proyecto

Objetivo

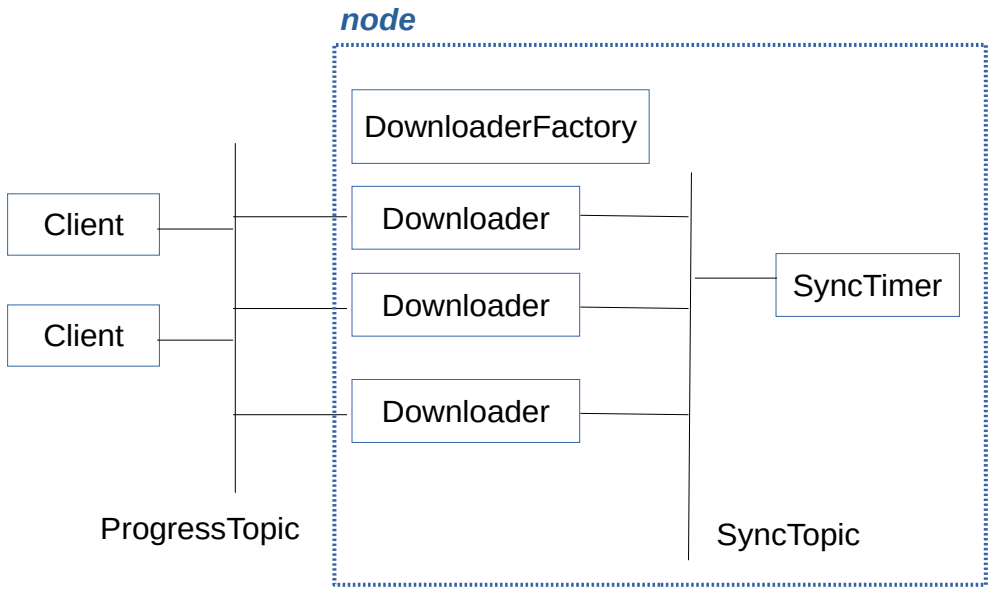
El objetivo principal del proyecto es diseñar un sistema cliente-servidor que permita la extracción de ficheros de audio a partir de la URL de clips de youtube. Este sistema debe ser escalable, permitiendo la creación bajo demanda de nuevos servidores encargados de las tareas de descarga y extracción de los audios. Además, estos servidores deben interconectarse y sincronizarse de forma automática entre sí, proporcionando de esta manera un sistema de alta disponibilidad

La implementación de este proyecto permitirá al alumno trabajar los siguientes aspectos:

- Comunicación asíncrona
- Manejo de canales de eventos
- Despliegue de servidores de forma dinámica
- Gestión de un grid

Arquitectura del proyecto

El proyecto se compone de 3 tipos de componentes: servidores, clientes y canales de eventos, y cuya arquitectura se muestra en la figura.



Servidores de descarga

Por un lado los servidores de descarga deben realizar las siguientes tareas, para lo que implementan la interfaz *DownloadScheduler*:

- recepción de las peticiones de descarga desde los clientes. Esta petición, que indicará la URL del clip de youtube, no retorna valores, sino que la información sobre la descarga se comunicará a través de un canal de eventos.

```
void addDownloadTask(string url);
```

- almacenamiento de los archivos obtenidos tras el proceso de extracción del audio en un directorio local al servidor.
- Método para descargar los ficheros almacenados por el servidor en el cliente.

```
Transfer* get(string song);
```

- Recepción de peticiones de listado de los audios almacenados en el servidor:

```
SongsList getSongsList();
```

- donde *SongsList* está declarado como una secuencia de strings

```
sequence<string> SongsList;

interface DownloadScheduler {
     ["amd"] string addDownloadTask(string url);
     SongsList getSongsList();
     Transfer* get(string song);
};
```

Interfaz Downloader

Los servidores de descarga deben implementar además una segunda interfaz destinada a la sincronización de los propios servidores entre sí (*SyncTopic*). La finalidad es que cada uno de ellos informe al resto de los audios que almacena localmente, con el fin de que los restantes servidores puedan obtener la lista completa de canciones descargador por todos los servidores. Para que los archivos estén disponibles para todos los servidores de descarga, deberá utilizarse una carpeta de descargas compartida por todos ellos.

La solicitud de sincronización se realizará mediante un canal de eventos distinto al utilizado por los clientes (*SyncTopic*), del que los servidores serán subscriptores. El publicador será un componente especial gobernado por un temporizador, que será quien realice las solicitudes.

Clientes

Los clientes deben poder realizar las siguientes tareas:

- solicitar la descarga de una URL correspondiente a un clip de Youtube:
- recibir las notificaciones relativas al proceso de la descarga, y mostrarlo al usuario por algún medio, mediante la implementación de la interfaz *ProgressTopic*, que implementa el método:

```
void notify(ClipData clipData);
```

- obtención de los ficheros de audio.
- obtención de la lista completa de ficheros descargados.

Canales de eventos

La comunicación entre clientes y servidores de descarga se realizará a través de un canal de eventos (*ProgressTopic*). A través de este canal se realizán 4 tipos de llamadas de tipo notify, correspondiéndose con los 4 estados en los que puede hallarse una descarga:

- *Pending*: indica que la descarga está a la espera en la cola de tareas del servidor
- *InProgress*: la descarga y extracción de audio están en proceso
- *Done*: la descarga y extracción se han completado

- *Error*: se ha producido un error durante la extracción

```
enum Status {Pending, InProgress, Done, Error}

struct ClipData {
      string URL;
      Status status;
}

interface ProgressTopic {
      void notify(ClipData clipData);
}
```

Interfaz ProgessTopic

Un segundo canal de eventos permite la sincronización entre los servidores de descarga. Cada servidor se convierte en un suscriptor del topic *SyncTopic*, recibiendo las notificaciones de tipo *requestSync* procedentes de un controlador gobernado por un temporizador (*SyncTimer*). Como respuesta, a este evento, cada servidor envía una lista de tipo SongsList con los archivos que almacena de manera local. A partir de esta lista, los distintos servidores deben obtener los ficheros faltantes, aunque para ello recurrirán a la transferencia directa de ficheros por ssh.

Nótese como los servidores de descarga actúan simultáneamente como suscriptores y publicadores de eventos del canal, mientras que el generador de peticiones de sincronización (*SyncTimer*) es solamente publicador.

```
interface SyncTopic {
      void requestSync();
      void notify(SongsList songsList);
}
```

Interfaz SyncTopic

Factoría de servidores de descarga

El despliegue de los servidores de descarga debe realizarse de manera dinámica, utilizando para ello una factoría (*SchedulerFactory).* Esta factoría contará con un método *make()* para realizar esta operación:

```
interface DownloaderFactory {
      DownloadScheduler* make(string name);
}
```

El manejo de la factoría no está ligado a los clientes, sino que puede producirse de forma independiente. Sin embargo los clientes deberán crear al menos un objeto de tipo *DownloadScheduler* para poder trabajar con el sistema de descargas.

Requisitos adicionales

Además de los aspectos arquitecturales descritos en el apartado anterior, relativo a los servidores, clientes y canales, debe tenerse en cuenta que la solución debe incluir:

- La gestión de todo el servicio debe realizarse a través de *IceGrid*, de manera que la aplicación de demostración incluya un nodo IceGridNode.

- Un documento breve que describa la solución adoptada, y la funcionalidad soportada, listando tanto los aspectos básicos (descritos en el apartado anterior), como las extensiones añadidas a éstos.
- Código fuente del proyecto, que debe incluir tanto los ficheros fuente de clientes y servidores, así como archivos de configuración, archivo xml de la aplicación *IceGrid*, y en general cualquier otro que sea necesario para la puesta en marcha de la aplicación.

Extensiones Opcionales

Como idea de las extensiones que pueden realizarse sobre el proyecto se incluyen las siguientes, aunque no se trata de una lista cerrada ni obligada, sino que podéis plantear mejoras complemetarios:

- extensión para considerar más de un nodo IceGrid. Hay que tener en cuenta que el directorio de descargas compartido entre servidores de descarga también debe ser compartido entre nodos.
- cancelar una tarea en cola en la cola de descargas
- extender la factoría para conocer el número total de servidores desplegados
- tener la posibilidad de eliminar servidores de descarga cuando éstos no están siendo utilizados
- interfaz gráfica para los clientes
- monitor suscrito a ambos canales de eventos que almacene en un archivo una traza de los distintos mensajes enviados a través del canal, incluyendo las marcas de tiempo del momento en el que tienen lugar.
- todos los servidores de descarga deben pertenecer a un mismo *Replica Group*, con el fin de que los clientes puedan conectar directamente con el grupo de réplica y se balancee la carga entre servidores de forma totalmente transparente.