Descripción de la solución

El problema se encaró desde una perspectiva donde el modelo que soluciona la lógica del problema (manejo de inscripciones/desinscripciones) es totalmente independiente del resto de la solución (manejo de threads y sockets para la comunicación cliente-servidor). Por lo que será el servidor quien contenga al modelo y se encargue de gestionar la comunicación de los clientes con éste, teniendo en cuenta la concurrencia de los distintos clientes al modelo.
Para poder acceder al modelo de manera concurrente se creó un monitor del modelo, tanto el modelo como el monitor comparten una misma interfaz por lo que el resto de las clases no saben con cual están comunicándose realmente. La utilización de mutex solo fue necesaria para los métodos de inscripción/desinscripción ya que los demás solo hacían lecturas de las estructuras de datos del modelo.
La atención simultanea de los clientes se logró creando un thread por cada cliente donde se ejecuta un proxy que se encarga de la comunicación con el cliente vía sockets, recibiendo comandos del cliente, enviándoselos al modelo y retornando la respuesta por el mismo medio al cliente.
Sumado a estos el thread principal del servidor se encarga de verificar la condición de fin de programa y otro thread se encarga de aceptar a los clientes nuevos, lanzar el thread correspondiente y luego recolectar los threads de los clientes desconectados.

Esquema de diseño

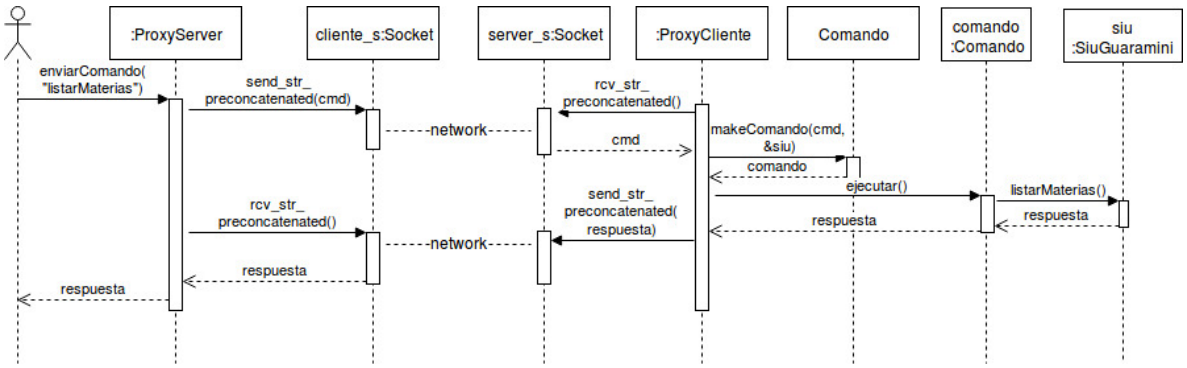

Figura 1: Diagrama secuencial envío del comando listarMaterias.