

Modelos de servidor (Sockets II)

Estas diapositivas explican las diferentes estrategias para el diseño de servidores de conexiones con sockets bloqueantes.

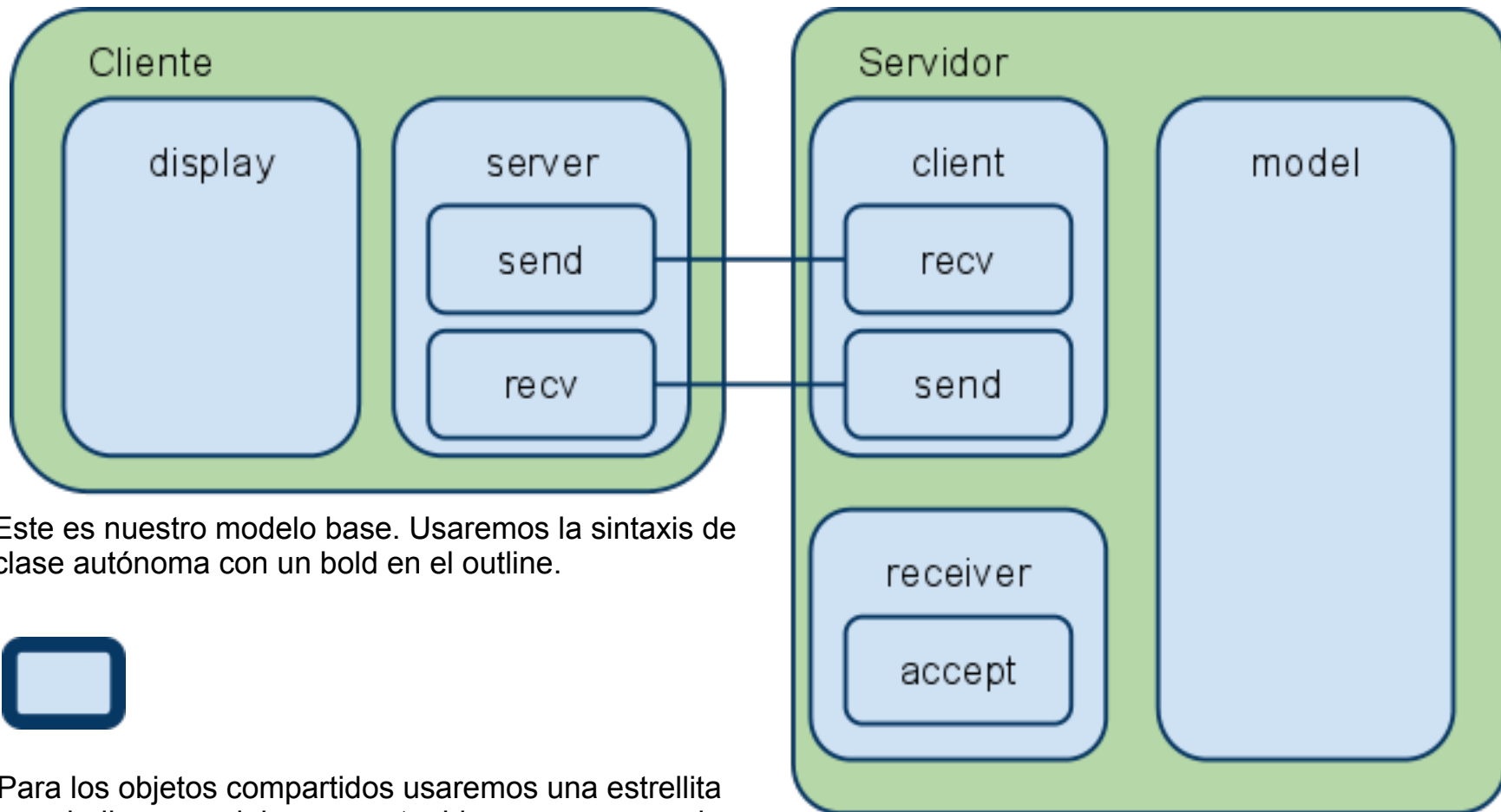
Se asume que el lector tiene buen conocimiento sobre el funcionamiento de los sockets.

Autor: Gonzalo Merayo

Revisiones:

1. Gonzalo Merayo
2. Leandro H. Fernández

Modelo general



Este es nuestro modelo base. Usaremos la sintaxis de clase autónoma con un bold en el outline.

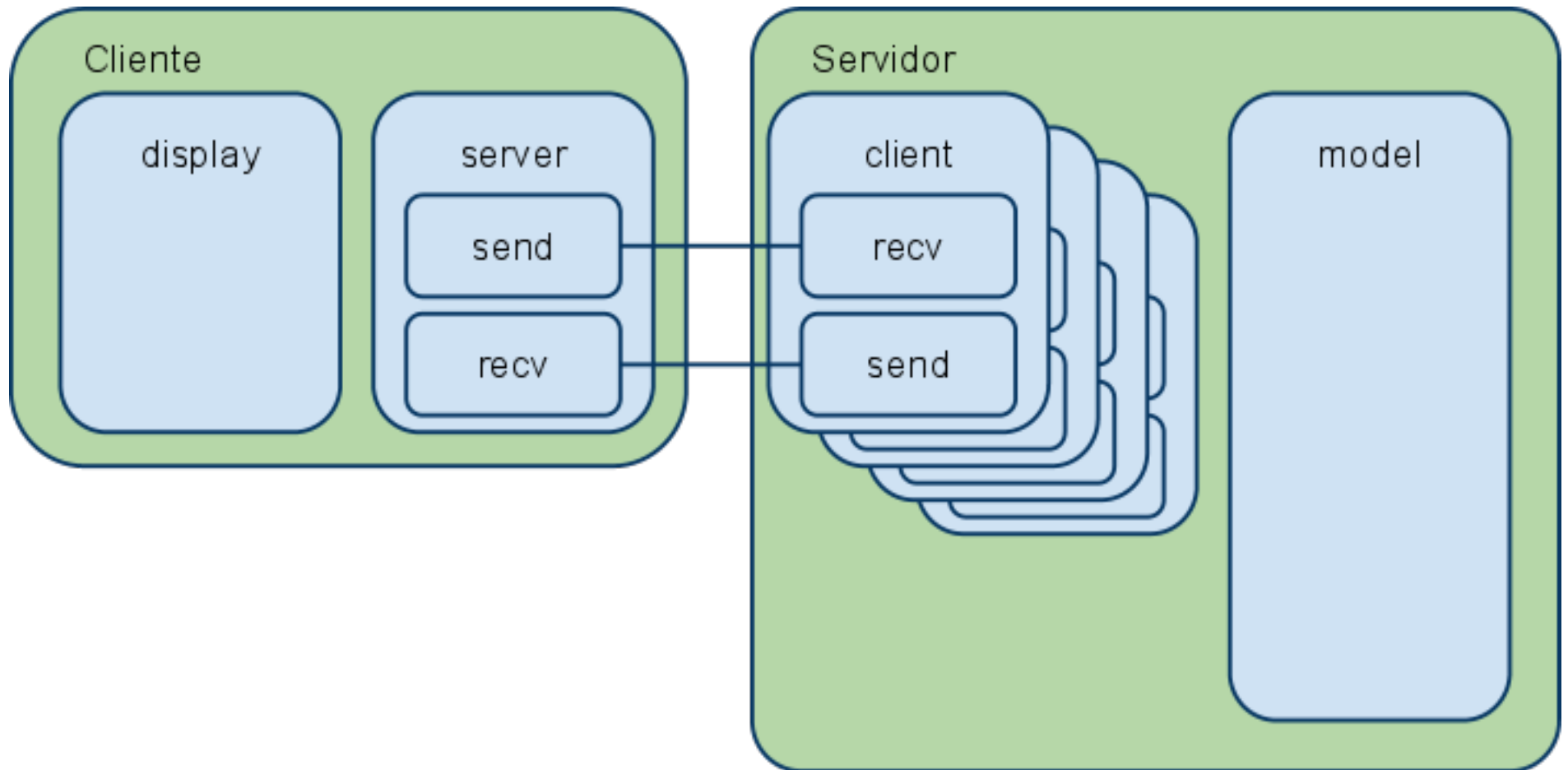


Para los objetos compartidos usaremos una estrellita para indicar que debe ser protegido por concurrencia.

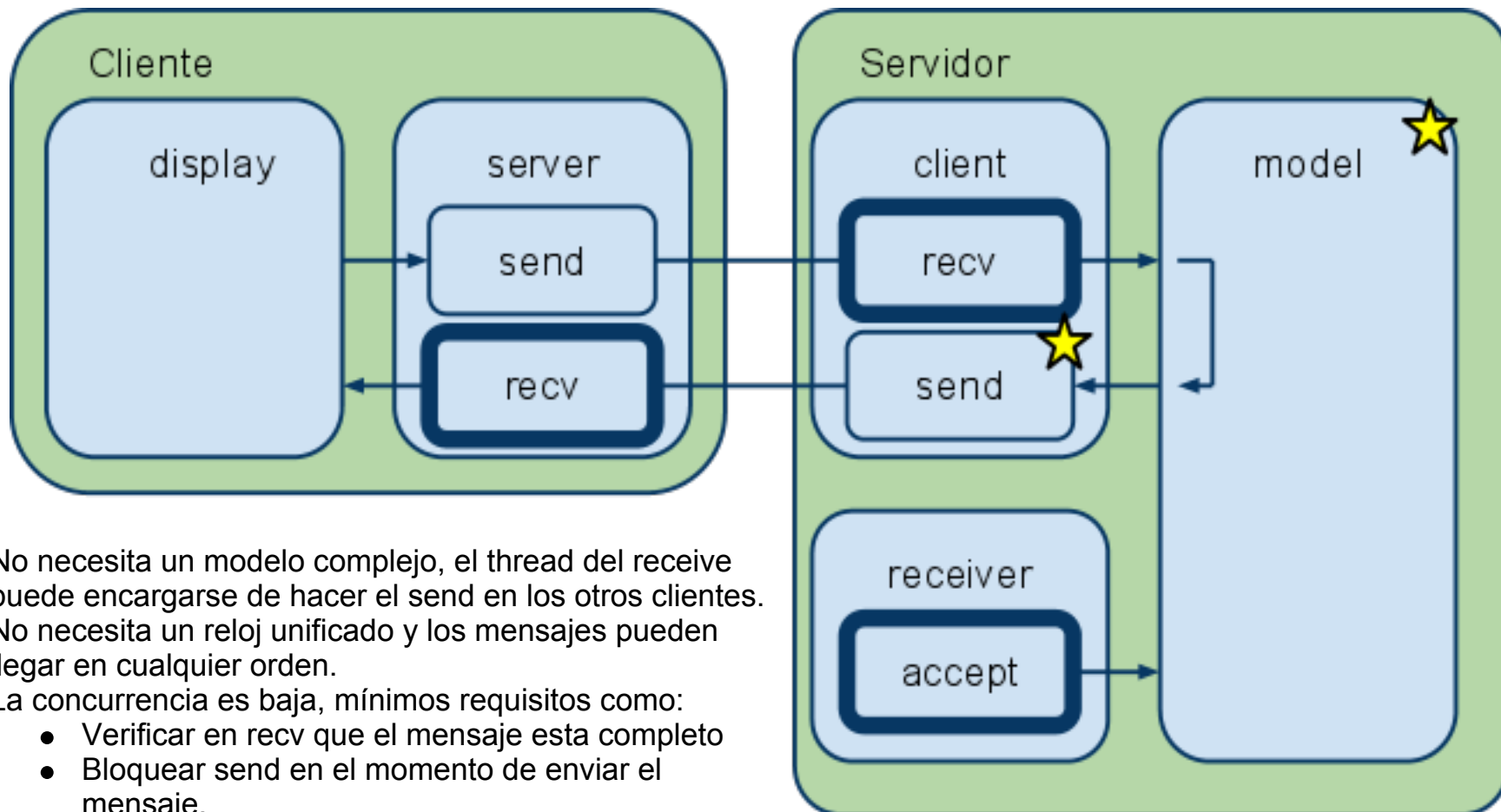
Abajo esta la sintaxis que usaremos para un consumer producer. Notar la segunda flecha roja, la flecha indica el sentido del mensaje, el color indica que es un pull de datos en lugar de push.



En realidad tendremos muchos clientes...



Servidor de Chat



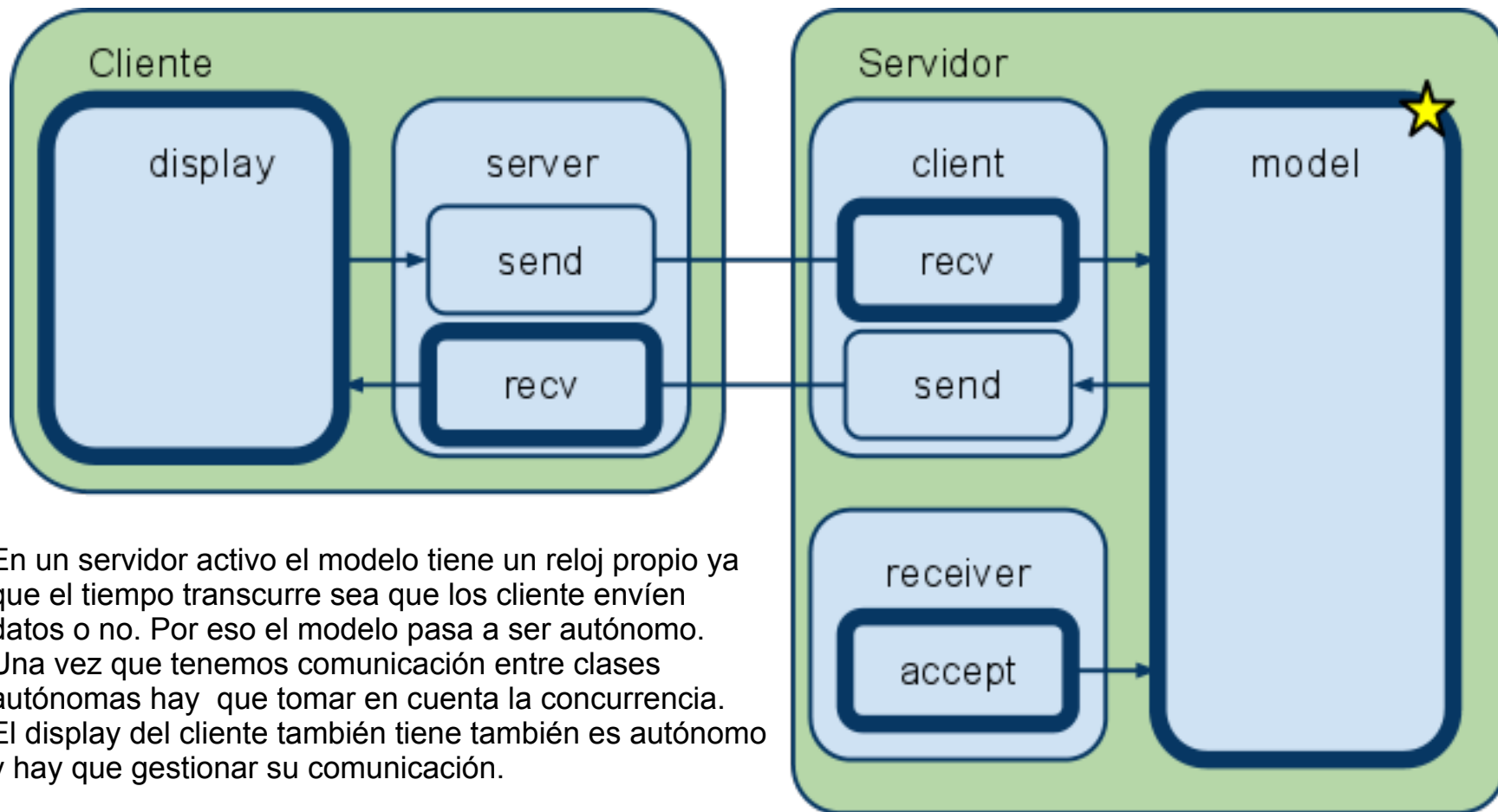
No necesita un modelo complejo, el thread del receive puede encargarse de hacer el send en los otros clientes. No necesita un reloj unificado y los mensajes pueden llegar en cualquier orden.

La concurrencia es baja, mínimos requisitos como:

- Verificar en **recv** que el mensaje esta completo
- Bloquear **send** en el momento de enviar el mensaje.
- Opcionalmente se puede bloquear **model** si se desea que todos reciban los mensajes en el mismo orden.

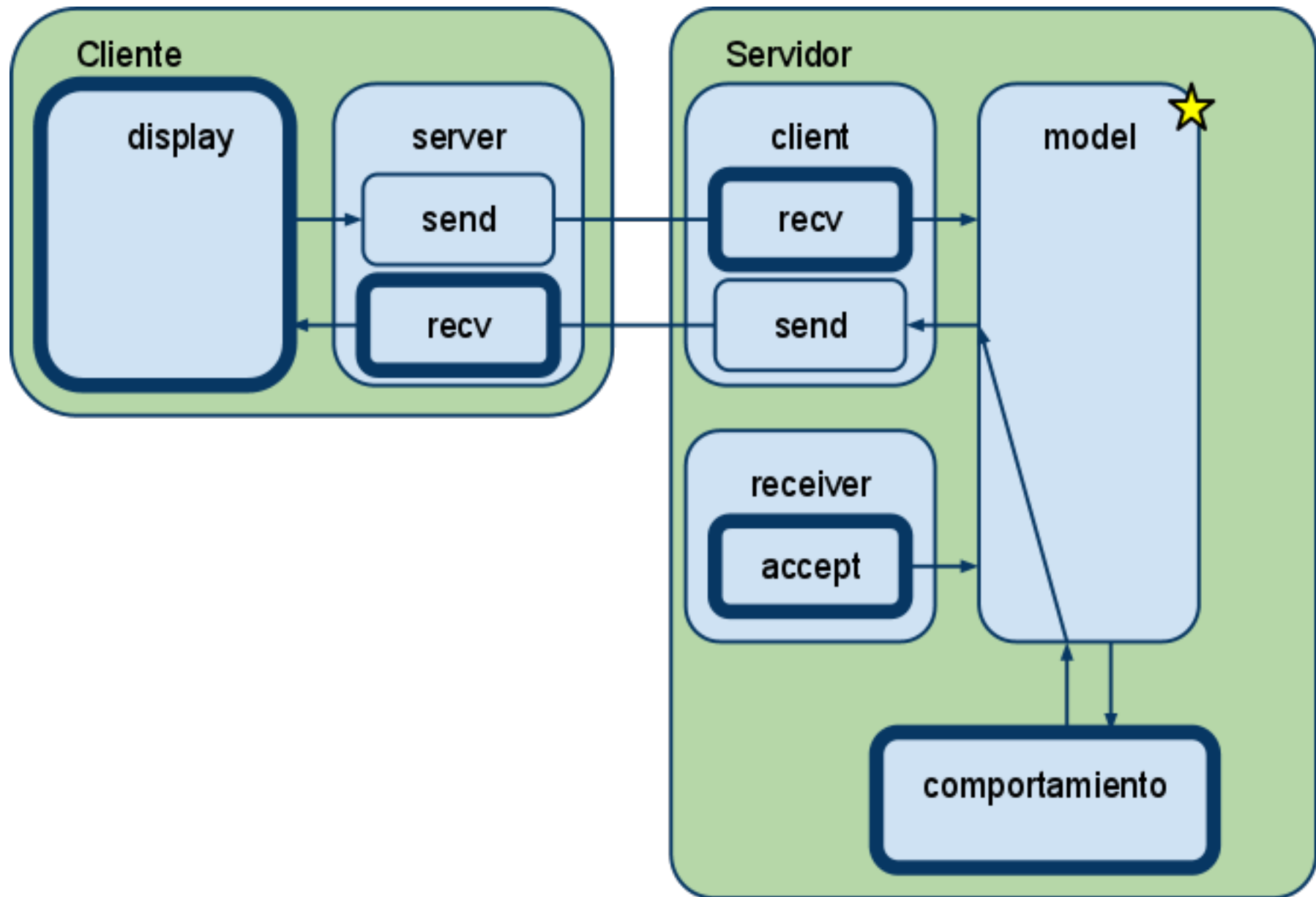
Se podría usar el mismo modelo para un juego por turnos

Servidor activo. Alternativa base

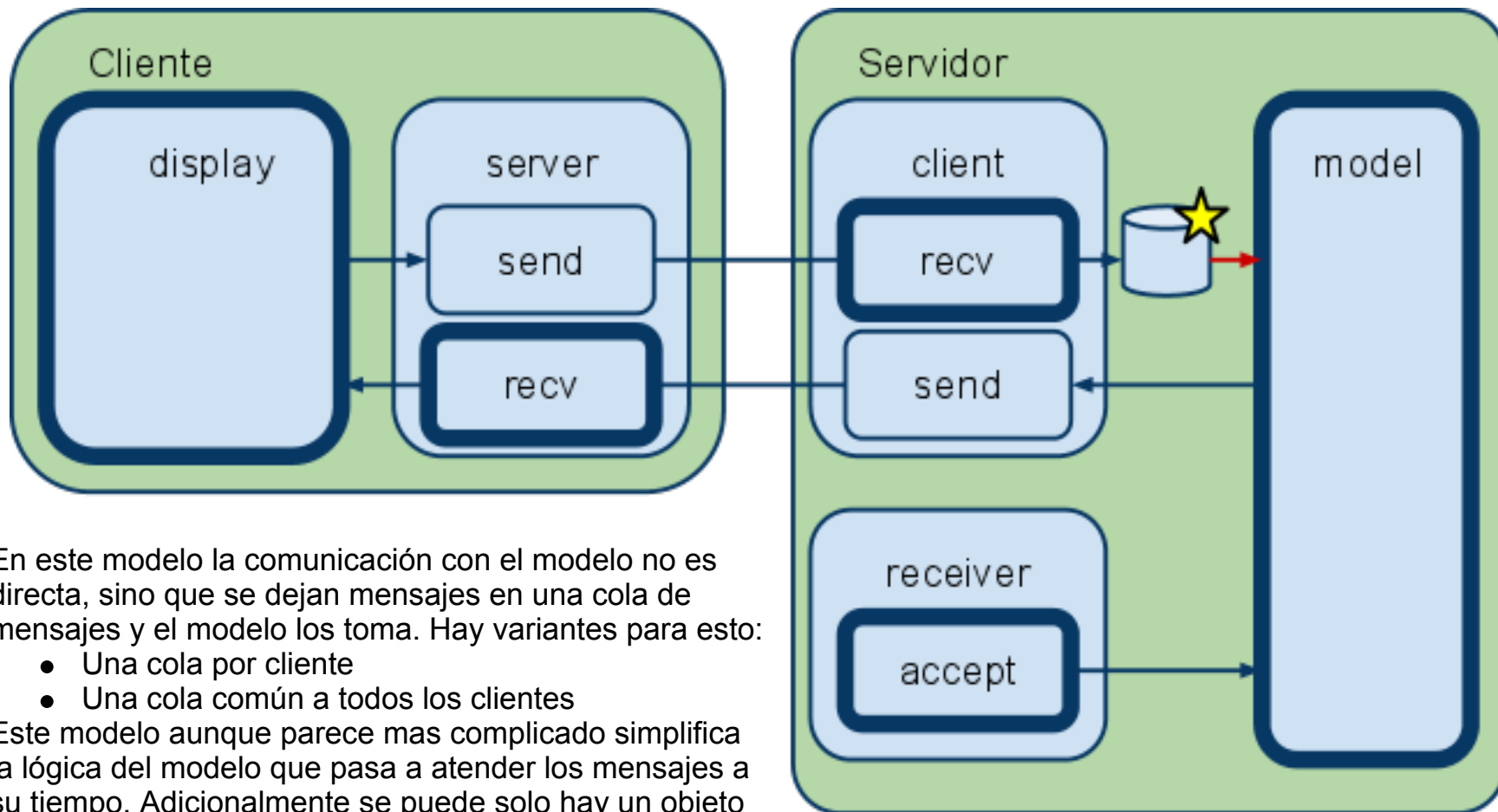


En un servidor activo el modelo tiene un reloj propio ya que el tiempo transcurre sea que los cliente envíen datos o no. Por eso el modelo pasa a ser autónomo. Una vez que tenemos comunicación entre clases autónomas hay que tomar en cuenta la concurrencia. El display del cliente también tiene también es autónomo y hay que gestionar su comunicación.

Servidor activo, modelo como comunicación



Servidor activo. Entrada Consumer-Producer

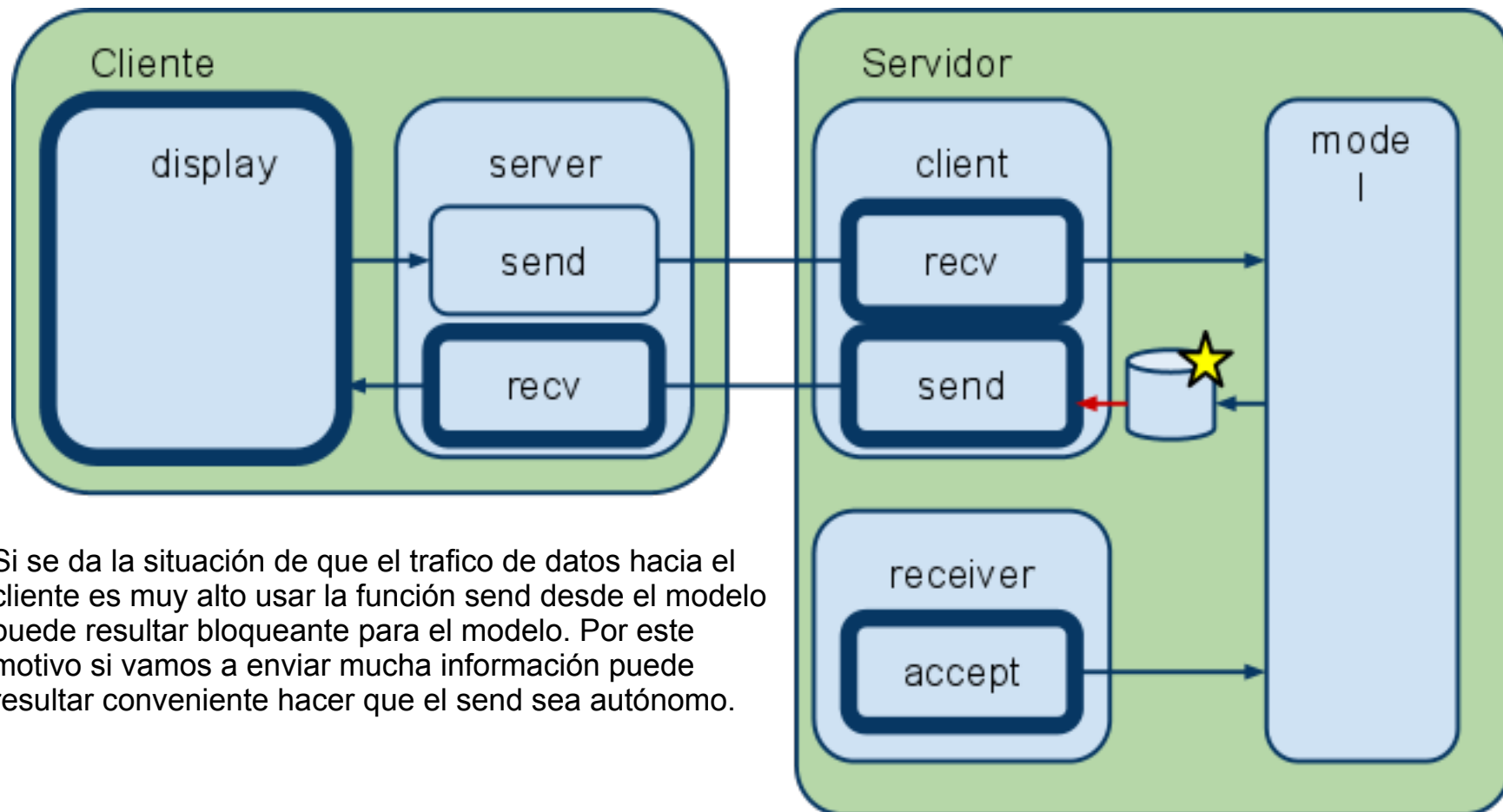


En este modelo la comunicación con el modelo no es directa, sino que se dejan mensajes en una cola de mensajes y el modelo los toma. Hay variantes para esto:

- Una cola por cliente
- Una cola común a todos los clientes

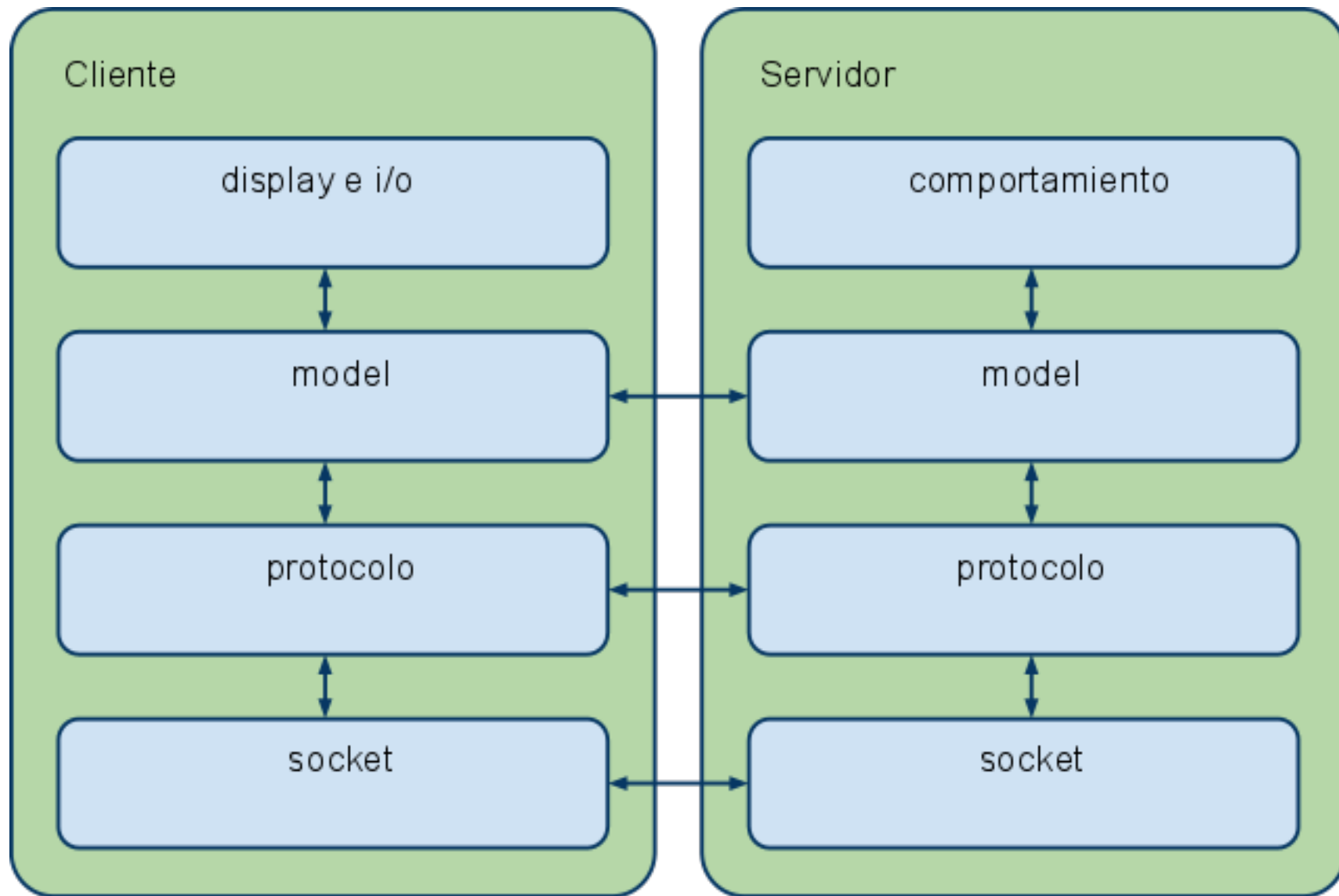
Este modelo aunque parece mas complicado simplifica la lógica del modelo que pasa a atender los mensajes a su tiempo. Adicionalmente se puede solo hay un objeto compartido que por ser una cola es muy simple de usar.

Servidor activo. Alto trafico



Si se da la situación de que el trafico de datos hacia el cliente es muy alto usar la función send desde el modelo puede resultar bloqueante para el modelo. Por este motivo si vamos a enviar mucha información puede resultar conveniente hacer que el send sea autónomo.

Modelo replicado y protocolo



Es común que repliquemos el modelo como esta en el server a todos los clientes mediante un protocolo. La forma mas recomendable es usar el mismo código model en los 2 programas antes de construir la interfaz, el comportamiento y el protocolo. A medida que evoluciona el desarrollo se construyen las capas de protocolo, y el resto alrededor de un modelo común. Esto adicionalmente simplifica el testing y hace el desarrollo iterativo e incremental.

- Cualquier combinación de las anteriores puede ser valida; cuáles son las mejores opciones depende de la situación.
- También se pueden explorar estas alternativas en el cliente.
- Se recomienda documentar adecuadamente el diseño para que quien corrige entienda qué se pretende.