Primera Práctica

Comunicación entre Agentes

La finalidad de la práctica es crear diferentes agentes con objetivos definidos. Los agentes deben localizar aquellos que puedan proporcionarles el servicio que esperan, independientemente del agente y el momento en el que puedan estar disponibles.

Los agentes que se presentan en el guión, con la descripción de sus tareas, son los siguientes:

Agente Consola

Este agente agente deberá registrarse en las páginas amarillas de la siguiente forma:

- Tipo de Servicio: “GUI”
- Nombre del Servicio: “Consola”

El agente tendrá las siguientes tareas:

1. Recepción de mensajes.
    a. Esta es una tarea cíclica que deberá explorar el buzón de mensajes del tipo ACLMessage.INFORM. Si no hay deberá bloquearse.
    b. Cuando reciba el mensaje deberá almacenar el agente que lo envía y su contenido.
    c. Añadir una tarea para presentar el mensaje
2. Presentación de mensaje.
    a. Esta es una tarea que se ejecuta una vez.
    b. Recogerá el mensaje que se quiere presentar
        i. Creará una ventana para los mensajes del agente, si no se ha creado ya antes. Es decir, habrá una ventana para cada agente que se comunique con la consola.
        ii. La ventana tendrá como título “Consola + nombre del agente” y un área de texto donde se presentan los mensajes que le envíe el agente.

Mejoras

Confirmar que el mensaje ha sido recibido.

Agente Formulario

Este agente tendrá asociada una ventana donde se presentará en el título el nombre del agente. Además en la ventana nos debe permitir recoger dos coordenadas X,Y. Por último en la ventana habrá un botón enviar.

El agente tendrá las siguientes tareas:

1. Localización de agentes
    a. Es una tarea que se repetirá cada 5 segundos.
    b. Debe localizar en las páginas amarillas agentes que proporcionen el servicio “Consola” y el servicio “Operación”.
2. Envío Operación
    a. Es una tarea que se ejecutará una vez, cada vez que sea solicitada por la interfaz.
    b. Enviará un mensaje del tipo ACLMessage.INFORM a todos los agentes que proporcionan el servicio “Operación”. El contenido del mensaje serán las dos coordenadas X,Y.
3. Envío Consola
    a. Es una tarea que se ejecutará cada 10 segundos.
    b. Si no hay una consola disponible para poder enviar el mensaje se esperará a otra ejecución de la tarea.
    c. Enviará un mensaje del tipo ACLMessage.INFORM a un agente que proporciona el servicio “Consola”. El contenido del mensaje indicará con cuantos agentes “Operación” se ha puesto en comunicación con la tarea Envío Operación.

Mejoras

Que se tenga constancia que los mensajes han sido procesados por los agentes a los que se les envió.

Agente Operación

Este es un agente que realizará una operación matemática (suma, resta, multiplicación) de forma aleatoria cuando reciba los parámetros para ello.

El agente tendrá las siguientes tareas:

1. Localización de agentes
    a. Es una tarea que se repetirá cada 5 segundos.
    b. Debe localizar los agentes que proporcionan el servicio “Consola”.
2. Recepción de mensaje
    a. Es una tarea cíclica que explorará el buzón para mensajes del tipo ACLMessage.INFORM. Si no hay deberá bloquearse.
    b. Deberá almacenarse la información relativa al agente que envió el mensaje y el contenido del mensaje.
    c. Añadir una tarea de Cálculo.
3. Cálculo
    a. Es una tarea de que se realiza una vez.
    b. Se operará sobre las coordenadas X,Y recibidas con una de las operaciones. La selección de la operación será aleatoria.
    c. Se almacenará el resultado con el agente que lo pidió.
4. Envío Consola
    a. Es una tarea que se ejecutará cada 10 segundos.
    b. Si no hay una consola disponible para poder enviar el mensaje se esperará a otra ejecución de la tarea.
    c. Enviará un mensaje del tipo ACLMessage.INFORM a un agente que proporciona el servicio “Consola”. El contenido del mensaje indicará el agente que solicitó la Operación, la operación realizada y el resultado.

Mejoras

Confirmar que el mensaje ha sido recibido.

Práctica a realizar

Hacer que las siguientes tareas sean generales y estén incluidas dentro de un paquete llamado `tareas`:

- Una tarea que permita buscar en las páginas amarillas la localización de agentes que provean un servicio especificado. Esta tarea deberá sustituir cualquier tarea propia que tenga el agente para un fin similar.
- Una tarea que permita a un agente enviar mensajes al `AgenteConsola` de la plataforma. Esta tarea deberá sustituir cualquier tarea propia que tenga el agente para un fin similar.

Esta es la descripción de los agentes que compondrán el desarrollo de la práctica. Para ello vamos a necesitar, además de los agentes, las siguientes clases:

- Para la interfaz gráfica:
  - `ConsolaJFrame`: Que nos mostrará los mensajes de cada uno de los agentes. Se creará una para cada agente presente.
  - `FormularioJFrame`: Estará asociado al `AgenteFormulario` para la recogida y envío de datos.
  - `OkCancelDialog`: Cuadro de diálogo para la finalización de los agentes.

- Para el desarrollo de la práctica:
  - `MensajeConsola`: Será la forma en que se presentarán los mensajes por `ConsolaJFrame`. Esta clase contendrá el agente que envía el mensaje y los métodos de acceso apropiados. Además se sobreescribirá el método `toString()` para dar una representación imprimible al mensaje en la consola.
  - `Punto2D`: Clase que se utilizará para el envío de información al `AgenteOperación` para la realización de sus tareas asociadas. Tiene los métodos de acceso necesarios para acceder a los atributos. Además se sobreescribirá el método `toString()` para dar una representación imprimible.

De esta forma el proyecto NetBeans tendrá la siguiente estructura:

- PrimeraPracticaAgentes
  - GUI
    - `ConsolaJFrame`
    - `FormularioJFrame`
    - `OkCancelDialog`
  - agentes
    - `AgenteConsola`
    - `AgenteFormulario`
    - `AgenteOperacion`
  - utilidad
    - `MensajeConsola`
    - `Punto2D`

Antes de presentar las clases que representan a los agentes se describirán las características de las clases que representan la interfaz para los agentes.

Clase `ConsolaJFrame`

Esta clase será la interfaz que se creará cuando un mensaje sea enviado al `AgenteConsola` por un agente. Se creará para cada agente que se comunique a la consola. No será único para todos los agentes.

Tendrá un atributo que será el agente que tiene asociado y que se mostrará como título de la ventana y así sea visualmente identificable qué agente está mostrando el mensaje. Como elemento visual se utilizará un `JTextArea` para mostrar los mensaje.

Para mostrar un mensaje se necesita un método que reciba un objeto de la clase `MensajeConsola` y comprobará si la ventana es visible, si no la pone visible, y añade el mensaje a los anteriormente mostrados.

```
public void presentarSalida (MensajeConsola mensaje) {
    if (isVisible() != true) {
        setVisible(true);
    }
    salida.append(mensaje.toString());
}
```

Cerrar la venta solo la cambia a no visible, no termina la ejecución del `AgenteConsola`. El `AgenteConsola` creará una ventana para cada uno de los agente que le envie un mensaje, por eso no se puede terminar la ejecución del `AgenteConsola` por cerrar una de las ventanas que representan la consola.

```
private void formWindowClosing(java.awt.event.WindowEvent evt) {
    // TODO add your handling code here:
    setVisible(false);
}
```

Clase `OkCancelDialog`

Es una clase que representa un cuadro de diálogo modal para terminar la ejecución de un agente que tenga asociada una interfaz de usuario.

En la ventana se mostrará el agente que se pretende finalizar como título de la ventana y dos botones, uno de finalización y otro de cancelación.

No está asociado a ningún agente en concreto y por eso el atributo tiene que ser de la clase `Agent` para ello.

Esta clase se puede utilizar en proyectos posteriores donde se tenga una interfaz cuya finalización implique la finalización del agente asociado.

Clase `FormularioJFrame`

Esta clase representará la interfaz para el `AgenteFormulario` y la finalización de la interfaz implica la finalización del agente.

Los atributos de la clase serán el `AgenteFormulario` que está asociado a la interfaz y una referencia a un objeto de la clase `OkCancelDialog` para la finalización del `AgenteFormulario` asociado.

Cuando se cierre la ventana se invocará el método apropiado que crea el cuadro de diálogo modal y se mostrará en pantalla.

```
    private void formWindowClosing(java.awt.event.WindowEvent evt) {
        // TODO add your handling code here:
        finalizacion = new OkCancelDialog(this, true, myAgent);
        finalizacion.setVisible(true);
    }
```

La interfaz muestra la información necesaria para que se recoja la información de un `Punto2D` y un botón para enviar esa información al `AgenteConsola`. El botón de envío no siempre estará activo, si el `AgenteFormulario` no tiene `AgentesOperación` a los que enviar la información no estará permitido que el usuario pueda enviarla. Para ello será necesario crear un método público para que el `AgenteFormulario` pueda activar o desactivar el botón si hay o no hay agentes a los que les pueda enviar la información.

Cuando se pulse el botón enviar se tiene que enviar la información recogida en la interfaz al `AgenteFormulario` asociado.

```
    private void enviarActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        Punto2D punto;

        punto = new Punto2D();

        punto.setX(Double.parseDouble(coorX.getText()));
        punto.setY(Double.parseDouble(coorY.getText()));

        myAgent.enviarPunto2D(punto);
    }
```

Para poder hacerlo es necesario que en la clase `AgenteFormulario` se disponga de un método público para el envío de la información que se ha recogido en la interfaz.

Una vez que se han presentado las clases que representan a la interfaz de los agentes se mostrarán las tres clases que definirán cada uno de los agentes necesarios para la resolución del problema que se presentó al principio del guión.

Clase `AgenteConsola`

Como el agente tendrá que tratar los mensajes que le envíen el resto de los agentes, tendrá los siguientes atributos:

- Una lista que contendrá los diferentes objetos `ConsolaJFrame` donde se mostrarán los mensajes de los agentes.
- Una lista de los mensajes que estén pendientes de presentar en la consola.

Las tareas que debe realizar el agente serán las siguientes:

- `TareaRecepcionMensajes`: Tarea cíclica que tratará los mensajes que un agente quiera presentar en la consola. Consistirá en crear un objeto `MensajeConsola` con el agente que ha enviado la información y el contenido del mensaje. Se guardará en la lista de los mensajes pendientes y se añadirá una tarea para que se presente en la ventana correspondiente. Si no hay mensajes pendientes que tratar la tarea se bloqueará.
- `TareaPresentarMensaje`: Tarea que se solo se ejecutará una vez. Tomará el primer mensaje pendiente y buscará la consola asociada al agente que envía el mensaje. Si aún no tiene una consola la creará. Por último enviará la información a la consola.

Para que el agente sea localizable por el resto de agentes de la plataforma deberá registrarse en las páginas amarillas cuando se inicia con la el nombre del servicio “**Consola**” que será por el que lo reconocerán el resto de agentes.

A la finalización del agente deberá eliminar el registro de las páginas amarillas y liberar los objetos `ConsolaJFrame` que tenga asociados.

Modificación propuesta

Confirmar que el mensaje ha sido recibido.

Clase `AgenteFormulario`

Los atributos del agente serán los siguientes:

- Un objeto de la clase `FormularioJFrame`: Donde el usuario pasará los datos a los `AgenteOperacion` que hayan sido localizados.
- Un array para almacenar los agentes del tipo `AgenteConsola` que haya presentes en la plataforma.
- Un array para almacenar los agentes del tipo `AgenteOperacion` que haya presentes en la plataforma.
- Una lista de mensajes pendientes para su envío a la consola. Lo que se almacenará será el contenido del mensaje que se quiere enviar a la consola.

Las tareas que deberá realizar este agente son las siguientes:

- `TareaBuscarAgentes`: Es una tarea que se repetirá cíclicamente cada cierto tiempo. Se buscarán en la plataforma los agentes del tipo `AgenteConsola` y `AgenteOperacion`. Se almacenarán para su posterior uso en el resto de tareas del agente. Si no se localiza ningún `AgenteOperacion` se desactiva el botón de envío de la interfaz, en caso contrario se activará.
- `TareaEnvioConsola`: Es una tarea que se repetirá cíclicamente cada cierto tiempo. Esta tarea enviará al primer `AgenteConsola` el primer mensaje que tenga pendiente para que se muestre. Si no hay consolas disponibles o no hay pensajes pendientes no deberá realizar ningún tipo de operación. Pero se ha construido para que se pueda realizar las operaciones que se considere oportunas en otro tipo de circunstancias.
- `TareaEnvioOperacion`: Es una tarea que se realizará una vez. Se enviará un mensaje a todos los agentes del tipo `AgenteOperacion` que se hayan localizado con la información de un objeto `Punto2D`. Para finalizar se añadirá un mensaje pendiente para la consola indicando que se ha realizado el envío.

Como en la interfaz del agente hay un botón, para que se le envíe el `Punto2D` a los agentes del tipo `AgenteOperacion`, deberemos crear un método público para que se añada la `TareaEnvioOperacion` que será la encargada de completar esta tarea.

```
public void enviarPunto2D(Punto2D punto) {
    addBehaviour(new TareaEnvioOperacion(punto));
}
```

En la finalización del agente no debemos olvidarnos de liberar el objeto interfaz asociado.

Modificación propuesta

Que se tenga constancia que los mensajes han sido procesados por los agentes a los que se les envió.

Clase `AgenteOperacion`

Los atributos de la clase serán los siguientes:

- Un array para almacenar los agentes del tipo `AgenteConsola` que haya presentes en la plataforma.
- Una lista de mensajes pendientes para su envío a la consola. Lo que se almacenará será el contenido del mensaje que se quiere enviar a la consola.
- Una lista donde se almacenarán los objetos `Punto2D` para la realización de las operaciones pendientes del agente.
- Una semilla de inicialización para la selección de la operación que realizará el agente.

Las tareas que deberá completar el agente son las siguientes:

- `TareaEnvioConsola`: Es una tarea que se repetirá cíclicamente cada cierto tiempo. Esta tarea enviará al primer `AgenteConsola` el primer mensaje que tenga pendiente para que se muestre. Si no hay consolas disponibles o no hay pensajes pendientes no deberá realizar ningún tipo de operación. Pero se ha construido para que se pueda realizar las operaciones que se considere oportunas en otro tipo de circunstancias.
- `TareaBuscarConsola`: Es una tarea que se repetirá cíclicamente cada cierto tiempo. Se buscarán en la plataforma los agentes del tipo `AgenteConsola`. Se almacenarán para su posterior uso en el resto de tareas del agente.
- `TareaRecepcionOperacion`: Es una tarea cíclica que comprobará la recepción de un mensaje que contendrá la información de la operación que debe realizar el agente. La tarea recoge esa información y añadirá un objeto `Punto2D` a la lista de operaciones pendientes del agente. Para finalizar añade una `TareaRealizarOperacion` que será la encargada de llevarlo a cabo. Si no hay mensajes pendientes que tratar se bloqueará la tarea.
- `TareaRealizarOperacion`: es una tarea que se ejecuta una vez. La tarea recogerá un `Punto2D` y realizará la operación correspondiente. Para finalizar almacenará la información en la lista de mensajes pendientes para enviar a la consola.

La operación que tiene que realizar el agente se ha implementado como un método privado donde seleccionará aleatoriamente una de las operaciones disponibles en el agente y devolverá el resultado como el contenido del mensaje que será almacenado en la lista de mensajes pendientes de envío a la consola.

```
private String operacion (Punto2D punto) {
    double resultado;

    //Se realiza una operación elegida de forma aleatoria
    int i = rnd.nextInt(4);

    switch (i) {
        case 0:
            //Suma
            resultado = punto.getX() + punto.getY();
            return "Se ha realizado la suma de " + punto
                    + "\ncon el resultado: "+ resultado;
        case 1:
            //Resta
            resultado = punto.getX() - punto.getY();
            return "Se ha realizado la resta de " + punto
                    + "\ncon el resultado: "+ resultado;
        default:
            //Multiplicación
            resultado = punto.getX() * punto.getY();
            return "Se ha realizado la multiplicación de " + punto
                    + "\ncon el resultado: "+ resultado;
    }

}
```

Modificación propuesta

Confirmar que el mensaje ha sido recibido.