Auditoría, Calidad y Fiabilidad Informáticas

Universidad Complutense de Madrid Curso 2013/2014

Práctica 4: Arquitectura básica de red

Especifica y analiza en un módulo orientado a objetos una arquitectura de red básica. Para ello:

1. Definición

Ejercicio 1 Define una clase `Nodo` con un atributo de tipo `String` con su dirección IP y otro de tipo `Estado` (que es necesario definir, y que toma valores `inactivo`, `esperando` y `activo`), que indica su estado.

Ejercicio 2 Define una subclase `Centro` de `Nodo` que tiene como atributo una tabla que tiene como clave direcciones IP y como valores los identificadores (de tipo `Oid`) de los respectivos nodos.

- Esta tabla debería estar definida en un módulo funcional con operaciones para insertar una dirección y un identificador (si la dirección ya está se modifica el identificador) y para eliminar una entrada de la tabla dada la dirección.

Ejercicio 3 Define una subclase `Extremo` de `Nodo`, con un atributo `centro` de tipo `Oid` con el identificador del centro. Inicialmente el valor de esta atributo es `null` (que tendrás que definir).

Ejercicio 4 Define un mensaje `info`, que no tiene destinatario y que tiene como argumentos un `String` y un `Oid`.

- Este mensaje lo envían los objetos de tipo `Extremo` inactivos para indicar su dirección y su nombre. Al enviarlo pasa al estado `esperando`.
- Este mensaje es recibido por objetos de tipo `Centro`, y se utiliza para actualizar la tabla.
- El `Centro` pasa de `inactivo` a `activo` en cuanto recibe uno de estos mensajes (nunca entra en estado `esperando`).

Ejercicio 5 Un mensaje `respuesta-info`, que tiene como destinatario un `Oid` y como argumento otro `Oid`.

- Este mensaje lo envía el `Centro` al `Extremo` como respuesta al mensaje `info`.
- Cuando el `Extremo` recibe el mensaje actualiza su atributo `centro` y pasa al estado `activo`.

Ejercicio 6 Define, en un módulo `EJEMPLO`, una configuración inicial con un objeto de tipo `Centro` y tres objetos de tipo `Extremo`, todos ellos inicialmente inactivos, y utiliza el comando `rew` para ejecutarlo.

2. Comportamiento

Ejercicio 7 Define el tipo (`sort`) `CjtoString`, que identifica un conjunto de `String`. Crea las constructoras y los `subsort` necesarios, pero no hace falta que definas funciones para este tipo.

Ejercicio 8 Define un nuevo atributo `recibido`, de tipo `String`, para la clase `Nodo`. Inicialmente este atributo contiene la cadena vacía.

Ejercicio 9 Define un nuevo atributo `amigos`, de tipo `CjtoString`, también en la clase `Nodo`. Este argumento contendrá las IPs de algunos de los otros nodos.

Ejercicio 10 Define un nuevo mensaje `to_:_`, que toma como argumentos 2 elementos de tipo `String`.

Ejercicio 11 Define un nuevo mensaje `to_:_`, que se diferencia del anterior porque en este caso el primer argumento es un `Oid`.

Ejercicio 12 Haz que cualquier nodo pueda usar el primer mensaje para mandar exactamente un mensaje a cada uno de sus amigos (el texto puede ser cualquier `String`; haz que se manden `"hola"`).

Ejercicio 13 Haz que los objetos de tipo `Centro` se encarguen de transformar los mensajes del primer tipo en mensajes del segundo tipo mirando en su tabla. Además, si el mensaje es para el centro lo recibe de la misma manera que se explica abajo para recibir mensajes en general.

Ejercicio 14 Cuando un mensaje del segundo tipo llega a su destinatario (o se encuentra uno del primer tipo dirigido al centro) el objeto concatena el mensaje en su atributo `recibido`.

Ejercicio 15 Define una función `numObjetos` que cuenta el número de objetos en una configuración.

Ejercicio 16 Actualiza el término inicial de la sección anterior para que cada objeto tenga 2 amigos.

Ejercicio 17 Utiliza el comando `search` para comprobar que el número de objetos permanece invariable durante toda la ejecución.

3. Análisis

Ejercicio 18 Define el estado sobre el que demostrarás las propiedades.

Ejercicio 19 Define propiedades para:

- Comprobar si un cierto nodo existe, dada su IP.
- Comprobar si algún nodo tiene como amigo a un cierto nodo (identificado por su `Oid`).
- Comprobar si existe un mensaje para un cierto nodo (identificado por su IP).
- Comprobar si la cantidad de nodos es una cierta cantidad, dada como argumento.
- Comprobar si la cantidad de objetos de tipo `Extremo` es una cierta cantidad, dada como argumento.

Ejercicio 20 Comprueba las siguientes propiedades con el término inicial de la sección anterior:

- La cantidad de nodos no varía.
- Si un nodo existe y otro lo tiene como amigo, le acaba mandando un mensaje.
- Cualquier mensaje acaba desapareciendo.

Ejercicio 21 Explica qué definiciones y qué reglas deberíamos cambiar para que los objetos que reciben un mensaje contesten al objeto que les envió el mensaje. En especial, piensa que quieres que los mensajes se contesten pero que no se entre en un ciclo de respuestas, es decir, si el objeto `o1` manda el mensaje `"hola"` al objeto `o2`, este lo almacenaría y contestaría `"buenas"`. Una vez `o1` recibe este mensaje lo almacena y acaba. Además, sería interesante que no dependa del mensaje enviado. ¿Qué harías para definir una propiedad que diga "los mensajes recibidos son contestados"?