Proyecto Tracker BitTorrent - Entregable 1 [15h - 12% calificación]

Este primer entregable del proyecto tiene como objetivo fundamental el diseño del sistema distribuido. Dicho diseño contempla la definición del modelo arquitectónico y la interfaz gráfica. En concreto, las cuestiones que habrá que definir son las siguientes:

- Diseño del **modelo de arquitectónico de despliegue** en el que se identifiquen claramente las diferentes entidades que componen el sistema distribuido, las responsabilidades de cada una de ellas, los enlaces de comunicaciones y los protocolos/tecnologías de comunicaciones utilizados. Este diseño se materializará en un diagrama de bloques (utilizando notación UML de diagramas de despliegue o algún otro diagrama que sea descriptivo y claro).

- Descripción en texto de la **funcionalidad que implementará cada una de las entidades** que participan en el sistema distribuido: trackers (Master y Slaves) y peers. Esta descripción se realizará en forma de tabla en la que cada una de las filas contendrá un identificador de la funcionalidad, la entidad a la que se asocia, y la descripción de la misma.

- Definición del **esquema de datos de la información que almacenarán los trackers**. Dicha definición contempla tanto la tecnología utilizada para dotar de persistencia a la información, una justificación de dicha elección, y un diccionario de datos en el que se detalle cada entidad de información a almacenar con sus campos (especificando el tamaño y tipo de dato de cada uno de ellos). Recuerda que se almacenará información de los enjambres o swarms cuando el tracker está activo.

- Diseño de los **modelos de interacción entre las diferentes entidades**: tracker-peer y tracker-tracker. Para cada una de las funcionalidades recogidas en la tabla que impliquen el intercambio de mensajes entre más de una entidad, habrá que definir de manera específica cada uno de los mensajes: origen, destino, formato y contenido del mismo. Los mensajes tracker-peer los define el estándar del protocolo bitTorrent[1]. Utilízalos como referencia para la definición de los mensajes que se intercambian entre los trackers. En este caso no hace falta trabajar a nivel de byte puesto que JMS permite la definición de los mensajes en diferentes formatos como, por ejemplo: XML, Objetos Java, String, etc. Es muy importante una buena y detallada definición de los mensajes para agilizar la labor de implementación.

- Definición de los **modelos de fallo** de cada una de las entidades; y de la interacción entre las entidades. La definición de los modelos de fallo debe contemplar los posibles fallos que pueden acontecer en el sistema distribuido identificando la causa (si es posible) y las medidas que se tomarán para hacer frente a dicho fallo. Los modelos de fallo se describirán en texto. Esta descripción podrá ir vinculada a los modelos de interacción (para los fallos que se produzcan durante la interacción entre entidades) o aparecer en un apartado diferente.

- Implementación en Java de la **interfaz gráfica del tracker**; y además, creación de la estructura "base" del código del proyecto completo. Con este objetivo en mente, tenéis que codificar todas las clases relativas a la interfaz gráfica, y además, el esqueleto (con atributos y métodos sin funcionalidad) de toda la estructura base de la aplicación del Tracker.

[1] UDP Tracker Protocol for BitTorrent. URL: http://www.bittorrent.org/beps/bep_0015.html [Fecha de consulta: 07-2016]

Notas acerca de la codificación

Para implementar la aplicación del tracker se recomienda utilizar el patrón de diseño MVC[2], de tal forma que el código relacionado con la funcionalidad de la aplicación quede desligado lo más posible de la interfaz gráfica. A continuación, se detalla alguna pauta para implementar el patrón MVC en el dominio propuesto:

- El **Modelo** hace referencia al software que ofrece la funcionalidad del tracker. En nuestro caso, como tenemos 3 funciones bien diferenciadas, lo más simple sería realizar la siguiente división:
    - **Gestor de Redundancia de Trackers**: implementa la funcionalidad relacionada con la redundancia de las diferentes instancias de los trackers.
    - **Gestor de Datos**: encapsula la funcionalidad relacionada con el almacenamiento de información persistente.
    - **Gestor de Tracket bitTorrent**: implementa la funcionalidad específica del protocolo UDP de un tracker bitTorrent.

  Con esta sencilla estructura, la aplicación se organiza de tal forma que permite la división del trabajo de programación; y además facilita el aislamiento de errores.

- La **Vista** es la "forma" en la que se visualiza la información de la aplicación. En este caso, la Vista la componen todos los elementos de la interfaz gráfica. De cara a organizar un poco la interfaz gráfica, una buena alternativa sería dividir cada una de las pestaña de la ventana principal en una clase independiente que herede de algún contenedor (por ejemplo: JPanel trabajando con swing) de tal forma que cada una de estas clases agrupe elementos de interfaz que estén relacionados entre sí.

- El **Controlador** representa una o varias clases intermediarias entre el Modelo y la Vista que se encargan de capturar la interacción (datos de entrada del cliente) para trasladar peticiones al Modelo y retornar los resultados a la Vista. En este caso, lo más práctico sería definir una clase "controlador" por cada una de las clases que definen la Vista, ya que cada una de ellas requerirá la interacción con una parte concreta del Modelo.

Para acceder al Modelo desde la Vista, cada Controlador tendrá una referencia al/los gestor/es necesario/s. Una cuestión importante del patrón MVC es el hecho de que las interacciones, y por lo tanto el conocimiento entre los diferentes elementos debe seguir la siguiente secuencia:

- Vista → Controlador → Modelo

Esta secuencia determina qué parte tiene acceso (referencia a nivel de programación) a otra:

- La Vista no tiene acceso directo al Modelo.
- El Controlador NO tiene acceso a la Vista.
- El Modelo no tiene acceso ni al Controlador ni a la Vista.

Además del patrón MVC, como hay elementos de la Vista que deben actualizarse cuando se cambie algo en el Modelo; por ejemplo, el estado de conexión de los trackers o los peers. La manera más sencilla de implementar esta funcionalidad sería comunicar directamente el Modelo

[2] Model-View-Controller. Wikipedia. URL: http://en.wikipedia.org/wiki/Model–view–controller [Fecha de consulta: septiembre de 2015]

con la Vista. Pero esta interacción rompe la secuencia que hemos definido más arriba y obliga a que el Modelo conozca a la Vista. Esto implica que el Modelo pasa a depender de la Vista, y por lo tanto, un cambio en la Vista puede afectar al Modelo.

La situación descrita en el párrafo anterior, es la fuente de muchos problemas de diseño que obligan a cambiar el Modelo (donde está la funcionalidad o lógica de negocio de una aplicación) ante cualquier cambio en la Vista. Imaginemos el caso de pasar una aplicación de escritorio a la web; este tipo de cambio, en muchos casos, obliga a reescribir completamente la aplicación, debido a que tanto el Modelo como la Vista están estrechamente relacionados; incluso en situaciones en las que la funcionalidad no se cambie en absoluto.

Para evitar este problema, hay que conseguir que el Modelo se comunique con la Vista, pero sin tener que conocer con exactitud las complejidades de ésta. La manera más sencilla de solucionar este problema es mediante el uso de otro patrón de diseño: el patrón Observer[3].

La esencia del patrón Observer, es el uso de una interface[4], que permite establecer la comunicación entre diferentes entidades, dónde una que posee cierta información cuyos cambios interesan a otras; evitando que dichas entidades se conozcan entre sí (realmente las entidades no conocen el "tipo" concreto de cada una de ellas).

El patrón Observer es un patrón ampliamente utilizado en el ámbito del desarrollo de software, de hecho, Java tiene su propia implementación del patrón Observer por medio de la clase Observable y la interface Observer. En nuestro caso, la implementación del patrón Observer, se traduce en las siguientes cuestiones a tener en cuenta:

- El Modelo tiene la información "cambiante", que interesa a la Vista; porque quiere representarla de forma visual. Por lo tanto, el Modelo tiene que actuar como parte Observable y, proveer la funcionalidad de subscribir/desubscribir Observers así como notificación de los cambios en la información "cambiante".

- A la Vista le interesa la información "cambiante" y tiene que implementar un mecanismo de recepción de dicha información. Para ello se convierte en Observer que se subscribe/desubscribe del Observable, y recibe las notificaciones e los cambios en la información "cambiante".

Además de estos dos patrones de diseño, cada grupo puede proponer alguna otra pauta/patrón de diseño, que será valorada positivamente; pero siendo lo prioritario la implementación de las comunicaciones entre los diferentes elementos del sistema distribuido propuesto.

En esta entrega tenéis que crear todas las clases correspondientes a los patrones MVC y Observer, establecer los enlaces entre Modelo, Contolador y Vista, y validar que la interacción entre las ventanas y los "gestores" funciona correctamente.

Requisitos no funcionales en relación a la codificación

- Tenéis que definir una jerarquía de paquetes que agrupe las clases siguiendo algún criterio.

- El código debe estar libre de errores y "warnings". Debe compilar y ejecutar.

[3] Observer Pattern. Wikipedia. URL: http://en.wikipedia.org/wiki/Observer_pattern [Fecha de consulta: 07/2016]
[4] Interface (computing) - Software Interfaces. Wikipedia. URL: http://en.wikipedia.org/wiki/Interface_(computing)#Software_interfaces [Fecha de consulta: 07/2016]

- La interfaz gráfica debe adaptarse de forma adecuada a diferentes resoluciones de pantalla.
- Los controles de la interfaz gráfica deben tener algún tipo de validación para evitar que el usuario comenta errores al introducir parámetros o datos de funcionamiento de la aplicación.
- La gestión de excepciones debe mostrar mensajes coherentes ante la ocurrencia de errores.

Formato de la entrega

- La entrega se realizará **a través** de la tarea que está preparada para ello en **ALUD**.
- El formato de la entrega debe ser un único archivo comprimido en formato ZIP que contenga un proyecto de eclipse con el nombre SSDD-Tracker-# (siendo '#' el código de grupo que tenéis asignado en ALUD). Dentro de la estructura de carpetas del proyecto habrá una carpeta "doc" con la documentación asociada al entregable. El formato de la documentación debe ser **DOC, DOCX o PDF** para que las correcciones puedan será añadidas a modo de comentarios.

  El documento tiene que tener la identificación clara de los miembros del grupo y su código. Además, debe llevar un encabezado y pie de página (con numeración). Podéis tomar como ejemplo los documentos que describen tanto la práctica como los diferentes entregables.

Sistema de evaluación

- La evaluación de este **primer entregable representa un** 12% de la calificación final de la asignatura.
- Las **cuestiones que se evaluarán** de este entregable son las siguientes:
  - **Definición de todas las cuestiones que se piden**; que en el documento aparezcan todas las cuestiones claramente definidas y detalladas: 7%
  - **Evaluación del código;** las cuestiones que se evaluarán relativas al esqueleto de la aplicación serán: 4%
    - **Diseño y jerarquía de clases**: 2%
    - **Limpieza y estructura del código**: 1%
    - **Estética** de la interfaz gráfica y **usabilidad**[5]: 1%
  - **Formato de la documentación**; que la documentación y los gráficos sean claros y su presentación esté cuidada: 1%
  - Nivel de **detalle y mejoras** al planteamiento inicialmente propuesto; concreciones y detalles que puedan quedar podo explicitados en la descripción general o sugerencias de mejora: 2%. A la hora de plantear mejoras, el grupo debe comprometerse a llevarlas hasta la implementación.
- Además de la calificación total del entregable, para optar al 5% de calificación correspondiente a la competencia genérica de manejo del inglés, tanto éste como el resto de entregable que impliquen la entrega de documentación deben estar escritos en inglés.

[5] Usabilidad. Wikipedia. URL: http://es.wikipedia.org/wiki/Usabilidad [Fecha de consulta: septiembre de 2015]