Programación Concurrente y Distribuida

Ingeniería Informática
Facultad de Ciencias
Universidad de Cantabria.

Documento:

Practica 5: SmartHunter RMI I

Autor: José M. Drake

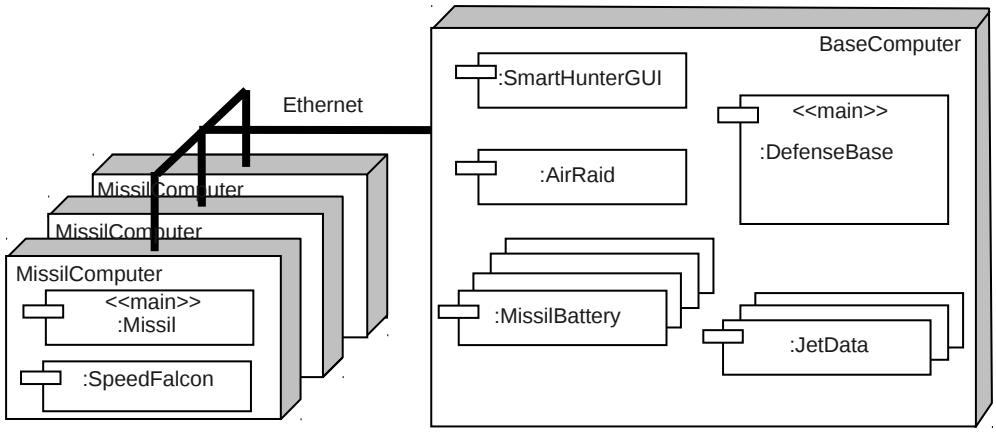
Santander, Noviembre, 2011

Aplicación SmartHunter

Distribución de la aplicación SmartHunter con RMI.

Opción del Miércoles 23-Nov.: Distribución con misiles inteligentes

En este caso se propone la distribución de la aplicación utilizando RMI, de acuerdo con el siguiente plan de despliegue:



Se considera que todo el software del sistema de defensa está instalado en un único computador, mientras que se distribuye en cada misil, tanto el software de control del hardware (SpeedFalcon), como el software de conducción del misil (Missile).

Criterios de diseño

La distribución afecta a cada misil, esto es, los objetos relativos a cada misil Missil y SpeedFalcon, quedan instanciados en el procesador MissileComputer, mientras que las clases de los servicios comunes se instancian en el procesador BaseComputer. En este caso las instancias Missil actúan como clientes, mientras que las clases MissileBattery actúan como servidores.

Hay que resolver la forma en que las instancias Missile y MissileBattery obtienen el puerto a través de la que interactúan, y la posición inicial desde la que se dispara:

- Los objetos de la clase MissileBattery tiene asignados de forma estática los puertos a través de los que atiende:
  - Battery “E” [10000,0] => 3000
  - Battery “N” [0,10000] => 3001
  - Battery “S” [-10000,0] => 3002
  - Battery “O” [0,-10000] => 3003

  Cuando las baterías se instancian, se registran en el rmiRegistry con el nombre de la batería “E”,“N”,”S” o “O”.
- Cuando los programas Missile se ejecutan, reciben como argumento de su main() el nombre de la batería a la que se asigna, esto es “E”,“N”,”S” o “O”, y con él buscan en el rmiRegistry el acceso a la batería. Alternativamente se puede codificar la batería en el main y crear main's distintos para cada cardinalidad.

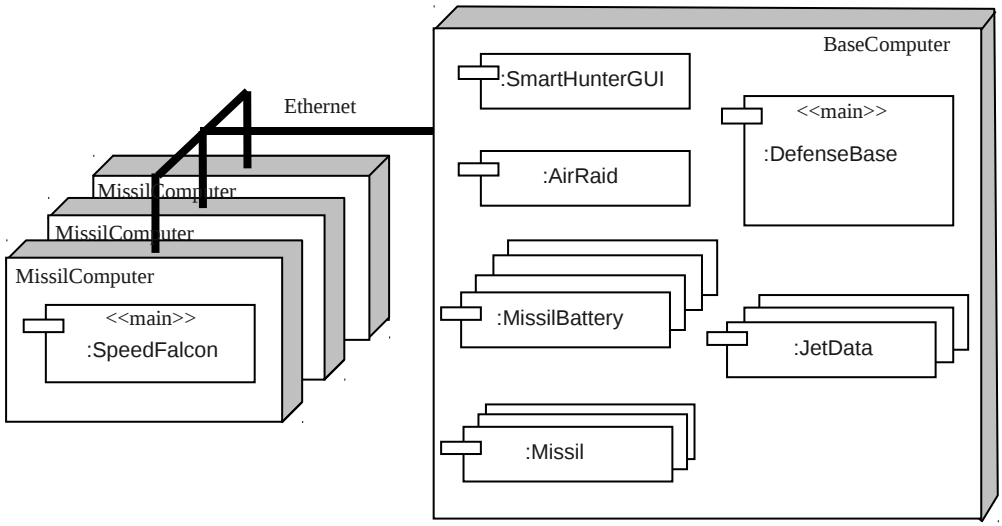
Nota: *En esta práctica sugerimos modificar:*

- *la clase **MissileBattery**: Se debe convertir en un servidor RMI que atienda a los misiles.*
- *La clase **Missile** que se convierte en un programa con su main() y que se constituye en un cliente del servidor ofrecido por las instancias de MissileBattery*
- *El programa principal **SmartHunterSystem** que no debe instanciar los misiles.*

*Las **restantes clases** deberian quedar inalteradas.*

Distribución de la aplicación SmartHunter con RMI.
Opción del Jueves 24-Nov: Distribución con SpeedFalcon tontos.

En este caso se propone la distribución de la aplicación utilizando rmi, de acuerdo con el siguiente plan de despliegue:



Se considera que todo el software del sistema de defensa está instalado en un único computador, mientras que se distribuye en cada misil sólo el software de control el hardware.

Criterios de diseño

La distribución afecta a cada misil, esto es, los objetos de la clase misil son instanciados por la clase MissilBattery sobre el procesador BaseComputer cada vez que sobre ella se ejecuta el método shot(), mientras que los programas SpeedFalcon son instanciadas en los procesadores MissilComputer. Las instancias MissileComputer actúan como servidores, mientras que las instancias Misil actúan como clientes.

Hay que resolver la forma en que las instancias Missile y SpeedFalcon obtienen el puerto a través de la que interactúan, y la posición inicial desde la que se dispara:

- Los Programa de la clase SpeedFalcon se registran en el rmiRegister con nombres compuestos del nombre de la batería y un ordinal, y utilizan un puerto definido a partir de una base propia de la batería en la que se instala:
    - Battery “E” => Puertos 3000,3001,3002, ... y con nombres “E1”,”E2”,”E3”,...
    - Battery “N” => Puertos 3100,3101,3102, ... y con nombres “N1”,”N2”,”N3”,...
    - Battery “S” => Puertos 3200,3201,3202, ... y con nombres “S1”,”S2”,”S3”,...
    - Battery “O” => Puertos 3300,3301,3302, ... y con nombres “O1”,”O2”,”O3”,...

Sin embargo, los SpeedFalcon **no saben** que número está libre. La cardinalidad se la pasamos como argumento de programa (o la codificamos en el main), pero el número de orden (y puerto TCP asociado) lo tienen que conseguir mediante **prueba-error**. Sugerimos que se intenten registrar con un nombre y puerto iniciales y traten la exception `AlreadyBound`. Incrementando el nombre y número de puerto.

- Los objetos Missile cuando son creados por la MissileBattery, reciben en su constructor el nombre de la batería a la que son asignados y un número que la batería incrementa consecutivamente.

  Conociendo el misil su cardinalidad y su número, ha de buscar periodicamente su SpeedFalcon respectivo, durmiéndose un periodo hasta que lo haya encontrado..

Nota: *En esta práctica sugerimos modificar:*

- *La clase MissileBattery que debe crear el objeto Missile en su método shot*
- *La clase **Missile**: Se debe convertir en un cliente RMI que localiza el SpeedFalcon que controla..*
- *La clase **SpeedFalcon** que se convierte en un programa con su main() y que se constituye en un servidor RMI que atiende a su controlador Missile.*
- *El programa principal **SmartHunterSystem** que no debe instanciar los misiles.*

*Las **restantes clases** no deberían quedar inalteradas.*