Guía Nro. 3

Diseño de Arquitectura

versión 1
octubre 2017

Modalidad: Actividad Grupal
Fecha de entrega : 1/11 (subir al repositorio del grupo y notificar por mail al docente)

Bibliografía:

- Bass, L., Software Architecture in Practice.
  Addison-Wesley. 2012.
- Clemens, P., Documenting Software Architectures Views and Beyond.
  Addison-Wesley. 2011.
- Cervantes, H., Designing Software Architectures: A Practical Approach.
  Addison-Wesley. 2016.
- Richards, M., Software Architecture Patterns.
  O'Reilly. 2015.
- Hohpe, G., Enterprise Integration Patterns.
  Addison-Wesley. 2003.
- Trowbridge, D., Integration Patterns.
  Microsoft. 2004.
- Apuntes de Cátedra

Antes de comenzar...

Diseñar una arquitectura es tomar decisiones, trabajar con las habilidades y materiales disponibles con el propósito de satisfacer los requerimientos y restricciones.
Mientras los patrones de diseño se enfocan a escala objeto (incluyendo instanciación, estructuración y comportamiento), se considera a los patrones de diseño de arquitectura, aquellos que influyen sustancialmente en la solución de alguno de los drivers arquitectónicos. Por otro lado la integración empresarial es muy compleja para ser diseñada desde un solo enfoque. Para ello existen patrones y modelos que sirven de guía para diseñar una solución robusta y adecuada en función de los objetivos.
Un sistema no está construido exclusivamente a partir de un solo estilo o patrón. Lo habitual es que un sistema sea una amalgama de diferentes estilos. A su vez, diferentes áreas de un sistema pueden exhibir diferentes estilos.

Objetivos

A continuación se presentarán diferentes casos para los cuales el equipo deberá indicar la solución más apropiada en función de las características del contexto y los requerimientos expresados en cada uno. En función de la situación planteada, la propuesta estará basada en uno o más patrones de arquitectura y/o integración, también, puede requerir más de una vista para comunicar la solución. Se recomienda estudiar cada caso y decidir a partir del consenso del equipo de trabajo.

Caso 1: **Sistema de archivos distribuido**

En un contexto descentralizado, el objetivo es que todos los nodos que integren la red mantengan sincronizados un conjunto de recursos en común. Cada nodo debe poder actuar tanto en modo cliente como servidor. La solución debe ser flexible para escalar horizontalmente y soportar eventuales fallos de conectividad entre nodos.

Se pide:

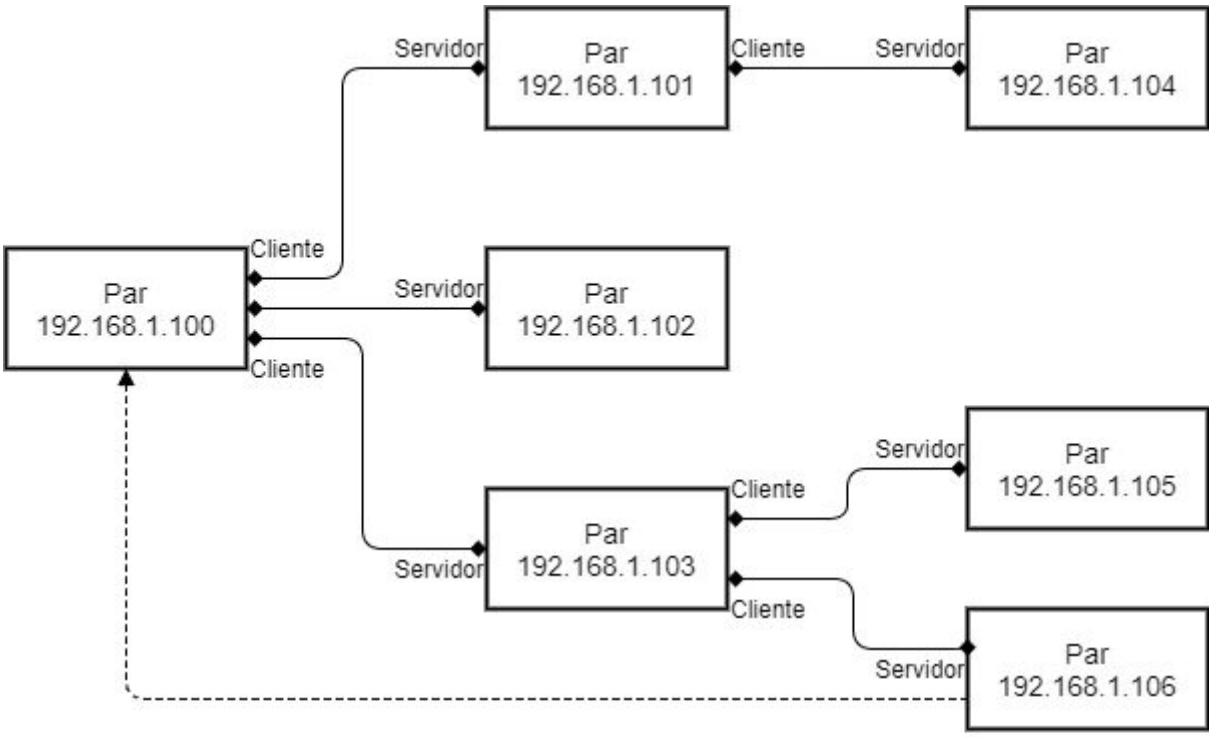
- Diseñar y documentar una arquitectura.
- Mencionar las posibles alternativas de conectores de integración indicando ventajas y desventajas de cada una.

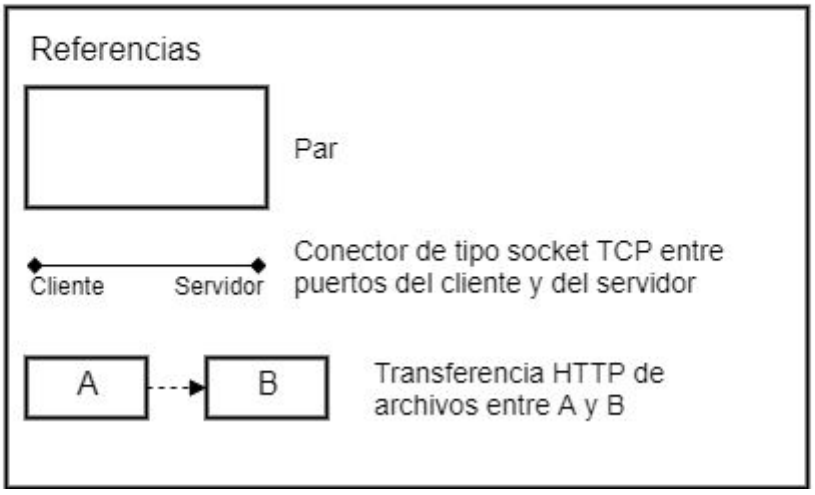
Para implementar el sistema de archivos distribuido, optamos por una arquitectura de tipo llamada y retorno, particularmente peer-to-peer. Dentro de los diferentes tipos de arquitecturas peer-to-peer optamos por una arquitectura peer-to-peer desestructurada, sin estructura impuesta, en la que los nodos se conectan entre sí de manera aleatoria (las relaciones entre los pares se forman arbitrariamente).

Un par (192.168.1.100) que requiere un recurso realiza pedidos a los pares que conoce (192.168.1.101/102/103); si estos nodos no tienen el recurso buscado, cada uno realiza un pedido a los nodos que conoce (192.168.1.101 a 192.168.1.104 y 192.168.1.103 a 192.168.1.105 y 192.168.1.106). Cuando se encuentra un par (192.168.1.106) que posee el recurso requerido, el par responde a la solicitud, la cual se trasmite hasta el par inicial (192.168.1.100), para luego dar comienzo a la transferencia de archivos entre ambos pares.

Las arquitecturas peer-to-peer se caracterizan por ser escalables (“flexible para escalar horizontalmente”), robustas (“soportar eventuales fallos de conectividad entre nodos”) y descentralizadas (no hay un servidor central en el que estén alojados todos los archivos; los nodos actúan simultáneamente como clientes y servidores).

La alternativa a esta solución podría ser un enfoque clásico del tipo cliente-servidor, lo cual si bien aceleraría la búsqueda de recursos y aseguraría que los mismos estén disponibles todo el tiempo (ya que no se depende de los nodos que están conectados a la red) tiene como desventaja la centralización de todo el modelo alrededor del servidor, pudiendo resultar en un cuello de botella (ante una demanda elevada) y un único punto de fallo (ante una eventual caída del servidor, ningún cliente podrá obtener recursos).





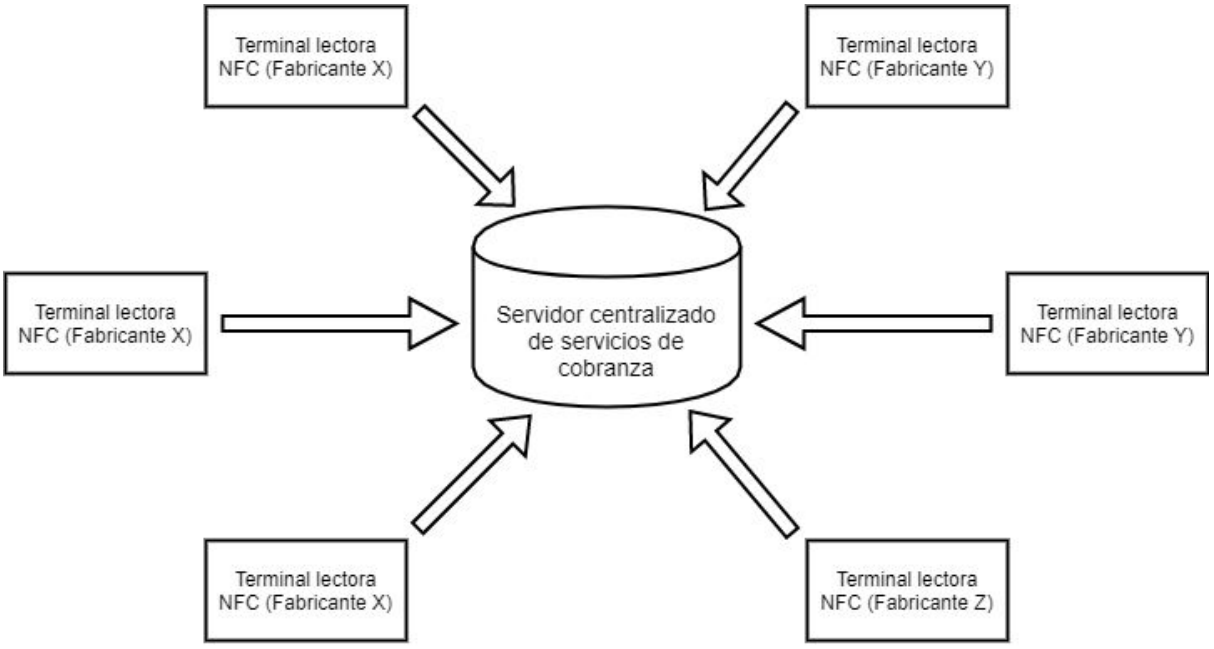
Caso 2: **Terminales NFC**

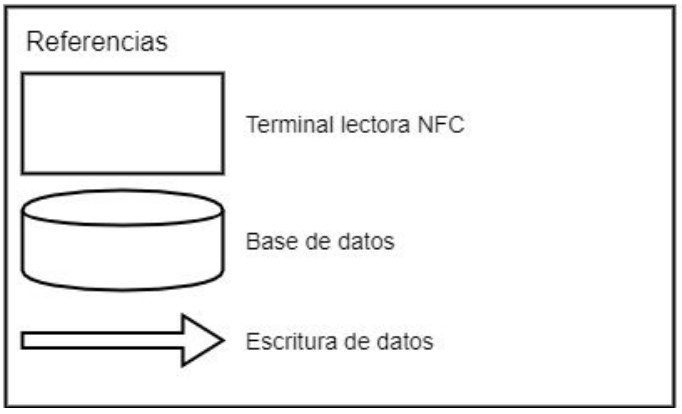
Un proveedor de servicios de cobranza necesita integrar las terminales lectoras NFC ubicadas en las tiendas de sus clientes, con el servidor central. Cada vez que se genera una transacción de compra, la terminal debe transmitir la operación asegurando su atomicidad. Se desea que en ningún momento la terminal quede bloqueada a la espera de una confirmación de recepción por parte del servidor de modo que, además, tolere alta frecuencia de operaciones. Asimismo, la solución debe ofrecer la suficiente flexibilidad para integrar diferentes fabricantes de terminales y soportar múltiples formatos de representación de los datos.

Se pide:

- Diseñar y documentar una arquitectura.
- Indicar el o los patrones de integración recomendados y justificar mencionando aspectos críticos del caso.

En este caso, optamos por una arquitectura centrada en datos, como una pizarra o blackboard. Creemos que es el patrón más adecuado a utilizar, dado que se requiere que todas las terminales lectoras NFC transmitan la información de las operaciones a un servidor central.





El patrón de integración que consideramos adecuado para este escenario es el bus de mensajes, ya que permite mantener desacoplados los sistemas que se están integrando (las terminales lectoras NFC de diversos fabricantes y el servidor central del proveedor de servicios de cobranza). Además, trabaja con mensajes asincrónicos como mecanismo de transporte, lo que implica que las terminales pueden enviar mensajes al servidor central y no quedar bloqueadas a la espera de respuesta, como se requiere. Otras ventajas que

proporciona este patrón de integración es el envío de mensajes con alta frecuencia (“tolere alta frecuencia de operaciones”) y la posibilidad de utilizar diferentes formatos de mensajes, algo útil para trabajar con terminales de fabricantes diferentes.

Caso 3: **Integración de sistemas empresariales**

Se precisa establecer interoperabilidad entre sistemas de tecnologías diversas. Si bien algunos de estos sistemas están basados en tecnología antigua (sistemas legacy), todos cuentan con acceso a Internet a través de HTTP. En algunos casos puede ser necesario realizar algunas conversiones en el soporte de representación de los datos. La solución deberá permitir administrar y redireccionar flujos de acuerdo a reglas de negocio específicas para diferentes servicios. Es deseable que también de soporte al envío de mensajes entre nodos.

Se pide:

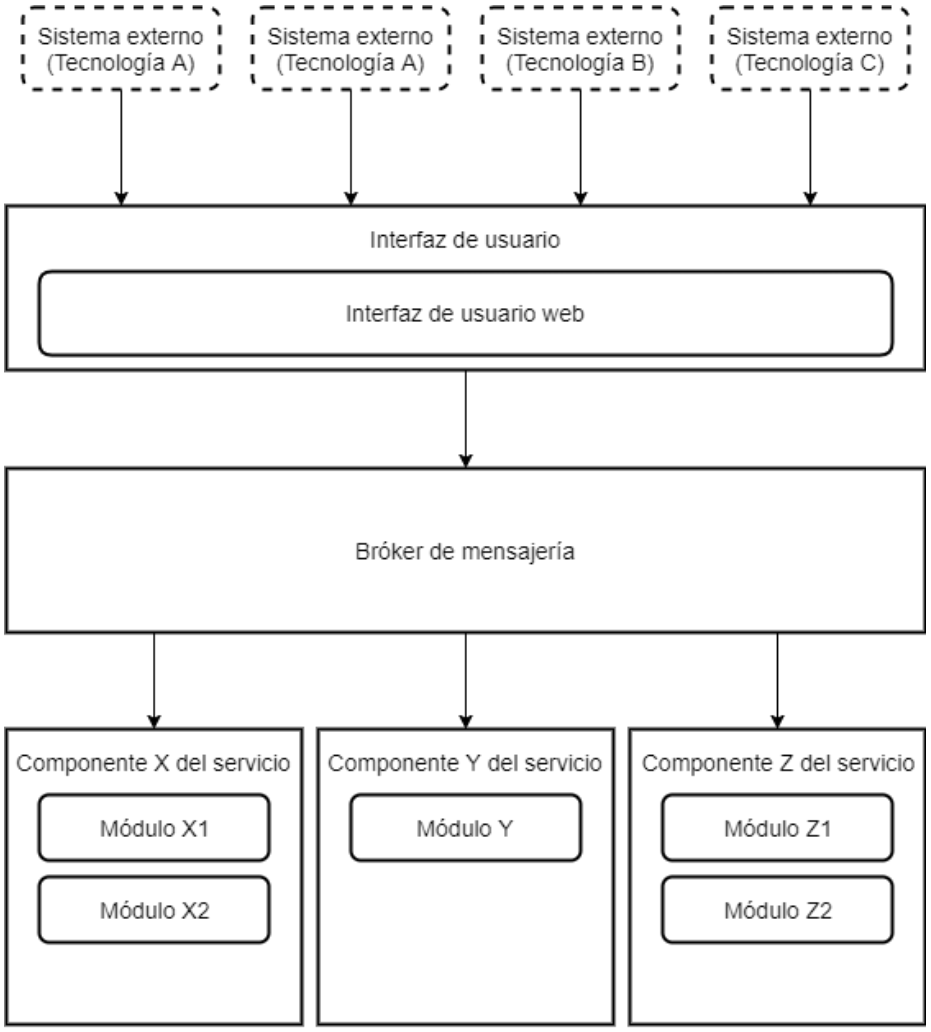
- Diseñar y documentar una arquitectura.
- Mencionar las posibles alternativas de conectores de integración indicando ventajas y desventajas de cada una.

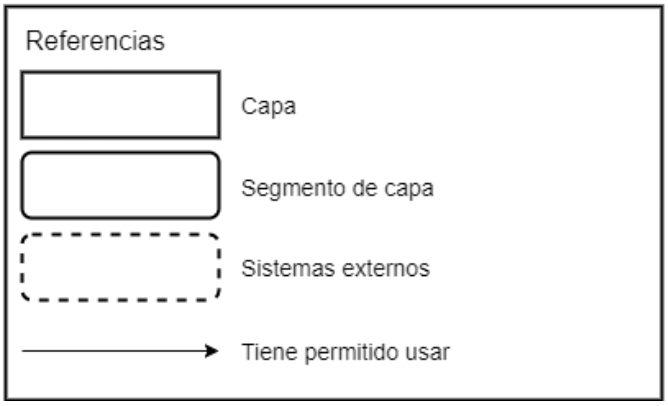
La arquitectura elegida para encarar este caso es la arquitectura basada en microservicios. La ventaja de esta arquitectura frente a una arquitectura SOA (service oriented architecture) tradicional radica, en este caso, en la posibilidad de dividir una aplicación monolítica en diferentes servicios, redireccionando cada pedido que llega a los diferentes servicios con funcionalidad específica.

De las principales topologías de microservicios -API REST, aplicación REST y mensajería centralizada-, optamos por un modelo de mensajería centralizada, ya que se desea que la solución soporte el envío de mensajes entre nodos.

Esta arquitectura se recibe pedidos HTTP, lo que le permite ser compatible con todos los sistemas que plantea el caso, dado que todos cuentan con acceso a internet.

Los microservicios, al igual que las arquitecturas SOA, permiten establecer una mayor interoperabilidad entre las diferentes aplicaciones; no obstante, los microservicios constituyen una alternativa simplificada y menos costosa de implementar.





La alternativa a la arquitectura planteada serían las topologías API REST y aplicación REST. Con la primera, se pierde el bróker de mensajería, con lo que la solución dejaría de soportar el envío de mensajes entre nodos, y el redireccionamiento de flujos de acuerdo a las reglas de negocio. Con el segundo, sólo se prescinde del bróker de mensajería.

Caso 4: **Tráfico aéreo**

En la consola de control de tráfico aéreo donde el operador desarrolla su actividad analítica, reside gran parte del procesamiento de datos recuperados de un servidor. Dada la criticidad del sistema, es tolerable que ocurran bloqueos en la terminal cuando debe sincronizar la información local con la remota. Al tratarse de una solución integral donde todo el software es desarrollado por el mismo fabricante, no es necesario ofrecer representación estandarizada de los datos que intercambien.

Se pide:

- Indicar a qué patrón de arquitectura está haciendo referencia el enunciado.
- Recomendar el patrón de integración más apropiado, explicando sus principales fortalezas y también mencionar aspectos críticos de su implementación.

Basado únicamente en el enunciado, el único patrón arquitectónico que pareciera encajar es el de cliente-servidor, considerando que hay un cliente (la consola de control de tráfico aéreo que recupera y envía datos procesados a un servidor, y que su comunicación se sincrónica, dado que aclara que es tolerable que ocurran bloqueos en la terminal (el cliente); cabe destacar que esto resulta algo poco intuitivo.

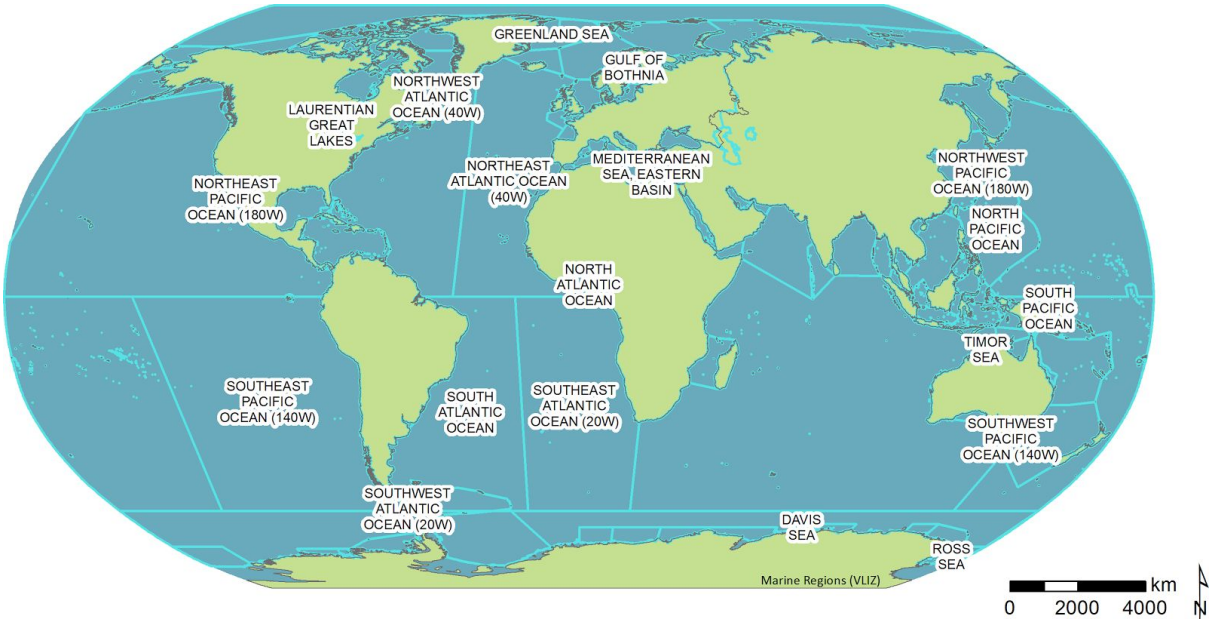
Tomando tan sólo la idea general del enunciado, la de un operador de control de tráfico aéreo, nos podríamos imaginar que el patrón que más se ajusta es la de un patrón centrado en datos, como podría ser el de pizarrón, donde hay un número de agentes (los operadores de las terminales de control de tráfico aéreo de los diversos aeropuertos del mundo) que se encuentran actualizando únicamente los datos de los vuelos de sus aeropuertos y sincronizando esa información en un servidor central (la pizarra).

El patrón de integración más apropiado para ambas soluciones sería el de una base de datos compartida, que permitiría realizar actualizaciones frecuentes para evitar inconsistencias; sin embargo, es importante planear una buena estrategia de recupero ante fallas, ya que la información es crítica y este enfoque cuenta con un único punto de falla: la base de datos en sí y su backup.

La base de datos también podría constituir un cuello de botella en términos de rendimiento, si se consideran todos los vuelos que salen o llegan a todos los aeropuertos del mundo en un determinado momento.

Caso 5: **Marina mercante**

Una compañía naviera precisa actualizar el modo como sus buques notifican a la base acerca de las condiciones meteorológicas de la región oceánica que están atravesando. A su vez, el sistema deberá suministrar a cada buque de reportes meteorológicos regionales que sean de su interés. Como la calidad del canal de comunicación satelital podría verse afectado por lluvias intensas, es crítico que, tanto en las mediciones [buque → base] como los informes [base → buque], esté garantizada su integridad.



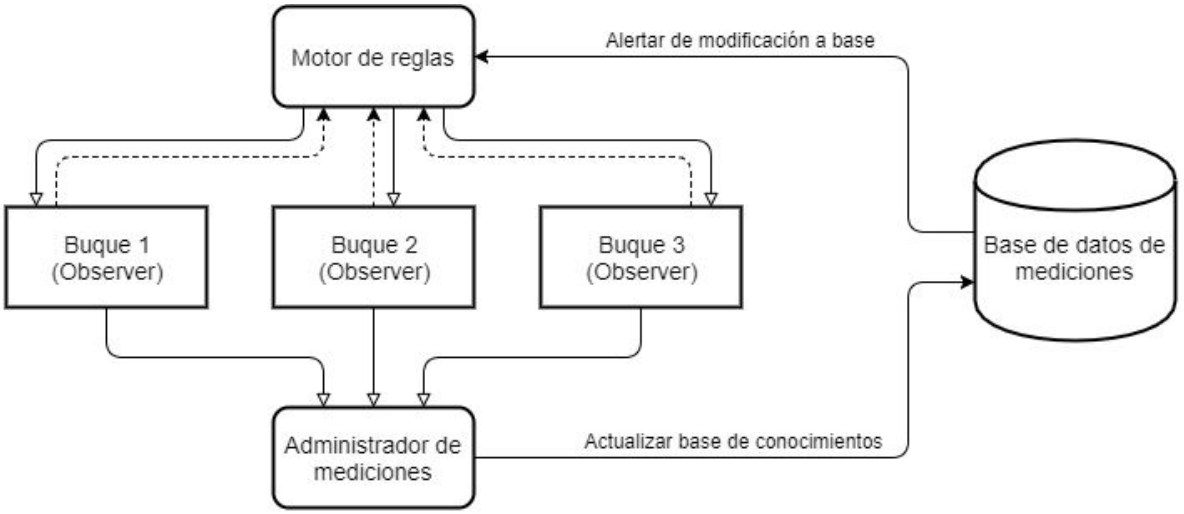
Fuente: http://www.marineregions.org

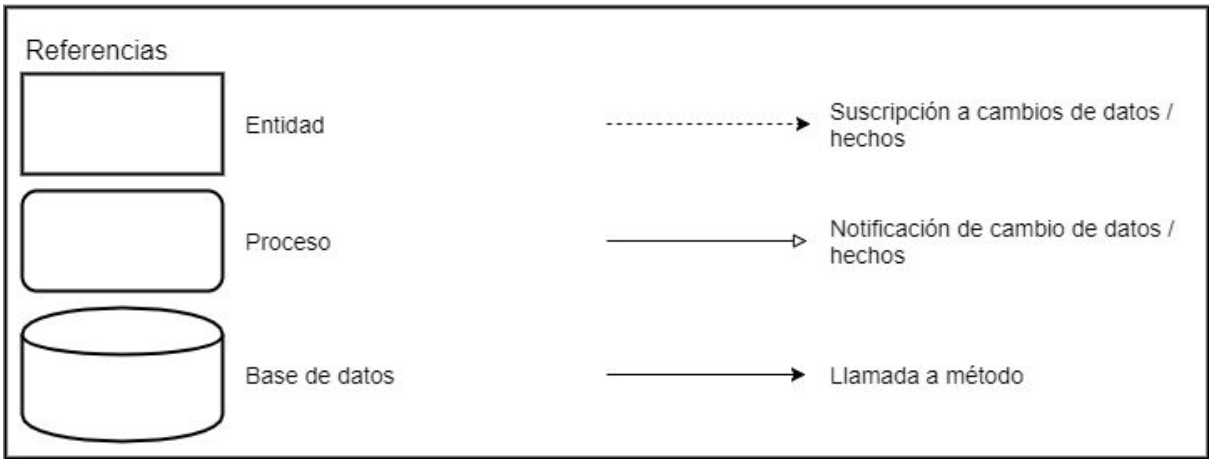
Se pide:

- Diseñar y documentar una arquitectura.
- Indicar el o los patrones de integración recomendados y justificar mencionando aspectos críticos del caso.

Para este caso, consideramos adecuado el uso de una arquitectura SOA mediante conectores de mensajería asíncrona de tipo punto-a-punto, para el caso de la comunicación buque → base, y publish/subscriber, para el caso de la comunicación base → buque.

La elección de la mensajería asíncrona se relaciona con la necesidad de garantizar la fiabilida, la integridad en el envío de los mensajes. Por otro lado, la elección de conectores de tipo publish/subscriber tiene que ver con que cada buque se puede suscribir a las zonas de su interés para recibir reportes meteorológicos regionales personalizados.





Para ambos casos, los conectores punto-a-punto y los de tipo publish/subscriber, el patrón de integración a utilizar será el de middleware.

Caso 6: **FinTech**

Un emprendimiento que surgió como una pequeña iniciativa “de garage” está incrementando su cartera de clientes. El núcleo de su actividad consiste en suministrar información a sistemas de inversores acerca del desempeño histórico de diferentes compañías que cotizan en la bolsa. Como en algunos casos estos servicios de información son personalizados, es restricción obligatoria (mandatory) que las fuentes estén aisladas entre sí (al menos desde un enfoque lógico). En algunos casos, estos informes pueden ser una agregación de servicios propios o de terceros. Es deseable que la solución permita alta flexibilidad en función de cambios en los requerimientos de los clientes.

Más info: https://www.pwc.com/us/en/financial-services/publications/viewpoints/assets/pwc-fsi-what-is-fintech.pdf

Se pide:

- Diseñar y documentar una arquitectura.

- Mencionar las posibles alternativas de conectores de integración indicando ventajas y desventajas de cada una.

En este caso, la necesidad de tener distintas vistas, personalizadas, para cada uno de los servicios de información, sumado al deseo que la solución sea altamente flexible, nos hace volcarnos por una arquitectura de tipo MVC (Model View Controller).