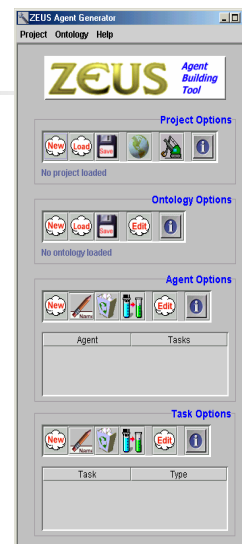


Introducción a ZEUS

Curso Doctorado
Sistemas Multi-agente

Introducción

- Zeus es una herramienta de desarrollo de SMA.





Introducción

- Está constituido fundamentalmente por 3 grupos funcionales:
 - Biblioteca de componentes de agentes
 - Programas de construcción
 - Agentes de utilidad



Introducción

- Biblioteca de componentes de agentes
 - Conjunto de clases implementadas reutilizables
 - Cubren necesidades básicas de un agente: comunicación, planificación, coordinación, ...



Introducción

- Programas de construcción
 - Conjunto de herramientas para desarrollar agentes
 - Incorpora:
 - Un editor de ontologías
 - Un editor de hechos
 - Un editor de agentes
 - Un editor de organización y coordinación
 - Un editor de tareas

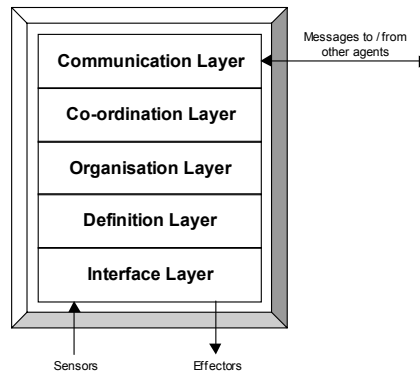


Introducción

- Agentes de utilidad
 - Servidor de nombres
 - DF
 - Visualizador, para operaciones de depuración y monitorización
 - A nivel de sociedad
 - De forma individual para cada agente
 - Herramientas de generación de informes y estadísticas

Guía de desarrollo

- La creación de un agente en Zeus se realiza por etapas, siguiendo las capas de las que consta



Guía de desarrollo

El orden de los pasos a seguir es el siguiente:

Paso 1: Creación de la Ontología

Antes de implementar agentes se debe definir la ontología, conocimiento declarativo que representan los conceptos del dominio de aplicación.

La herramienta empleada es el ZEUS Ontology Editor.

También se da la posibilidad de importar una ontología ya existente.



Guía de desarrollo

Paso 2: Creación del Agente

El agente es configurado para cumplir sus responsabilidades específicas. Este proceso se realiza con el ZEUS Agent Editor e incluye cuatro subfases (aunque depende de la naturaleza del agente), éstas son:

- Agent Definition – se especifican sus tareas, recursos iniciales y habilidades de planificación
- Task Description - se especifican atributos de las actividades del agente
- Agent Organisation – se especifica el contexto social de cada agente
- Agent Co-ordination – cada agente es equipado con las habilidades sociales para su interacción



Guía de desarrollo

Paso 3: Configuración de los Agentes de Utilidad

Este paso define los atributos de los agentes de utilidad, que proveen la infraestructura de soporte de la sociedad de agentes.

Esta información se introduce por medio del Code Generation Editor.

Paso 4: Configuración de las Tareas del Agente

Establece los parámetros de las tareas especificadas: host, recursos externos, programas a enlazar (GUI).



Guía de desarrollo

Paso 5: Implementación del Agente

En este paso se invoca a la función Code Generator, de esta forma el código se genera automáticamente.

Esto hace que el programador deba centrarse en la implementación de:

- Tareas
- Recursos externos
- Programas (interfaces)
- Nuevas estrategias de interacción

Cuando esta fase termina la aplicación está preparada para su ejecución.



Aplicación

Sencillo ejemplo donde disponemos de 2 agentes:

- **InfoSupplier** – proporciona una cadena.
- **Displayer** – visualiza una cadena en una ventana MS-DOS cuando el usuario la requiere.

Y los siguientes hechos:

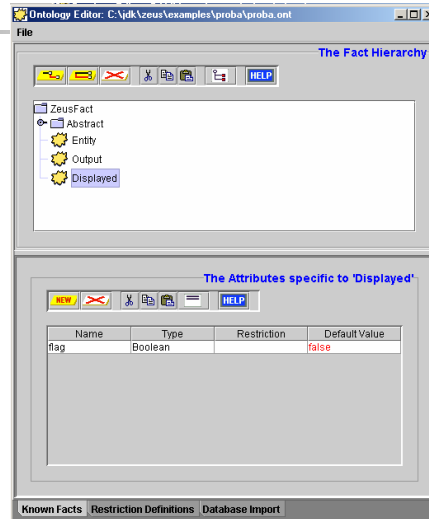
- **output** contiene el texto a visualizar
- **displayed** es un booleano que está a *true* si se ha visualizado la cadena

Aplicación

Definición de la ontología:

Definimos dos hechos:

- **output**
- **displayed**



Aplicación

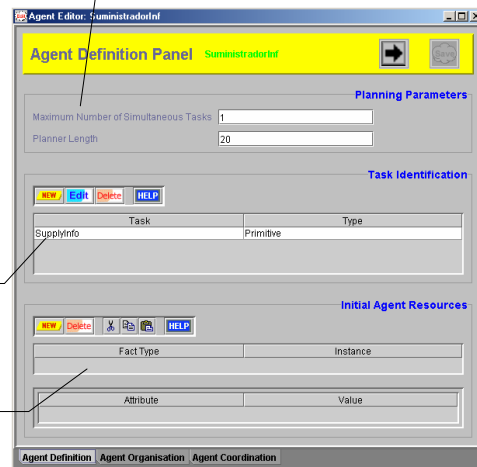
Definición de los agentes:

InfoSupplier
Configuración

2. Identificación de tareas:
SupplyInfo

3. Recursos iniciales

1. Parámetros de Configuración



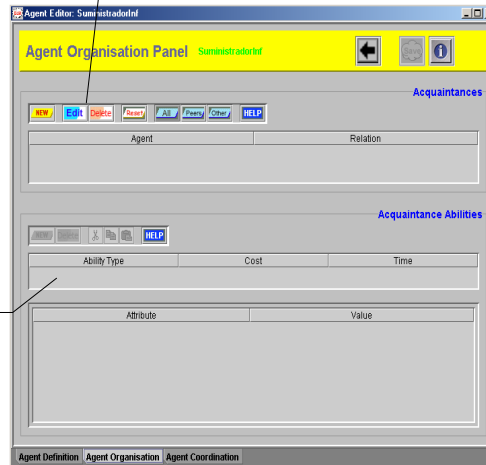
Aplicación

Definición de los agentes:

InfoSupplier
Organización

2. Habilidades frente a esos agentes

1. Posición frente a otros agentes



Aplicación

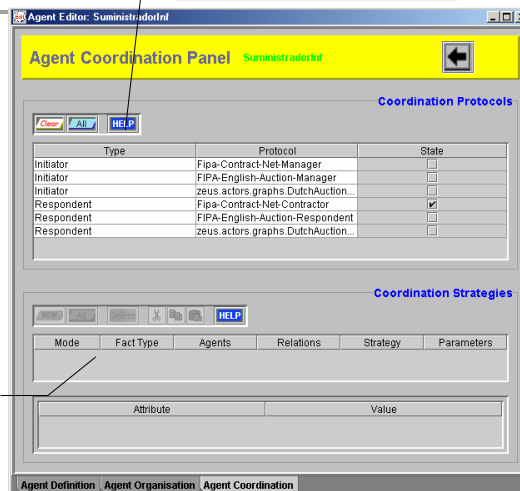
Definición de los agentes:

InfoSupplier
Coordinación

- Respondedor en el contract-net protocol.
- Estrategia por defecto

2. Definir estrategias en las interacciones

1. Definir protocolos que emplea el agente



Aplicación

Definición de los agentes:

Displayer

Identificación de tareas:
DisplayInfo



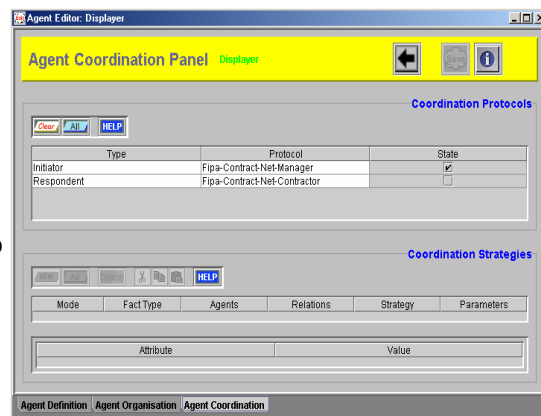
Aplicación

Definición de los agentes:

Displayer

Coordinación

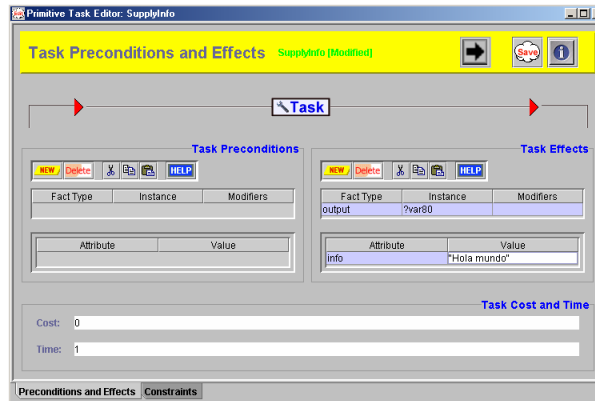
- Iniciador en el contract-net protocol.
- Estrategia por defecto



Aplicación

Definición de las tareas:

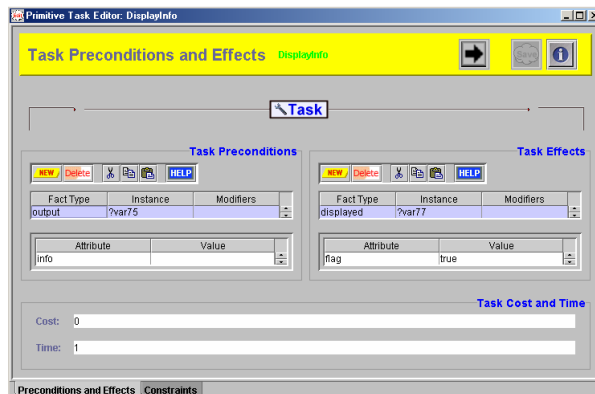
SupplyInfo



Aplicación

Definición de las tareas:

DisplayInfo



Aplicación

Implementación:

La implementación conlleva dos etapas.

- La primera es generar automáticamente parte del código y los scripts empleando el generador de agentes.
- El segundo paso consiste en escribir el código en Java de las tareas y GUI externos.

Aplicación

Generación del código

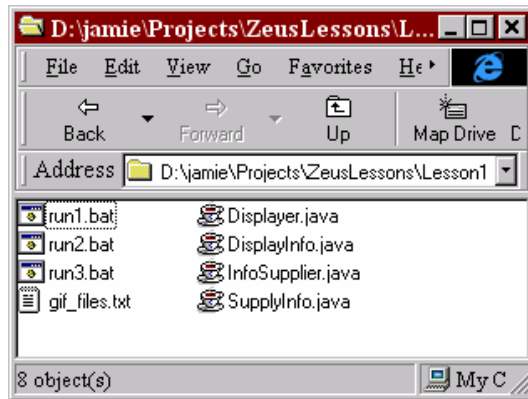


- **Tasks Tab**
 - Especificar un programa externo "DisplayInfoExternal" para la tarea DisplayInfo
- **Task Agents Tab**
 - Marcando la etiqueta en cada agente se asegura la existencia de una ventana de monitorización de cada agente. Además se especifica un programa externo "DisplayerGui" para el agente Displayer
- **Utility Agents Tab**
 - Comprobar direcciones IP y aceptar resto de opciones
- **Generation Plan Tab**
 - Seleccionar directorio y tipo de sistema

Aplicación

Generación del código

GENERATE



Aplicación

Implementando los programas externos: DisplayerGui

■ Crear la ventana

```
/** Constructor sets up the frames and add an ActionListener to the button*/
public DisplayerGui(){
    button = new Button("Assert Goal");
    add(button);
    setSize(290,155);
    setVisible(true);
    button.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent event){
            Object object = event.getSource();
            if (object ==button)
                assertGoal();
        }
    });
}
```



Aplicación

Implementando los programas externos: DisplayerGui

■ Otras funciones

This is called as the agent is created. Here all that happens is that AgentContext is stored

```
public void exec(AgentContext agentContext){
    agent = agentContext;
}
```

This method does all the work. It creates a new displayed fact, sets the value of the flag attribute to true, creates a new goal using this fact and asserts it into the agents coordination engine.

```
public void assertGoal(){
    Fact fact = agent.OntologyDb().getFact(Fact.FACT,"displayed");
    fact.setValue("flag","true");
    Goal g = new Goal(agent.newId("goal"), fact,
        (int)(agent.now()+6, 0.1,agent.whoami()),(double)(agent.now()+3));
    agent.Engine().achieve(g);
}
```



Aplicación

Implementando los programas externos: DisplayInfoExternal

```
public class DisplayInfoExternal implements TaskExternal{
    // To implement the TaskExternal we must implement the exec method
    public void exec(TaskContext tc){
        // The taskContext's inputArgs is an array Fact arrays.
        // There is only one input so we want the first,
        // i.e. get the first precondition which is an outputFact
        Fact[] outputFact = tc.getInputArgs()[0];
        // We then get the outputFact
        // Which is an array. Since there is only one fact it is the first one
        // We print the value of the info attribute.
        System.out.println("*****");
        System.out.println(""+outputFact[0].getValue("info"));
        System.out.println("*****");
    }
}
```

Aplicación

Implementando los programas externos:

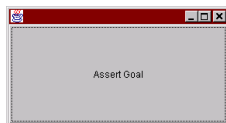
- **Compilar los programas**

```
javac DisplayerGui
```

```
javac DisplayInfoExternal
```

Aplicación

Ejecutando el sistema.



```
C:\WINNT\system32\java.exe
Input: Fact01
<type output
  :id var_57
  :modifiers 0
  :attributes <<info "Hello World">>
>
*****
"Hello World"
*****
-Expected Output-
<type displayed
  :id var_36
  :modifiers 1
  :attributes <<flag true>>
>
-Output-
<type displayed
  :id var_36
  :modifiers 0
  :attributes <<flag true>>
>
```