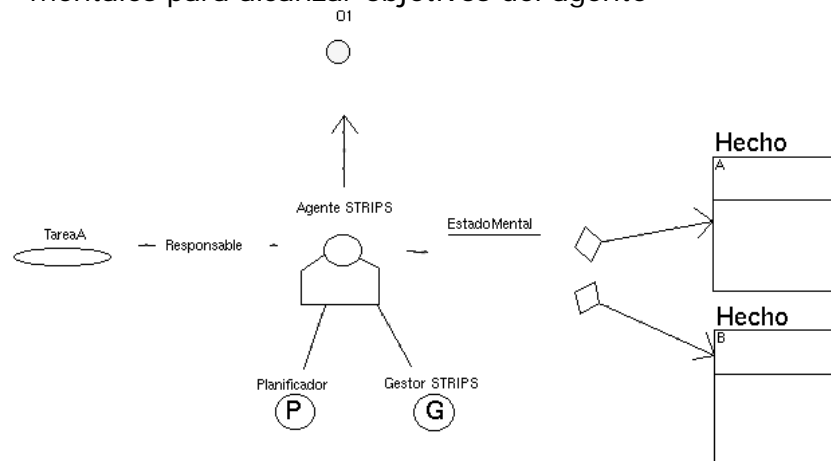
JGS & JPM  
Facultad de Informática UCM, 2004

Desarrollo con INGENIAS

14

## Agente planificador

- Planificador clásico. Las tareas transforman entidades mentales para alcanzar objetivos del agente



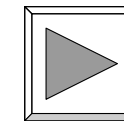
JGS & JPM  
Facultad de Informática UCM, 2004

Desarrollo con INGENIAS

15

## Descripción del meta-modelo de INGENIAS

- Disponible en HTML



- Enumera relaciones, roles, propiedades y entidades
- No está completamente comentada. Para encontrar más detalles acerca de la notación, ir a <http://grasia.fdi.ucm.es/ingenias>

JGS & JPM  
Facultad de Informática UCM, 2004

Desarrollo con INGENIAS

16

## Son muchas entidades y relaciones

### ■ Por ello definimos

- Un proceso de desarrollo
  - Compuesto de actividades
  - Y que determina entregas a realizar
- Un entorno de desarrollo que facilite la implementación

### ■ Y damos

- Ejemplos de modelado
- Una tesis doctoral
- Otras dos próximamente

## INGENIAS Development Kit

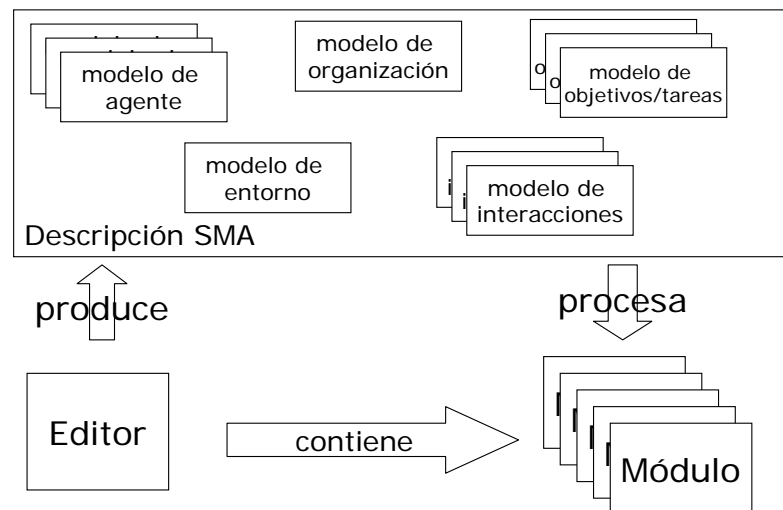
### ■ Editor de modelos

- Herramienta visual (notación *grasia!*)
- Inicialmente basado en herramienta de meta-modelado (METAEDIT+)
  - Actualmente 100% Java
- Generación de modelos siguiendo los meta-modelos
- Integración con módulos para procesamiento de las especificaciones
- Integración con agentes (en desarrollo)

### ■ Módulos:

- Para la generación de código
  - Armazones (plantillas) configurables, especificados con XML, para distintas plataformas de agentes
    - Jade, Robocode, Servlets, Agentes *grasia!*
- Para validar especificaciones: basado en AT
- Para generar documentación (HTML)
- Armazón para desarrollar módulos personalizados

## IDK



## Editor del IDK

### ■ El editor del IDK permite

- Crear y modificar modelos de SMA
- Generar documentación (HTML)
- Sacar snapshots de los diagramas para utilizarlos en otras aplicaciones
- Procesar las especificaciones mientras se están generando con el editor o una vez grabadas en un fichero
- Introducir explicaciones en lenguaje natural de los diferentes diagramas y de cada elemento en los diagramas, así como añadir etiquetas de texto

## Manejo del Editor del IDK

**Diagramas estructurados con paquetes**

**Ontología de entidades admitidas**

JGS & JPM  
Facultad de Informática UCM, 2004

21

## Manejo del Editor del IDK

**Entidades admitidas**

**Panel principal**

JGS & JPM  
Facultad de Informática UCM, 2004

22

## Manejo del Editor del IDK

**Cortar/pegar/ampliar/deshacer/layout**

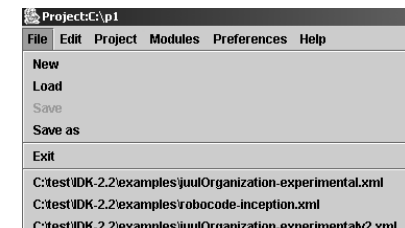
**Mensajes de los módulos**

JGS & JPM  
Facultad de Informática UCM, 2004

23

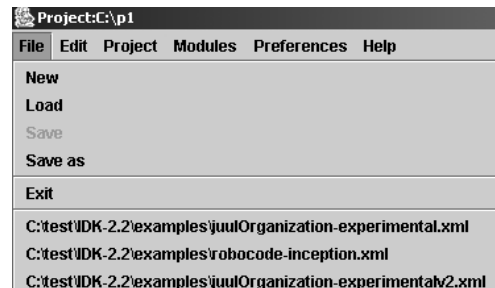
## Cargar diagramas

- En el menú principal File -> Load
- Elegid el fichero
- Pulsad en Aceptar
- O bien elegir alguno de los que aparecen al final del menú
- El editor guarda un histórico de ficheros abiertos



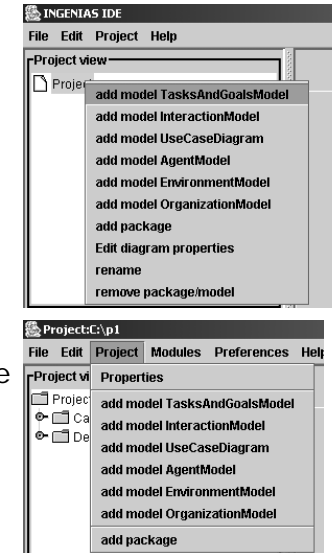
## Guardar diagramas

- En el menú principal File -> Save
- Elegid el nombre del fichero y ubicación
- Pulsad en Aceptar
- Mirad en la sección Messages por si algo hubiera salido mal



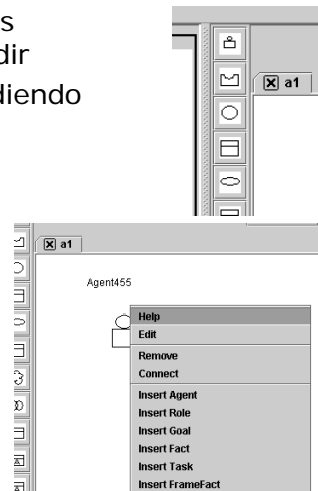
## Creando diagramas

- Pulsando en Project con el botón derecho del ratón
  - Elegid el tipo y escribid un nombre
- 
- O bien eligiendo un paquete donde ubicar el diagrama y desde el menú principal en la opción *Project*



## Añadiendo entidades

- Pulsad en la barra de iconos la entidad que queráis añadir
  - Las barras cambian dependiendo del tipo de diagrama
- 
- O bien desde el panel principal, pulsando con el botón derecho del ratón

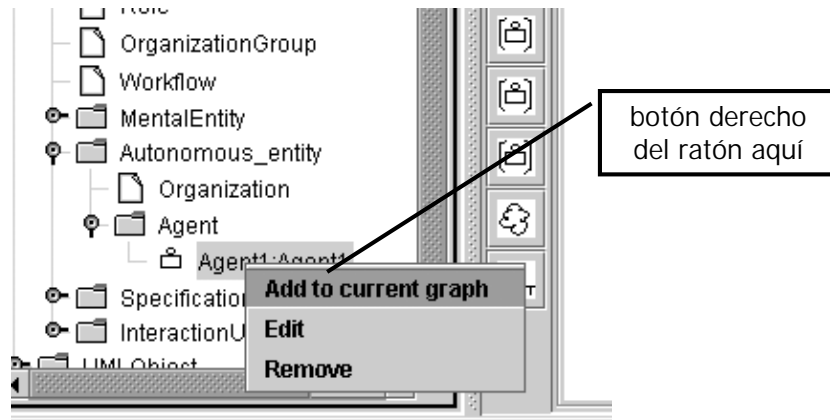


## Añadir elementos existentes

- Dos formas:
  - Forma 1: pulsando en copiar y pegar
  - Forma 2: desde la sección entities view
    - Desplegad las carpetas hasta encontrar la entidad que buscáis
    - Botón derecho del ratón encima de la entidad
    - Elegid add to current diagram

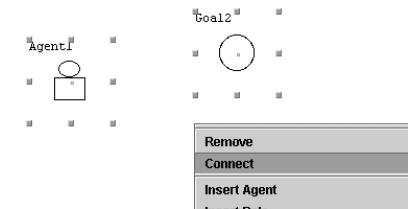


## Añadir elementos existentes



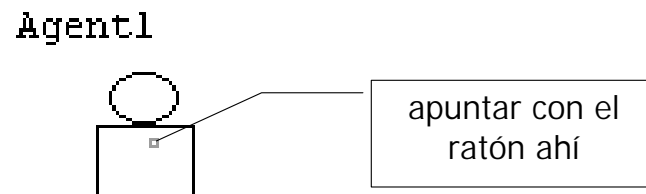
## Conectar dos relaciones: método 1 (1)

- Elegid los elementos a conectar dejando shift (tecla mayúsculas) pulsado
- Pulsad con el botón derecho del ratón en el fondo del diagrama
- Elegid connect



## Conectar dos relaciones: método 2 (1)

- Apuntad al centro del icono. Veréis un pequeño rectángulo.
- Dejad pulsado el botón izquierdo del ratón y arrastrad hasta el mismo rectángulo de otro icono



## Conectar dos relaciones (2)

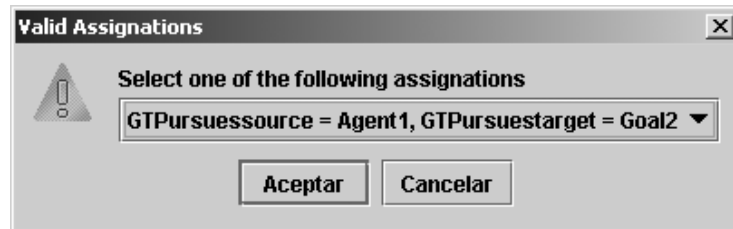
- Si se puede crear la relación os preguntará por el tipo
- Elegid uno y pulsad en Aceptar





## Conectar dos relaciones (3)

- Ahora toca elegir quien será qué extremo de la relación. Es importante si queréis dirigir la relación
- Elegid el sentido que más os convenga



## Otras funcionalidades

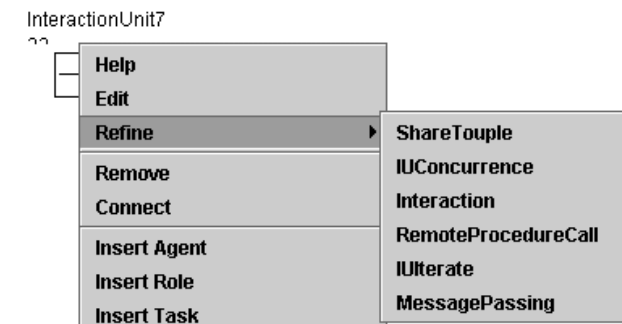
- Para gestionar múltiples diagramas
  - Se pueden crear paquetes. Los paquetes contienen otros paquetes o diagramas. La opción de creación de paquetes son una más del menú de creación de diagramas, también accesible desde el menú Project
  - Para cada diagrama abierto se tiene una pestaña en el panel principal. Estas pestañas se etiquetan con el nombre del diagrama
  - Se puede arrastrar un diagrama a un paquete o un paquete a otro paquete
- Reutilizando
  - Herramienta copiar y pegar. Esta herramienta no copia relaciones
  - Desde el árbol *entities view* pulsando en una entidad se puede añadir al *diagram*

## Otras funcionalidades

- Para documentar
  - Se puede utilizar el módulo de documentación HTML en el menú *modules*
  - Se pueden copiar las imágenes de los diagramas en un fichero o en el portapapeles de Windows: se puede pegar los diagramas en cualquier aplicación (por ejemplo, en word o en powerpoint). Estas opciones aparecen en el menú Edit
- Ayuda
  - Pulsando en una entidad con el botón derecho y eligiendo *help*
  - En menú Help y luego en Manual

## Otras funcionalidades

- Refinando los diagramas
  - Las ontologías se estructuran mediante la herencia
  - Pulsando en una entidad con el botón derecho del ratón, puede aparecer una opción refine que permite transformar una entidad en otras más concretas



## Desarrollo de módulos

---

- Un módulo en IDK es un programa capaz de procesar una especificación INGENIAS
- La utilidad principal de los módulos es traducir las especificaciones a código ejecutable
- Sin embargo, también los utilizamos para
  - Generar documentación
  - Verificar las especificaciones
  - Generar informes de uso de las entidades del meta-modelo

## Conocimientos necesarios

---

- Para desarrollar módulos hay que saber
  - Cómo se recorren los diagramas internos del IDK
  - Cómo se crea una plantilla
  - Cómo se rellena una plantilla
  - Cómo se genera el código
  - Cómo se escribe un módulo
  - Cómo se despliega un módulo

## Cómo se recorren los diagramas

---

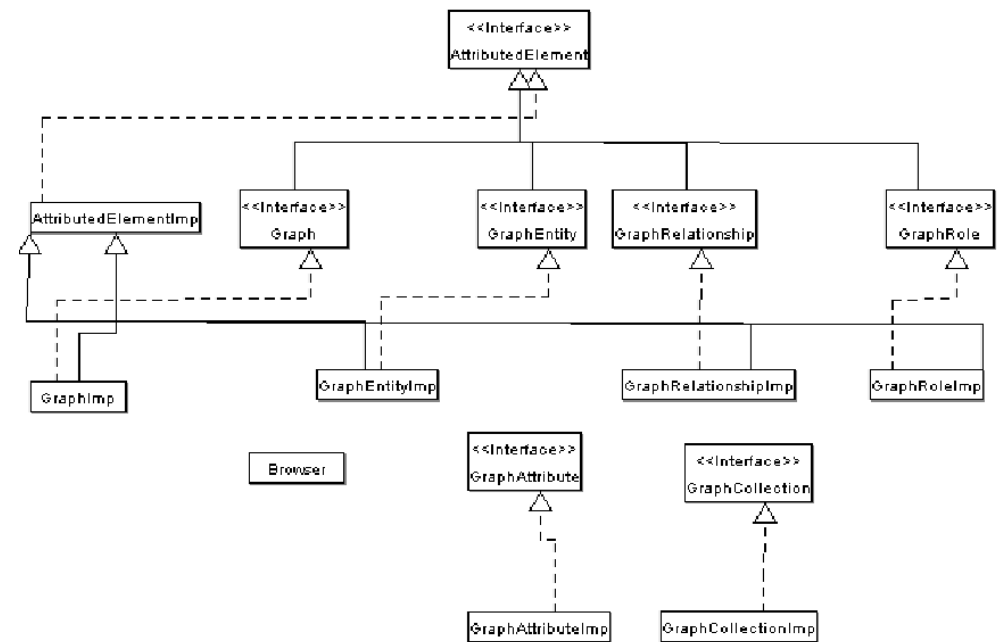
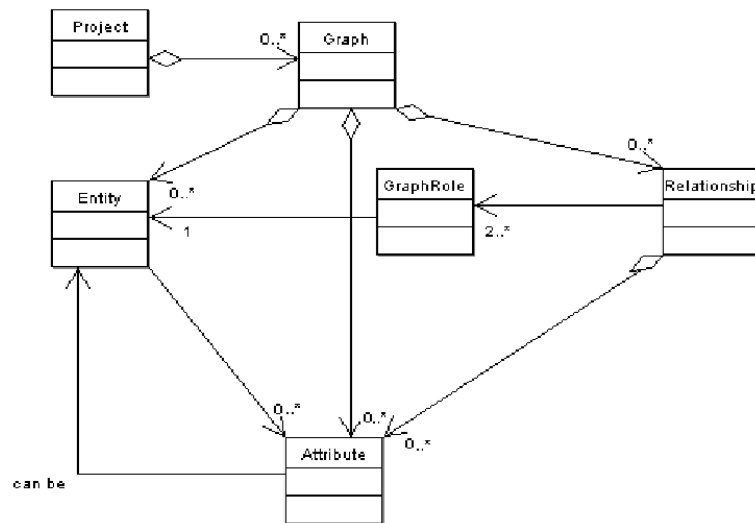
- El funcionamiento interno del editor se basa en la arquitectura de componentes de SWING: requisito del JGRAPH
- Las entidades, relaciones y diagramas de INGENIAS se mapean en diversos componentes del editor
- Para evitar trabajar a bajo nivel se proporcionan un conjunto de entidades que ofrecen esta visión al desarrollador
  - Se proporciona una fachada
  - Esta fachada se puede ver como una representación del grafo en forma de vértices y aristas
- La fachada se construye con un conjunto de interfaces que proporcionan una visión de alto nivel similar a la del lenguaje de meta-modelado GOPRR

## Tipos de elementos

---

- Grafos: contienen entidades unidas por relaciones. También pueden tener atributos. Un grafo puede entenderse como un diagrama
- Entidades: contienen atributos y representan los elementos principales de la especificación. Hay entidades que permiten hacer referencia a un diagrama
- Relaciones: conectan entidades. A los extremos de la relación se les denomina *roles*. Los roles pueden tener también atributos
- Atributos: pueden ser entidades o hacer referencia a otros grafos

## Tipos de elementos



## Accediendo a los diagramas

- El punto de entrada a los diagramas es la clase *ingenias.generator.browser.BrowserImp* que implementa un **patrón singleton**
- Los diagramas se pueden acceder directamente desde el editor mientras se están editando o bien externamente a través del fichero de especificaciones. Para ello se ejecutan los siguientes pasos:
  - Conseguir una referencia al punto de entrada:
    - Cuando el acceso se planea hacerlo mientras se está trabajando con el editor, la inicialización es la
 

```
Browser browser=BrowserImp.getInstance();
```
    - Cuando el acceso se quiere hacer de forma externa, directamente sobre el fichero de especificación INGENIAS, la inicialización es la siguiente
 

```
File file;
....
ingenias.editor.Log.initInstance(new java.io.PrintWriter(System.err));
ingenias.generator.browser.BrowserImp.initialise(file);
Browser browser=BrowserImp.getInstance();
```

## Accediendo a los diagramas

- Una vez inicializado, se pueden utilizar los métodos del browser para acceder a los diagramas y obtener todas las entidades definidas en todos los diagramas
  - El siguiente programa hace un recorrido de todos los diagramas definidos en un editor o fichero INGENIAS
  - Nótese que se usa la clase StringBuffer en lugar de utilizar directamente concatenación. Se hace por eficiencia.

```

Graph[] gs = browser.getGraphs();
StringBuffer result = new StringBuffer();

for (int k = 0; k < gs.length; k++) {
    Graph g = gs[k];
    result.append( "\n##### Diagram " + g.getName() +
                  " #####\n");
    result.append(this.generalInformeDiagrama(g)+"\n");
}
System.out.println(result);

```

## Accediendo a los diagramas

3. En cada diagrama el desarrollador se puede plantear examinar las entidades existentes o bien obtener un listado de las relaciones actuales.

- Ello se hace mediante el API definido en las interfaces vistas antes
- El siguiente fragmento de código del módulo *example* obtiene las relaciones de un diagrama y guarda en una tabla hash las apariciones de cada tipo

```
private void generateADiagramReport(Hashtable stats, Graph g)
{
    GraphRelationship[] grels=g.getRelationships();
    for (int k=0;k<greels.length;k++){
        if (stats.containsKey(grels[k].getType())){
            Integer old=(Integer)stats.get(grels[k].getType());
            stats.put(grels[k].getType(),new
            Integer(old.intValue()+1));
        } else
            stats.put(grels[k].getType(),new Integer(1));
    }
    ....
}
```

## Observaciones

- Acceder a las entidades a través de los diagramas puede no ser conveniente siempre ya que una misma entidad puede pertenecer a varios diagramas
  - Existe un método en la interfaz Browser, *getAllEntities*, que devuelve todas las entidades definidas en el proyecto, sin repeticiones
- Cuando se le pregunta a una entidad por sus relaciones, la entidad dispone de dos métodos
  - **getRelationships**. Proporciona las relaciones de una entidad **en el diagrama del que se la sacó**
  - **getAllRelationships**. Proporciona todas las relaciones de una entidad en todos los diagramas del proyecto
- Las relaciones son n-arias. Esto quiere decir que la relación puede conectar más de dos entidades.

## Cómo se crea una plantilla

- Las plantillas se ven como documentos XML que satisfacen la DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT file (#PCDATA | v)*>
<!ATTLIST file  overwrite (yes|no) #REQUIRED>
<!ELEMENT program (#PCDATA|repeat|saveto|v)*>
<!ELEMENT repeat (#PCDATA | saveto | v | repeat)*>
<!ATTLIST repeat id CDATA #REQUIRED>
<!ELEMENT saveto (file, text)>
<!ELEMENT text (#PCDATA | repeat | v | saveto)*>
<!ELEMENT v (#PCDATA)>
```

- Cada uno de los tags definidos en el DTD está pensado para ser sustituido por información de los diagramas

## ¿Qué significan estos tags?

- Program: marca el inicio del programa
- Saveto: indica que hay que grabar un texto en un fichero. Este tag es el último en procesarse, y limpiará todos los tags no reemplazados
  - File: señala el fichero donde se va a escribir
  - Text: indica qué texto hay que grabar
- Repeat: indica que el texto encerrado entre dos tags Repeat ha de repetirse
- V: indica que hay que insertar un dato

## Problemas de XML

- Estos documentos se van a rellenar utilizando palabras de lenguajes de programación
- Esto plantea algunos inconvenientes
  - Existen caracteres especiales del idioma: palabras acentuadas, eñe
    - Como solución, los documentos XML, en caso de necesitar caracteres especiales latinos, hay que encabezarlos con  
<?xml version="1.0" encoding="ISO-8859-1"?>
    - En lugar de  
<?xml version="1.0" encoding="UTF-8"?>

## Problemas de XML

- Existen símbolos prohibidos en XML como &, <, >, ", '
- Las alternativas que se ofrecen son varias:
  - Convertir los símbolos del código a instanciar a entidades válidas XML, como &amp; &lt; o &gt; → demasiado trabajo
  - Encerrar nuestro código en paquetes PCDATA → demasiado trabajo. Las plantillas no son fácilmente comprensibles
  - Reemplazar los caracteres de los tags de XML, trabajar en nuestro código como si no existiera XML, y luego transformar el documento a XML
    - Sustituimos < y > con @ (ver siguiente transparencia)

```
@program xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../plantilla.xsd"@
  @saveto@
    @file overwrite="yes"@
      @v@output@/v@/index.html@/file@
    @text@
<HTML>
<BODY bgcolor="#FFFFFF">

<p> </p>
<p><font size="5">Specification Diagrams are:</font>
  <BR>
</p>
<ul>

  @repeat id="paquete"@
    <li><font size="4"><b>@v@name@/v@</b></font> </li>
    <ul>
      @repeat id="graph"@
        <li>Diagram name: <A HREF="@v@name@/v@.html">@v@name@/v@</A> type :
<font color="#000099"> @v@tipo@/v@ </font>
        </ li>

      @/repeat@
    </ul>
    <br>
  @/repeat@

<p><font size="3">Document generated automatically with the Ingenias Development
Kit <font color="#993333">
  IDK 2.1</font></font></p>

</BODY>
</HTML>

@/text@
@/saveto@
@/program@
```

## Para trabajar con estos documentos

- Para pasar de un formato a otro tenemos un programa conversor

java -cp "lib\ingeniaseditor.jar" ingenias.generator.util.Conversor [-a2t|-t2a] mifichero

- a2t: significa traducir del formato @ a uno compatible con XML
- t2a: significa traducir del formato XML al formato @
- Con este programa se puede verificar que el formato de los tags es correcto con alguna de las herramientas estándar

## Cómo se rellena una plantilla

- Se trata de reemplazar los tags por información de los diagramas
- La información se crea siguiendo la misma estructura que la indicada por la plantilla
- Ante la duda, tenemos programas que informan de dicha estructura al suministrarles la plantilla

java

```
-cp "lib\ingeniaseditor.jar; lib\xerces_2_3_0.jar; lib\xercesImpl.jar; lib\xerces-J_1.4.0.jar"
ingenias.generator.util.ObtainInstantiationStructure
fichero_de_plantilla
```

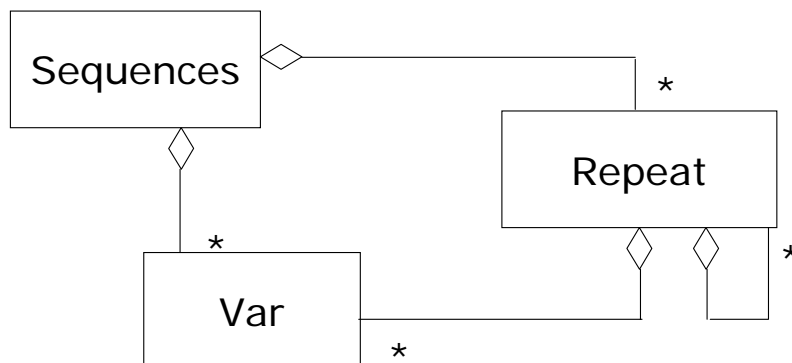
## Estructura a rellenar

```
@program xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="..plantilla.xsd"@
@saveTo@
  @file overwrite="yes"@
  @v@output/v@/index.html@/file@
  @text@
<HTML>
<BODY bgcolor="#FFFFFF">
<p> </p>
<p><font size="5">Specification Diagrams are:</font>
  <BR>
</p>
<ul>
  @repeat id="paquete"@
    <li><font size="4"><b>@v@name@/v@</b></font> </li>
    <ul>
      @repeat id="graph"@
        <li>Diagram name: <A href="@v@name@/v@.html">@v@name@/v@</A> type :
        <font color="#000099"> @v@tipo@/v@ </font>
        </ li>
      @/repeat@
    </ul>
    <br>
  @/repeat@
<p><font size="3">Document generated automatically with the Ingenias Development
Kit <font color="#993333">
  IDK 2.1</font></font></p>
</BODY>
</HTML>
@/text@
@/saveTo@
@/program@
```

v output  
repeat id = paquete  
v name  
repeat id = graph  
v name  
v name  
v tipo

## Creando la estructura (I)

- Internamente, el IDK representa esta estructura con un conjunto de clases



## Creando la estructura (II)

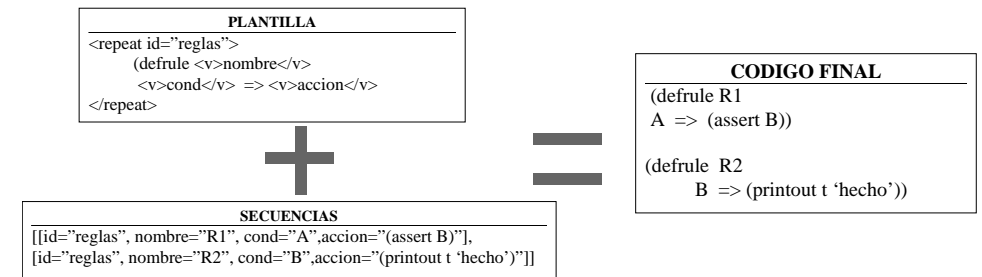
- Para la estructura anterior, deberíamos tener algo como

```
Sequences seq=new Sequences();
Repeat r1=new Repeat("paquete");
Repeat r2=new Repeat("paquete");
seq.add( r1 ); seq.add( r2 );
r1.addVar(new Var("name","mipaquete1"));
r2.addVar(new Var("name","mipaquete2"));
.....
```

## Cómo se genera el código

- La generación de código utiliza las plantillas y el recorrido de diagramas diseñado
- Las plantillas se rellenan con la información extraída de los diagramas durante el recorrido
- Ya se ha explicado
  - cómo se recorre
  - cómo se hace una plantilla
  - con qué estructuras se organiza la información con la que rellenar la plantilla
- La generación es un proceso automático una vez se tiene esta información

## Ejemplo



## Lanzando la sustitución de código

- El proceso se lanza al invocar

```
Sequence seq;
InputStream is;
...
ingenias.generator.interpreter.Codegen.applyArroba(seq.toString(),is);
```

- Al método se le pasa como parámetro la secuencia que contiene los datos y un flujo de entrada a la plantilla a rellenar

## Cómo se escribe un módulo

- Hay partes en el proceso de generación de código que son automatizables. De hecho, lo que depende del usuario es la plantilla y el recorrido de la especificación. Por ello se han creado clases que reducen el trabajo de escribir un módulo. Son las clases: *BasicToolImp* y *BasicCodeGeneratorImp*
- Para escribir un módulo, hay que extender alguna de estas dos clases e implementar los métodos abstractos que éstas definen
- *BasicToolImp* es un módulo libre en el sentido de que no tiene por qué utilizarse para generar código. Sería el caso de un módulo para examinar los diagramas y ver si falta algún elemento o bien hay alguna inconsistencia

## Cómo se escribe un módulo

---

- *BasicCodeGeneratorImp* es una clase dedicada para aquellos módulos que generen código
- Actualmente, los métodos que tiene deben:
  - Devolver el nombre del módulo
  - Devolver una descripción del módulo
  - Devolver las propiedades externas configurables del módulo
  - Devolver una estructura *Sequence* con la que rellenar las plantillas. Esta estructura contendría la información extraída de la especificación durante un recorrido
  - Informar del nombre de las plantillas a utilizar

## Cómo se despliega un módulo

---

- El IDK es capaz de cargar módulos en tiempo de ejecución
- Todo se basa en el mecanismo de empaquetamiento jar de Java y en cómo funcionan los ClassLoader de Java
- Se puede hacer a mano utilizando el programa jar de java
- El módulo debería tener la siguiente estructura

directorio

mispaquetes/../../../../binarios...

templates/ ... ficheros xml ...

- El desarrollador debería invocar algo como  

```
jar -cvf mimodulo.jar *
```

desde "directorio"

## Problemas del despliegue a mano

---

- Cuando se empaqueta, los binarios deben estar acompañados de los templates. El javac no transporta los .xml al directorio donde se depositan los binarios.
  - Hay que mover los .xml al directorio de binarios manteniendo la estructura de directorios anterior
- Por ello, dentro del IDK, se prefiere ANT para hacer despliegues de los módulos
- El ANT se puede descargar de <http://ant.apache.org> y se puede ver como el equivalente al programa make

## Problemas del despliegue a mano

---

- Para no tener que aprender a manejar el ant desde cero, se puede utilizar como base el fichero build.xml que contiene ejemplos de configuraciones para compilar y desplegar módulos



## Inicio del build.small.xml

- Comienza definiendo un conjunto de propiedades. Para un nuevo módulo, sólo habría que modificar la propiedad modhtmldoc

```
<project name="ingenias" default="runide" basedir=".">
<property name="modhtmldoc"
  location="modules/srhtmldoc" />
<property name="build" location="classes" />
<property name="temp" location="tmp" />
<property name="genlib" location="genlib" />
<property name="specfile" location=""/>
```

## Tareas en el build.small.xml

- El inicio de la tarea modhtmldoc consiste en definir directorios y lanzar la compilación

```
<target name="modhtmldoc">
<delete dir="${temp}"/>
<mkdir dir="${temp}/templates" />
<depend srcdir="${modhtmldoc.dir}" destdir="${temp.dir}"
  cache="depcache"><include name="**/*.java" /></depend>
<javac compiler="modern" depend="true" destdir="${temp}"
  debug="true">
  <src path="${modhtmldoc}" />
  <classpath>
    <pathelement path="${build}" />
    <fileset dir="lib">
      <include name="**/*.jar" />
    </fileset>
  </classpath>
</javac>
```

## Tareas en el build.small.xml

- El final de la tarea modhtmldoc consiste en copiar las plantillas al directorio de binarios, invocar al jar para crear el fichero de despliegue y mover este fichero al directorio de despliegue

```
<copy todir="${temp}/templates">
  <fileset dir="${modhtmldoc}/templates"></fileset>
</copy>
<jar jarfile="${modhtmldoc}/modhtmldoc.jar"
  basedir="${temp}" />
<delete file="${moddeploy}/modhtmldoc.jar" />
<move file="${modhtmldoc}/modhtmldoc.jar"
  toDir="${moddeploy}" />
</target>
</project>
```

## Desplegando el módulo

- Para lanzar la compilación y despliegue hay que ejecutar el comando  
ant -f build.small.xml modhtmldoc
- Si en vez del fichero build.small.xml se le cambia el nombre a build.xml, no hace falta especificar el fichero, bastaría con  
ant modhtmldoc
- Por supuesto, habría que cambiar modhtmldoc por el nombre de la tarea en cuestión, si se hubiera modificado
- O bien cambiar únicamente el valor asociado a modhtmldoc declarado al principio del build.xml

## ¿Y el IDK?

- Si el despliegue se hace con el editor en marcha, el editor actualizará automáticamente la versión del módulo
- Además, mostrará un mensaje en la sección de mensajes de los módulos indicando la carga del nuevo módulo
- Véase en la figura que el mismo módulo se carga varias veces, tantas como despliegues se hagan

```
[03:28] Loading model juul organization with the new organizational {
[03:28] Project loaded successfully
[03:28] Added new module with name "HTML Document generator"
[03:28] Added new module with name "HTML Document generator"
[03:29] Added new module with name "HTML Document generator"
```

## Y ahora que se escribir módulos ...

- Visto el ejemplo de generación de HTML ya se está preparado para un desarrollo más complejo como es el desarrollo de SMA
- En los módulos incluidos en el IDK se puede
  - Generar SMA basados en JADE
  - Generar simulaciones de los flujos de trabajo utilizando servlets
- El caso es que con estos módulos podemos conectar los diagramas de INGENIAS con cualquier plataforma
- Además resolvemos una vieja pregunta de forma sencilla: ¿es la especificación correcta?
  - ➔ si con ella se puede rellenar las plantillas y además se respetan las restricciones semánticas de la misma, la respuesta es que sí

## Generación de código

