Practica 6 de Programación Concurrente y Distribuída

Miguel Tellería, Laura Barros, J.M. Drake

30 Nov, 1 Dic 2011

Resumen

En esta práctica el alumno aprende a diseñar aplicaciones distribuídas bidireccionales definiendo una interfaz por cada dirección de llamada, pasando argumentos de clase serializable y usando una de las interfaces para transferir el stub de la otra.

1. Introducción

Planteamos un nuevo particionamiento de la aplicación SmartHunter utilizada en la práctica anterior de forma que las particiones se comporten simulatáneamente como clientes y servidores, es decir ofrezcan métodos para ser invocados por otros e invoquen métodos ofrecidos remotamente.

1.1. Modificaciones al código base

El código base sigue basado en la versión monoprocesador concurrente. Sin embargo se han hecho algunas pequeñas modificaciones respecto al código base de la práctica anterior.

- Por un lado los objetos de la clase `Missil` ya no hacen un polling a su `MissileBattery` para pedir la Id. En lugar de ello, el objeto `Missil` ofrece el método `asignaJet()` que se le ha de llamar para que empiece a buscar su objetivo.
- Se ha eliminado la descomposición en parámetros `x`, `y`, `z` de los argumentos de los métodos de posición y dirección para las clases `MissileBattery`, `Missil` y `SpeedFalcon` (interfaz `MissilHardware`). Ahora las coordenadas se pasan de una vez por medio de la clase `Posicion` (o `CosenoDirector` para el caso de la dirección del `SpeedFalcon`).

1.2. Diagrama de colaboración

Con estas modificaciones el diagrama de colaboración de la práctica 5 tiene ahora *relaciones bidireccionales* como se muestra en la Figura 1.

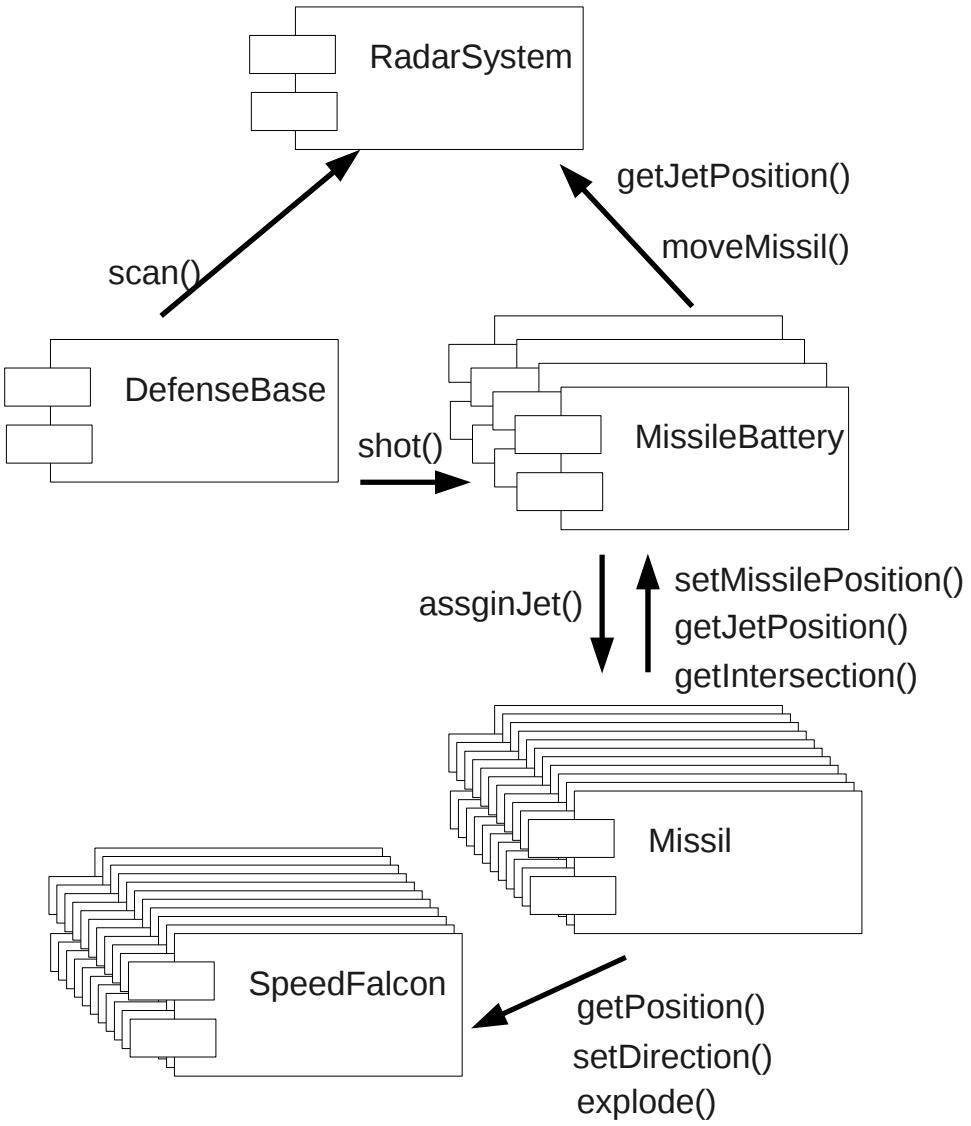

Figura 1: Diagrama colaboracion simplificado del nuevo código base

2. Práctica del Miércoles

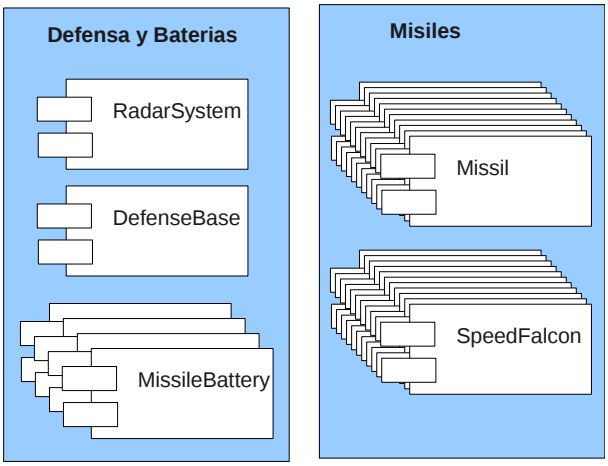

Figura 2: Partición del Miércoles

En el grupo del miércoles se ha de particionar el sistema de la manera mostrada en la Figura 2:

- Particion **Defensa y Baterias** que incluye las clases `AirRaid` (con su interfaz `RadarSystem`), `DefenseSystem` y las cuatro `MissilBatery`.
- Partición **Misiles** que incluye **todos** los `Missil` con sus correspondientes `SpeedFalcon`.

Si se observa el diagrama de colaboración, hay llamadas en ambos sentidos de una partición a la otra.

2.1. Parámetros de interfaz del miércoles

La partición *Defensa y baterias* se inscribe en el registro RMI con 4 interfaces (una por cada batería) con los nombres RMI y escuchando en los puertos TCP que se listan en la tabla:

Cardinalidad	Nombre RMI	Puerto TCP
Norte	`N`	13100
Este	`E`	13101
Oeste	`O`	13102
Sur	`S`	13103

La partición *Misiles* instancia cinco misiles de cada cardinalidad estáticamente ofreciendo una interfaz RMI igual por cada misil, exportando los stubs a los puertos de la manera siguiente:

- Misiles para batería Norte: escuchando en los puertos 14001, 14002, 14003, ...
- Misiles para batería Este: escuchando en los puertos 14101, 14102, 14103, ...
- Misiles para batería Sur: escuchando en los puertos 14201, 14202, 14203, ...
- Misiles para batería Oeste: escuchando en los puertos 14301, 14302, 14303, ...

Esta partición (*Misiles*) **no se ha de registrar en el rmiRegistry**, sino que los stubs se han de pasar a la otra partición preveyendo un método para ello en la interfaz de la partición *Defensa y baterías*.

3. Practica del Jueves

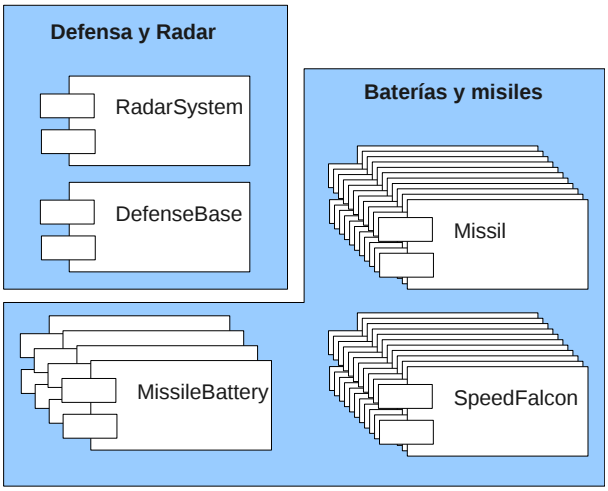

Figura 3: Partición del Jueves

En el grupo del jueves se ha de particionar el sistema de la manera dada en la figura 3:

- Particion **Defensa y Radar** que incluye las clases `AirRaid` (con su interfaz `RadarSystem`) y `DefenseSystem`.
- Partición **Baterias y Misiles** que incluye las 4 baterías, los misiles y los Falcons.

Si se observa el diagrama de colaboración, hay llamadas en ambos sentidos de una partición a la otra.

3.1. Parámetros de interfaz del jueves

La partición *Defensa y radar* se inscribe en el registro RMI un único stub nombre **DefensaRadar** y escucha las peticiones en el **puerto TCP 13000**.

La partición *Baterias y Misiles* exporta **cuatro stubs, uno por cada batería de misiles** escuchando en los siguientes puertos TCP.

Cardinalidad	Puerto TCP
N	14100
E	14101
O	14102
S	14103

Dichos stubs se han de pasar (conjunta o separadamente) a la partición *Defensa y radar* mediante una función de paso, sin registrarse en RMI.

4. Procedimiento sugerido

Para abordar esta práctica se sugiere el siguiente procedimiento.

1. Separar en proyectos eclipse las 2 particiones dejando un tercer proyecto comun (visible desde cada partición) para datos comunes (puertos TCP, interfacesRMI, nombres RMI, clases de argumentos...).
2. Cada una de las 2 particiones tendrá su método main() que instanciará los objetos de su partición.
3. Identificar los métodos que han de ir en cada interfaz, preveyendo un método extra (*función de paso*) en la interfaz que se registra en el RMI registry para que se pase(n) el (o los) stub(s) de la otra.
4. Los argumentos o valores de retorno de los métodos de cada interfaz que no son tipos primitivos requieren ser serializados.
5. La partición que se registra en el rmiRegistry, ha de hacer todo el proceso de exportar su stub y registrarlo en el rmiRegistry. Luego ha de esperar a que la otra partición le pase su(s) stub(s) en la *función de paso* antes de poder llamarlos.
6. La partición que pasa su stub ha de exportarlo también, pero en lugar de registrarse en el rmiRegistry, ha de pasar el stub directamente mediante la función de paso de la interfaz registrada.