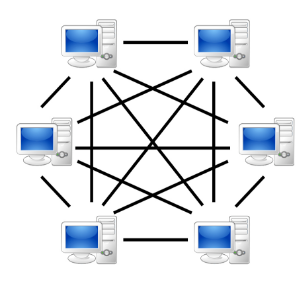
**Caso 1**

Para el caso 1 se propone utilizar una arquitectura **Peer-to-Peer** (P2P) ya que se pide que todos los nodos que integran la red puedan actuar como servidor o cliente y que estén todos sincronizados entre sí.

Además, esta arquitectura es descentralizada ya que todos los nodos son iguales.

Para conectar todas las PCs entre sí, proponemos utilizar un espacio de **memoria compartida,** ya que todas las PCs tienen que poder acceder a los datos, con este mecanismo se activaría un flag permitiendo bloquear y desbloquear los accesos.

Con esta arquitectura, dicho pedido de cumple, el diagrama sería el siguiente:



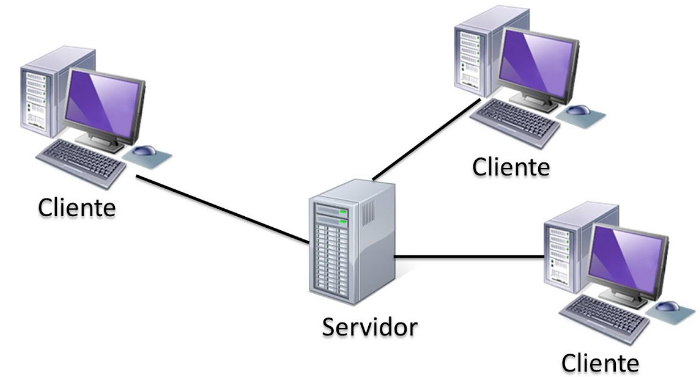
**Caso 2**

Para el caso 2 se propone utilizar una arquitectura **Cliente-Servidor** ya que al ser un servicio en donde los equipos terminales ubicados en los distintos clientes tienen la necesidad de conectarse a un único servidor central.

Además, se pide que las terminales soporten alta frecuencia de operaciones y que no se quede bloqueada, lo cual con este modelo se cumple.

Por otro lado, se pide atomicidad y esta arquitectura lo cumple ya que todas las transacciones son canalizadas por un solo servidor.

El patrón de integración que recomendaríamos para esta solución es **Eventos** (Mensajería asincrónica) ya que este se adecua perfectamente a lo pedido, es decir, pide que bajo ningún momento las terminales se bloqueen (razón por la cual descartamos **Call-Return**) y este patrón al funcionar de forma asincrónica cumple lo pedido.



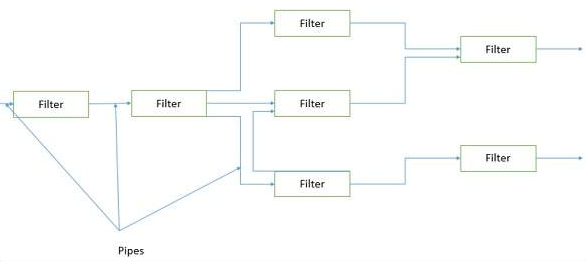
**Caso 3**

Para el caso 3 se propone utilizar una arquitectura de **Flujo de Datos** ya que se pide que cada nodo sea capaz de administrar y redireccionar datos a otros nodos de acuerdo a reglas de negocio.

Además plantea la posibilidad de en algunos casos, realizar una transformación de datos para comunicárselo al siguiente nodo.

El conector de integración que proponemos utilizar es el de Microservicios, ya que son sistemas independientes y todos cuentan con conexión a HTTP y este conector, normalmente utiliza este para conectarse entre los sistemas.

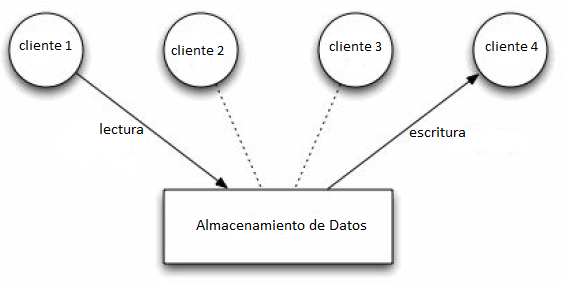
Se podría usar SOA pero estaría implementado medio forzado y generaríamos una solución más compleja ya que la gestión en SOA es más compleja, además que en el enunciado ya está diciendo que son sistemas diversos.



**Caso 4**

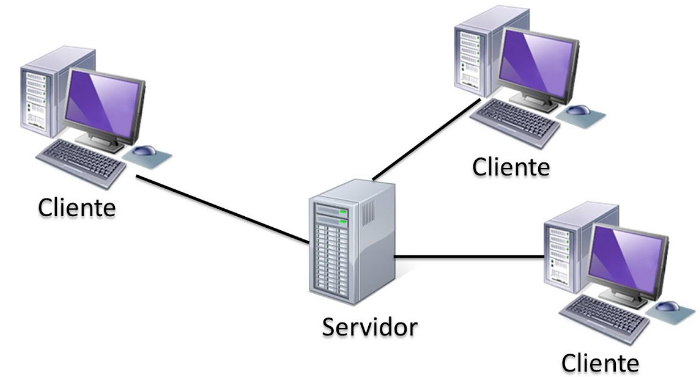
En este caso creemos que el enunciado describe el uso de una arquitectura **Centrada en Datos** ya que es fundamental que la información local con la remota se encuentre sincronizada a costa de que se puedan efectuar bloqueos.

Este estilo arquitectónico garantiza cubrir dicho requerimiento ya que el repositorio de datos (pizarrón) sirve para poder coordinar las transferencias de información que realicen los distintos clientes y es el único medio por el cual dichos clientes se pueden comunicar.



**Caso 5**

En este caso creemos que sería necesario implementar una arquitectura **Cliente-Servidor** donde cada uno de los buques (clientes) tienen la necesidad de conectarse a un único servidor central (base central de la Marina) para notificar las condiciones que atraviesan y recibir reportes que son de su interés.



**Caso 6**

Para este último caso, ante la necesidad de contar con servicios independientes entre sí (sean de un tercero o propios), contando con una gran flexibilidad para poder adaptarlo según lo desee cada cliente, creemos beneficioso implementar una **Arquitectura MVC** donde la **capa** de modelo se diseñe **Orientada a Servicios** con conectores Call&Return **REST**.

De esta manera contamos con un protocolo estándar de los más habituales del mercado para permitir una gran adaptabilidad a cualquier otro servicio que sea necesario agregar sin perder la precisión que ofrece XML o JSON.

Al diseñar la aplicación con una arquitectura MVC garantizamos la flexibilidad y mantenibilidad antes mencionada y que creemos fundamental para un sistema de este tipo.