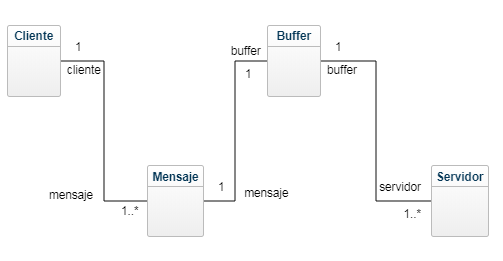
Universidad de Los Andes.   
Departamento de Ingeniería de Sistemas y Computación.   
Infraestructura Computacional  
Caso I

Christian Chavarro Espejo 201613724   
Juan Sanmiguel Mateus 201617603

**FUNCIONALIDAD Y GENERALIDADES CASO I**

**Funcionalidad**

A continuación, se muestra un diagrama de clases simplificado de la solución del caso I.

  
Imagen I: diagrama de clases para la solución del caso I con sus cardinalidades.

**Nota:** en este diagrama se omite la clase que ejecuta el main por simplicidad. La funcionalidad de esta clase y el método principal se explicarán más adelante.

**Descripción de clases:**

**Cliente:** Se encarga de mandar los mensajes por medio de un objeto de tipo mensaje, así como de avisar al buffer cuándo termina. Los objetos de tipo mensaje son almacenados en el buffer, o permanecen en espera hasta que puedan ser agregados.

**Mensaje:** Clase encargada de representar el mensaje creado por el cliente. Este mensaje debe ser encolado en el buffer, cuando pueda realizarse esta acción. Así mismo, los objetos mensaje cuentan con el método responder (), encargado de sumar 1 al mensaje original, de esta manera se da como respondido un mensaje.

**Servidor:** Cuando el servidor inicia, responde todos los mensajes existentes en el buffer, mientras aún puedan ser respondidos. El servidor se comunica con el buffer y ordena que se responda el mensaje.

**Buffer:** Es la clase encargada de conectar el servidor con los mensajes. Además, maneja la concurrencia. Cuenta con un cupo limitado de mensajes, dado al iniciar el programa. Si hay cupo en el buffer, pueden guardarse mensajes, en caso contrario, tendrán que esperar su cupo.

**Principal:** Clase que contiene el main de la aplicación. Este método inicia con la lectura del archivo datos, se encuentra en docs\datos.txt este archivo será detallado más adelante. Tras leer el archivo, inicia la construcción de los clientes y servidores, además del buffer.

**Archivos adicionales:**

El archivo adicional que facilita la funcionalidad de la aplicación se encuentra en docs\datos.txt y será descrito a continuación.

Este documento de texto cuenta con las siguientes líneas:

**NUM\_CLIENTES:** Describe el número de clientes que la aplicación atenderá. Sobre cada cliente, se crearán n solicitudes. Este número, debe corresponder a un entero.

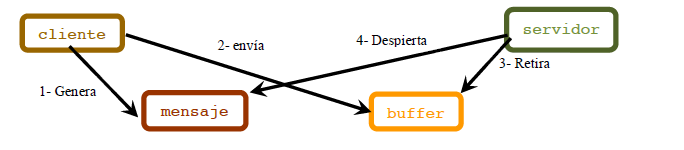
**NUM\_SERVERS:** Detalla el número de servidores que atenderán mensajes que se encuentren en el buffer. Debe ser un número entero.

**TAM\_BUFFER:** Número entero que informa la capacidad del buffer, el número de mensajes que puede almacenar a la vez.

Ahora, para detallar el número de solicitudes realizadas por cada cliente, se tiene una cadena de números separados por “:”. Esta cadena se interpreta como un arreglo.

**#: #: #: #: #: #: #: #** Cada numeral corresponde al número de solicitudes para el cliente con el identificador de la posición del arreglo de este numeral. Para el caso **#: #: #: #: #: #: #: #: #**, se tienen 8 clientes, con identificadores 0…7 y cada numeral corresponde a la cuantía de solicitudes del cliente de id i.

**Sincronización e interacción entre los objetos de la aplicación.**

  
Imagen 2: interacción entre los objetos del caso de estudio.

Para cada clase, se describirá cómo interactúa con los demás. Estas clases, son las que fueron descritas anteriormente.

**Cliente:** Interactúa directamente con la clase mensaje, pues lo construye. Además, tiene interacción con la clase buffer, pues envía los mensajes a él y le notifica cuando se retirará.

**Servidor:** Presenta relación directa con buffer y mensaje. Para el caso de buffer, se encarga de verificar que haya mensajes que necesiten respuesta. Si hay mensajes que necesiten respuesta, el servidor inicia el retiro del mensaje del buffer.

**Mensaje:** Los mensajes son creados por un cliente, se responden por solicitud de un servidor y almacenador en el buffer. El buffer se encarga de almacenar los mensajes hasta que tengan una respuesta.

**Buffer:** En esta clase se presenta la sincronización.

A la hora de guardar un mensaje, mientras este no pueda ser almacenado, se ejecuta el método yield () que regresa el hilo a su estado justo antes de inicial. Una vez agregado, notifica a todos que el estado del buffer fue modificado.

Para retirar los mensajes, si no hay en la cola, el método espera. Una vez encuentre un mensaje disponible, se procede a responder y notificar a todos que el estado del buffer fue modificado.