
**UNIVERSIDAD DE BUENOS AIRES**

**Departamento de Computación**

**Facultad de Ciencias Exactas y Naturales**

# Ingeniería del Software II

## Trabajo Práctico II - 2da entrega

**22 de junio de 2015**



| Integrante | LU | Correo electrónico |
|---|---|---|
| Taravilse, Leopoldo | 464/08 | ltaravilse@gmail.com |
| Langberg, Martín | 86/10 | martinlangberg@gmail.com |
| Claverino, Daniel | 273/10 | dclave@gmail.com |
| Conde, Fernando | 423/09 | ferconde87@gmail.com |

# Escenarios de Uso

Diferenciamos entre *datos internos*, *datos externos* y *datos de auditabilidad*. *Internos* son aquellos que se busca almacenar en los servidores locales (los datos sensibles por ejemplo). Los *externos* son aquellos que se pretende guardar en los servidores externos o privados. Los logs y toda información de auditabilidad, a pesar de guardarse en los servidores externos, los distinguimos aparte como *datos de auditabilidad*.

Definimos estos términos porque es posible que haya que almacenar *datos internos* en los servidores externos y viceversa, y a veces es necesario actuar distinto para cada caso. Por ejemplo, que los *datos internos* se guarden externamente requiere algún grado de seguridad.

También utilizamos el concepto de "frecuencia de uso" de un dato. Manteniendo una ventana de 30 días, definimos la *cantidad de uso* de un dato como la suma entre la cantidad de accesos y la de modificaciones realizadas dentro de la ventana. Dicho esto, decimos que un *dato interno* es de "uso frecuente" si la *cantidad de uso* del mismo es mayor o igual a la del 70% de los *datos internos* existentes. De la misma forma, un *dato externo* se considera de "uso frecuente" si el porcentaje es de 80%.

## Disponibilidad

**01** Durante la etapa de migración el sistema debe maximizar la disponibilidad de los *datos externos*, que pueden ser difíciles de obtener debido a las fallas de comunicación.

| | |
|---|---|
| Fuente | Interno al sistema. |
| Estímulo | Acceso o modificación de un *dato externo*. |
| Entorno | Etapa de migración de servidores. |
| Artefacto | Controlador de Información. |
| Respuesta | Se devuelve o modifica el dato. |
| Medida de Respuesta | Si el dato está comprimido en los servidores locales, la operación es exitosa el 100% de los casos. En caso contrario, la operación es exitosa en al menos un 60% de los casos, siendo 60% el porcentaje de disponibilidad durante la etapa de migración asegurado en el acuerdo con los servidores privados. |

**02** Se quiere que ante una notificación de poco espacio disponible en los servidores locales, algunos de los *datos internos* que se solicitan con baja frecuencia se compriman para liberar recursos.

| | |
|---|---|
| Fuente | Servidores locales. |
| Estímulo | Notificación de poco espacio. |
| Entorno | Normal y Etapa de migración de servidores. |

| | |
|---|---|
| Artefacto | Optimizador de Recursos. |
| Respuesta | Se comprimen algunos de los *datos internos* de menor uso. |
| Medida de Respuesta | Los servidores locales no rechazan operaciones por falta de espacio en el 99.99% de los casos. |

**03** Ante una notificación de poco espacio disponible en los servidores locales, y teniendo la mayoría de los *datos internos* de poco uso ya comprimidos, se quiere que se los traslade a los servidores privados para liberar recursos.

| | |
|---|---|
| Fuente | Servidores locales. |
| Estímulo | Notificación de poco espacio. |
| Entorno | Datos internos con poco margen de compresión. |
| Artefacto | Optimizador de Recursos. |
| Respuesta | Se trasladan algunos de los *datos internos* ya comprimidos a los servidores privados. |
| Medida de Respuesta | Los servidores locales no rechazan operaciones por falta de espacio en el 99.999% de los casos. |

**04** Se quiere que a medida que se piden *datos externos* durante la etapa de migración de servidores, se guarde localmente una copia comprimida para aumentar su disponibilidad.

| | |
|---|---|
| Fuente | Interna al sistema. |
| Estímulo | Acceder o modificar un *dato externo* de uso frecuente que no se encuentra de manera local. |
| Entorno | Etapa de migración de servidores. |
| Artefacto | Direccionador de Pedidos. |
| Respuesta | Se busca el dato en los servidores externos y se guarda comprimido en los servidores locales. |
| Medida de Respuesta | Al menos el 60% de las operaciones tiene éxito, siendo 60% el porcentaje de disponibilidad durante la etapa de migración asegurado en el acuerdo con los servidores privados. |

**05** Si el canal puede informar el envío fallido de un mensaje, ante dicho evento el sistema debe reintentar el envío más adelante.

| | |
|---|---|
| Fuente | Comunicador de Canales. |
| Estímulo | El mensaje no se envió. |
| Entorno | Normal y “Etapa de migración de servidores” |
| Artefacto | Servicio de mensajería. |

| | |
|---|---|
| Respuesta | Se intenta el reenvío del mensaje en incrementos de 1 hora (1h la 1ra vez, 2hs la 2da, 5hs la 5ta) hasta un tope de 8 horas. Si el mensaje no se envía pasados 10 días, descartarlo. |
| Medida de Respuesta | El 70% de los mensajes que fallaron alguna vez terminan enviandose correctamente. |

**06** En la zona patagónica no se puede garantizar una conectividad permanente de la red de datos, por lo que para los envíos de mensaje de la zona se utilizará una red de datos dedicada con mejor disponibilidad apenas se encuentre en línea.

| | |
|---|---|
| Fuente | Externa al sistema. |
| Estímulo | Envío de mensaje hacia la zona patagónica. |
| Entorno | Red dedicada en línea. |
| Artefacto | Dispositivos Receptores. |
| Respuesta | El mensaje se envía utilizando la red dedicada. |
| Medida de Respuesta | La tasa de mensajes recibidos exitosamente al usar la red dedicada supera la tasa teórica utilizando la red original. |

## Performance

**01** Pasada la etapa de migración, los *datos externos* guardados localmente deben almacenarse sin compresión en los servidores privados para mejorar la performance del sistema.

| | |
|---|---|
| Fuente | Interno al Sistema. |
| Estímulo | Acceso o modificación de un *dato externo* que existe en los servidores locales. |
| Entorno | Normal. |
| Artefacto | Direccionador de Pedidos. |
| Respuesta | Actualizar el dato en los servidores privados a partir de la información local y eliminar el dato localmente. |
| Medida de Respuesta | Aumenta el throughput de las operaciones usando *datos externos* que estaban guardados localmente. |

**02** Como durante la etapa de migración de servidores es posible que haya que realizar procesos adicionales sobre los datos (comprimir/descomprimir, encriptar/desencriptar) se quiere reducir el tiempo de procesamiento dedicado a la auditabilidad para que la performance no se vea tan impactada.

| | |
|---|---|
| Fuente | Interno al Sistema. |
| Estímulo | Loguear una acción. |

| | |
|---|---|
| Entorno | Etapa de migración de servidores. |
| Artefacto | Logger. |
| Respuesta | Ignorar el pedido. |
| Medida de Respuesta | El throughput del logging es máximo y los recursos dedicados al proceso son mínimos. |

**03** Se quiere que el sistema procese los pedidos de acceso o manipulación de la información de manera rápida.

| | |
|---|---|
| Fuente | Externo al sistema. |
| Estímulo | Acceso o manipulación de información. |
| Entorno | Normal. |
| Artefacto | Sistema. |
| Respuesta | Se devuelve o modifica la información. |
| Medida de Respuesta | El promedio de la latencia es de 500 milisegundos. |

**04** Se quiere obtener información masiva de monitoreo de campañas de manera rápida.

| | |
|---|---|
| Fuente | Externo al sistema |
| Estímulo | Pedido de información de monitoreo de campañas. |
| Entorno | Normal. |
| Artefacto | Sistema. |
| Respuesta | Se devuelve la información más certera disponible. |
| Medida de Respuesta | Se responde en menos de 5 segundos. |

## Seguridad

**01** Se debe mantener un log de las acciones importantes que realiza un usuario autorizado del sistema.

| | |
|---|---|
| Fuente | Usuario autorizado. |
| Estímulo | Modifica, genera o elimina algún ente del sistema (campaña, receptores, etc). |
| Entorno | Normal. |
| Artefacto | Logger. |

| | |
|---|---|
| Respuesta | Se loguea el identificador del usuario y el identificador del ente junto con la acción realizada (creación/alta/baja/modificación) y la fecha y hora de la misma. |
| Medida de Respuesta | Puede identificarse el causante de alguna acción importante en el 100% de los casos. |

**02** Los *datos internos* almacenados en los servidores externos deben estar seguros ante lecturas no autorizadas.

| | |
|---|---|
| Fuente | Individuo o ente desconocido. |
| Estímulo | Intento de lectura de un *dato interno* en los servidores externos. |
| Entorno | Etapa de migración de servidores. |
| Artefacto | Dato interno. |
| Respuesta | El dato está encriptado, por lo que la lectura falla. |
| Medida de Respuesta | La encripción no puede romperse usando una computadora hogareña corriendo un algoritmo de fuerza bruta por al menos 100 años. |

**03** Para evitar brute-forcing de las contraseñas de los usuarios del sistema, se quiere que ante consecutivos intentos fallidos de autorización desde un mismo origen se bloqueen todos los pedidos del sistema desde dicho origen. Dicho bloqueo se levantará luego de un tiempo, que se incrementará con cada aplicación.

| | |
|---|---|
| Fuente | Autorizador. |
| Estímulo | Intento fallido de autorización. |
| Entorno | Normal y Etapa de migración de servidores. |
| Artefacto | Firewall. |
| Respuesta | Ante 10 fallos consecutivos se bloquean los pedidos al sistema. La 1ra vez por 1h, la 2da por 3hs, la 3ra por 6hs, la 4ta por un día y de la 5ta en adelante por una semana. |
| Medida de Respuesta | Las contraseñas de los usuarios del sistema no pueden ser obtenidas por brute-forcing en un tiempo razonable. |

## Certeza de Datos

**01** Durante la etapa de migración debe evitarse lo más posible la desincronización total entre los datos de los servidores privados y sus contrapartes locales.

| | |
|---|---|
| Fuente | Interno al Sistema. |
| Estímulo | Acceso o modificación de un dato guardado en los servidores privados que existe en los servidores locales. |

| | |
|---|---|
| Entorno | Etapa de migración de servidores. |
| Artefacto | Sistema. |
| Respuesta | Mantener un proceso Sincronizador que cada tanto intente comunicarse con el servidor externo y sincronice los datos. |
| Medida de Respuesta | Los datos externos no están más de un 90% desincronizados. |

**02** Se quiere que la información de monitoreo de campaña sea lo más certera posible.

| | |
|---|---|
| Fuente | Interno al Sistema. |
| Estímulo | Pasan 30 minutos. |
| Entorno | Normal o Etapa de migración de servidores. |
| Artefacto | Compilador de Info de Monitoreo. |
| Respuesta | Se busca y procesa la información de monitoreo de campaña. |
| Medida de Respuesta | La información de monitoreo disponible es correcta para una hora atrás como mucho. |

## Modificabilidad

**01** El usuario desea agregar, modificar o eliminar entes (campañas, receptores, mensajes, etc).

| | |
|---|---|
| Fuente | Usuario. |
| Estímulo | Agregar, modificar o eliminar ente. |
| Entorno | En tiempo de ejecución. |
| Artefacto | Sistema. |
| Respuesta | La operación es exitosa salvo en el caso en que los datos provistos no sean válidos |
| Medida de Respuesta | El tiempo total requerido es menor a 2 minutos. |

**02** Pueden agregarse rápidamente nuevos canales de comunicación (del tipo Whatsapp) si estos utilizan los mismos o menos datos que los ya implementados (nombre, teléfono, etc) y usan la red de datos.

| | |
|---|---|
| Fuente | Diseñador. |
| Estímulo | Agregar nuevo canal de comunicación. |

| | |
|---|---|
| Entorno | En tiempo de ejecución. |
| Artefacto | Sistema |
| Respuesta | El canal está listo para usarse. |
| Medida de Respuesta | El tiempo requerido para agregar y testear el nuevo canal es menor a 12 HH. |

**03** Los proveedores de contenido pueden diseñar e implementar métricas de evaluación local.

| | |
|---|---|
| Fuente | Consumidor de sistema. |
| Estímulo | Diseñar o implementar métricas de evaluación local. |
| Entorno | En tiempo de ejecución. |
| Artefacto | Sistema. |
| Respuesta | La operación es exitosa. |
| Medida de Respuesta | El tiempo requerido para agregar y testear es menor a 4 HH. |

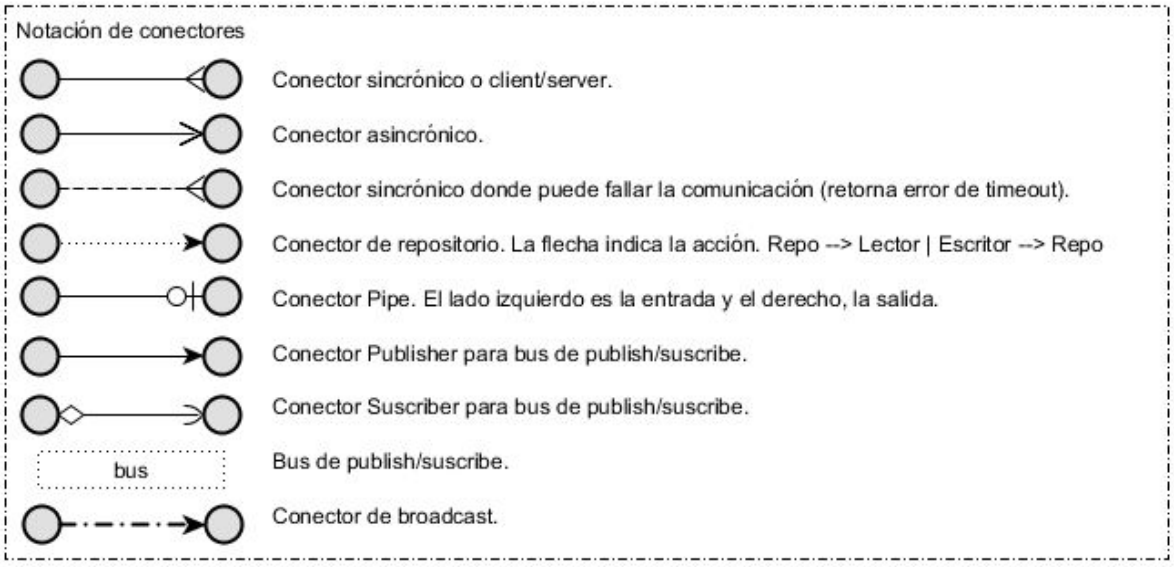
## Usabilidad

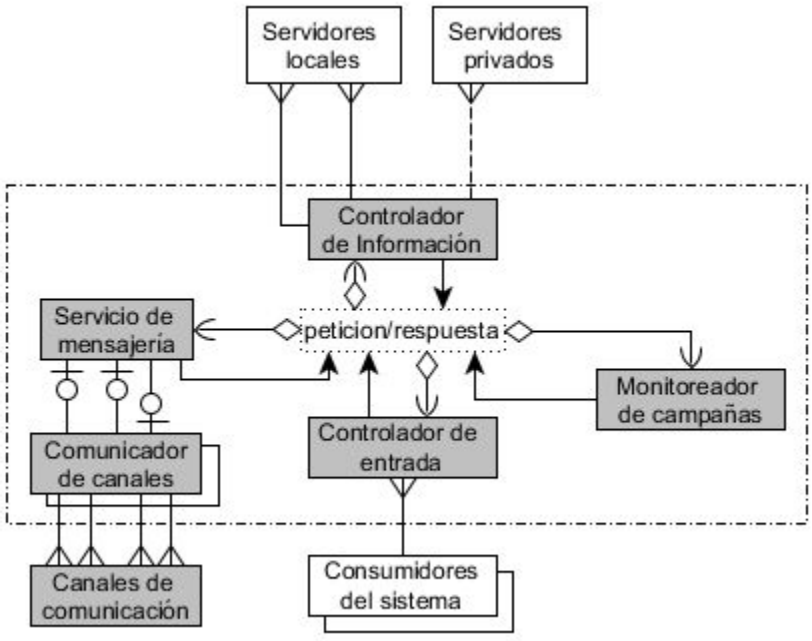
**01** Los receptores deben poder desuscribirse con facilidad de los avisos que quiera.

| | |
|---|---|
| Fuente | Consumidor de sistema. |
| Estímulo | Desuscribirse de los avisos que el usuario no considera importantes. |
| Entorno | Operación normal. |
| Artefacto | Sistema. |
| Respuesta | El usuario logra desuscribirse de los avisos que desea satisfactoriamente. |
| Medida de Respuesta | 95% de los usuarios pueden desuscribirse de los avisos que desean en menos de 30 segundos. |

# Arquitectura TP2

Para explicar la arquitectura, mostraremos primero un esquema general de los componentes y luego iremos entrando en sus detalles, explicando las interacciones entre ellos.




La idea es que los *Consumidores* interactúen de forma cliente/servidor con el *Controlador de Entrada*. Este componente se suscribe al *Bus de petición/respuesta* para recibir mensajes de tipo “respuesta de información” y “respuesta de monitoreo”. Según el pedido, y si se dan las condiciones de seguridad pueden pasar dos cosas:

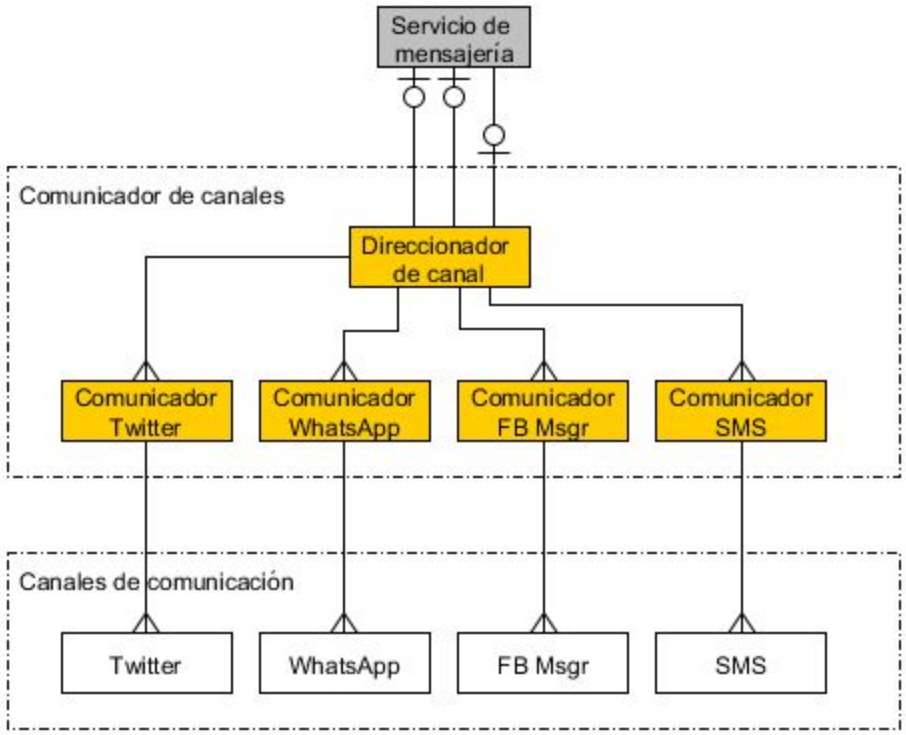
1. Se procede a buscar, crear o modificar información en los servidores. Para ello, se pushea al *Bus* un mensaje de tipo “información”, con un identificador para poder saber a qué consumidor le corresponde la respuesta.

2. El pedido tiene que ver con el monitoreo de las campañas, por lo que se pushea al *Bus* un mensaje de tipo “monitoreo”, también con algún tipo de identificación.

El *Servicio de Mensajería* se encarga del envío de mensajes y recepción de las respuestas usando algún canal de comunicación. Respecto al envío, pushea al *Bus* “mensajes pendientes de envío” y se suscribe a “respuesta de mensajes pendientes de envío”. Para la parte de recepción, pushea “respuestas de mensajes”.

Se comunica ida y vuelta con varios *Comunicadores de Canales* usando pipes. La idea es que se puedan hacer varios envíos de mensaje a la vez para ganar en performance. El pipe hacia el *Servicio de Mensajería* sirve para informar el envío fallido de un mensaje cuando el canal lo permita.

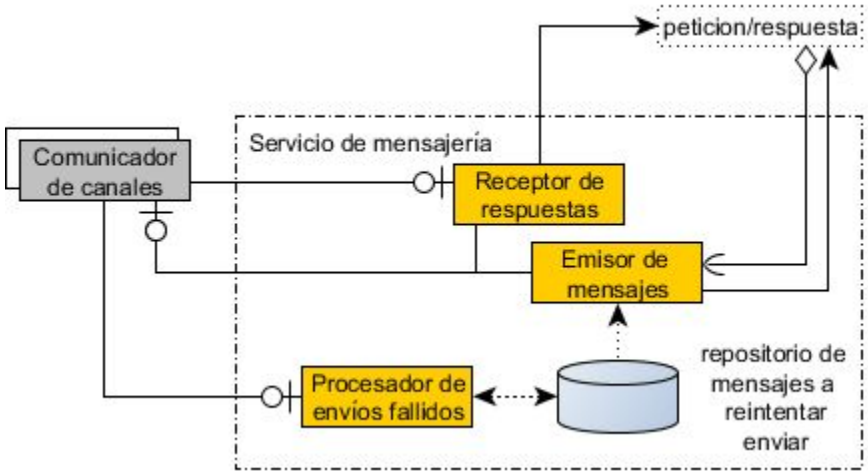
Hay cuatro puertos en los Canales de Comunicación debido a los cuatro canales iniciales (FB Messenger, WhatsApp, SMS y Twitter).




La idea de usar un *Direccionador de Canal* es la de servir como interfaz para el *Servicio de Mensajería*, y mejorar así la modificabilidad respecto a los canales según lo definido en el Escenario M02. La comunicación es cliente-servidor, y como mencionamos antes, si es posible que el canal informe del envío fallido de un mensaje, se le avisará al *Servicio de Mensajería*.

Se busca que se envíen varios mensajes a la vez, no repetirlos, por lo que el *Direccionador* de un *Comunicador de Canales* particular busca un pedido en el pipe y se lo queda, sin compartirlo con los demás.

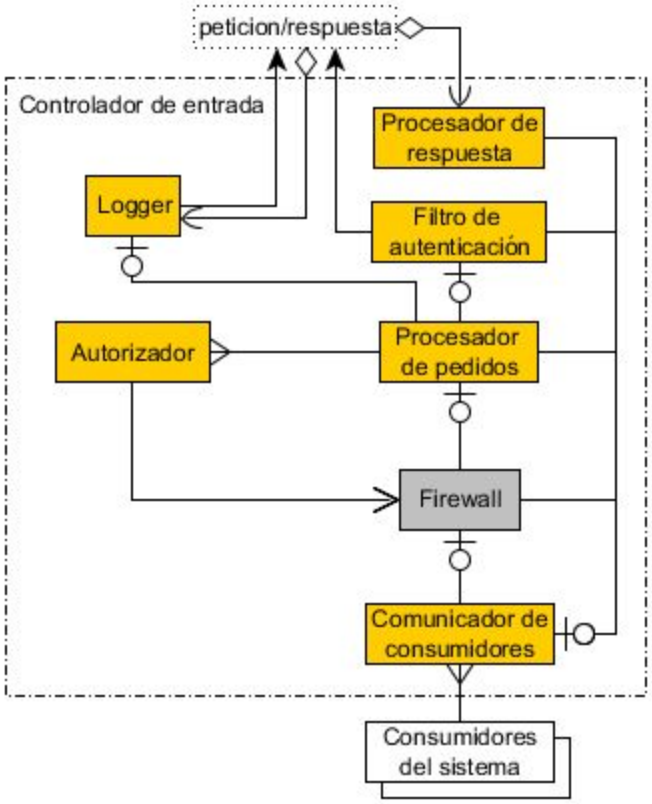
Otro pedido que se le puede hacer al direccionador es el de obtener respuestas a un mensaje. De encontrarse, el Direccionador las pushea en otro pipe hacia el Servicio de Mensajería.




Como adelantamos, el *Emisor de Mensajes* pushea al *Bus* “mensajes pendientes de envío” y se suscribe a “respuesta de mensajes pendientes de envío”. Cada cierto tiempo, puede buscar los mensajes a enviar y pasárselos al pipe de los *Comunicadores de Canales*, así como reenviar mensajes que hayan fallado previamente.

El *Receptor de Respuestas* cada tanto se comunica con los Comunicadores de Canales para que le avisen ante nuevas respuestas a los mensajes. En tal caso, pushea al *Bus* las respuestas obtenidas.

Ante un envío fallido, el *Procesador de Envíos Fallidos* se encarga de mantener el *repositorio de mensajes a reintentar enviar*, según lo definido en el Escenario D05.



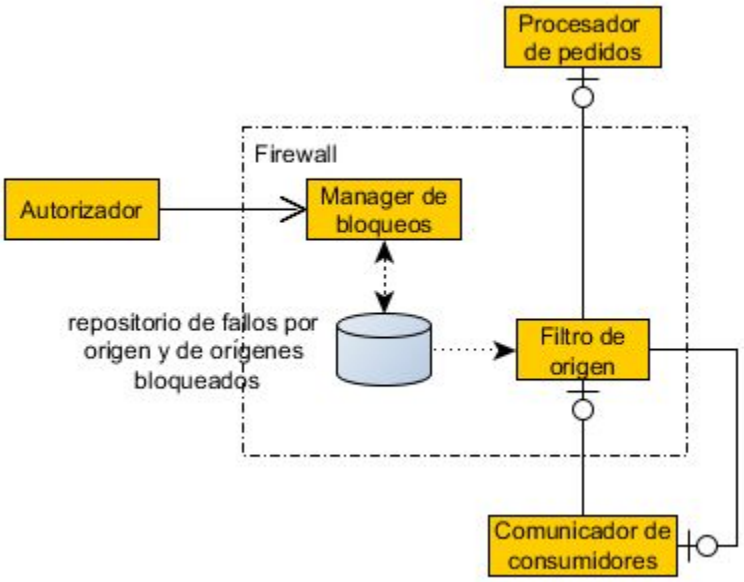

Pasemos al *Controlador de Entrada*. El *Comunicador de Consumidores* se encarga de atender las consultas de los *Consumidores* y responderlas. Ante un pedido, que es

filtrado por el *Firewall*, llega al *Procesador de Pedidos*. Allí puede tratarse de un pedido de autenticación, que es resuelto por el Autorizador. En caso de fallo, se lo comunicará al Firewall.

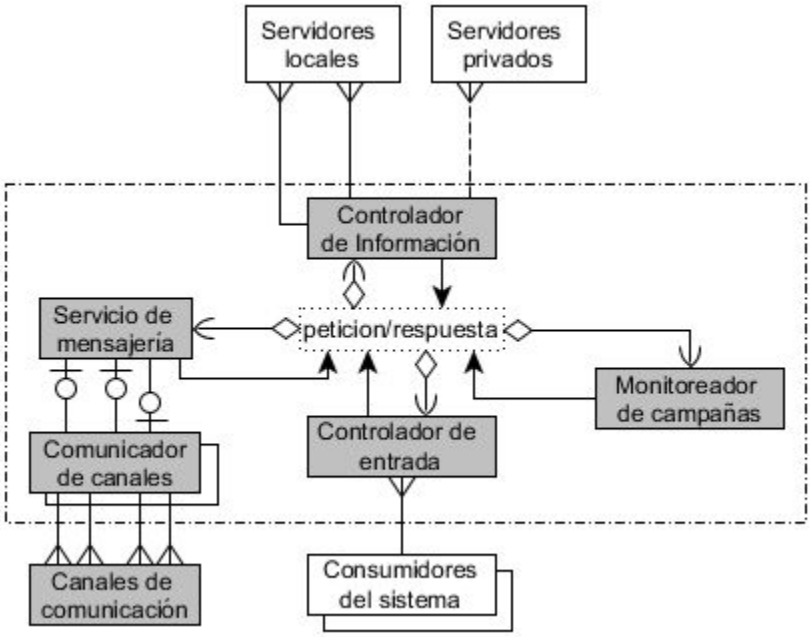
Para auditarse los pedidos de alta/baja/modificación según el Escenario S01 existe el *Logger*, que se encargará de publicar al *Bus* el pedido de loggeo. El Logger también se suscribe a mensajes de tipo “estado de migración”. Si el *Bus* le avisa que la migración empezó, el *Logger* simplemente no hará pushes al *Bus* hasta que le avise que la etapa finalizó, como detalla el Escenario P02.

Finalmente, el pedido debe pasar por un *Filtro de Autenticación*, que dejará pasar aquellos que sean válidos desde un punto de vista de seguridad. A partir de ahí, se pusheará al Bus como evento de “información”, “monitoreo”, o “estado de migración”. El primer caso indica un tipo de ABM de algún dato del sistema (campañas, receptores, etc), mientras que el segundo indica un pedido de monitoreo de campañas. El tercer caso ocurre cuando un consumidor debidamente autentificado quiere hacer saber al sistema que empezó o terminó la etapa de migración de servidores.

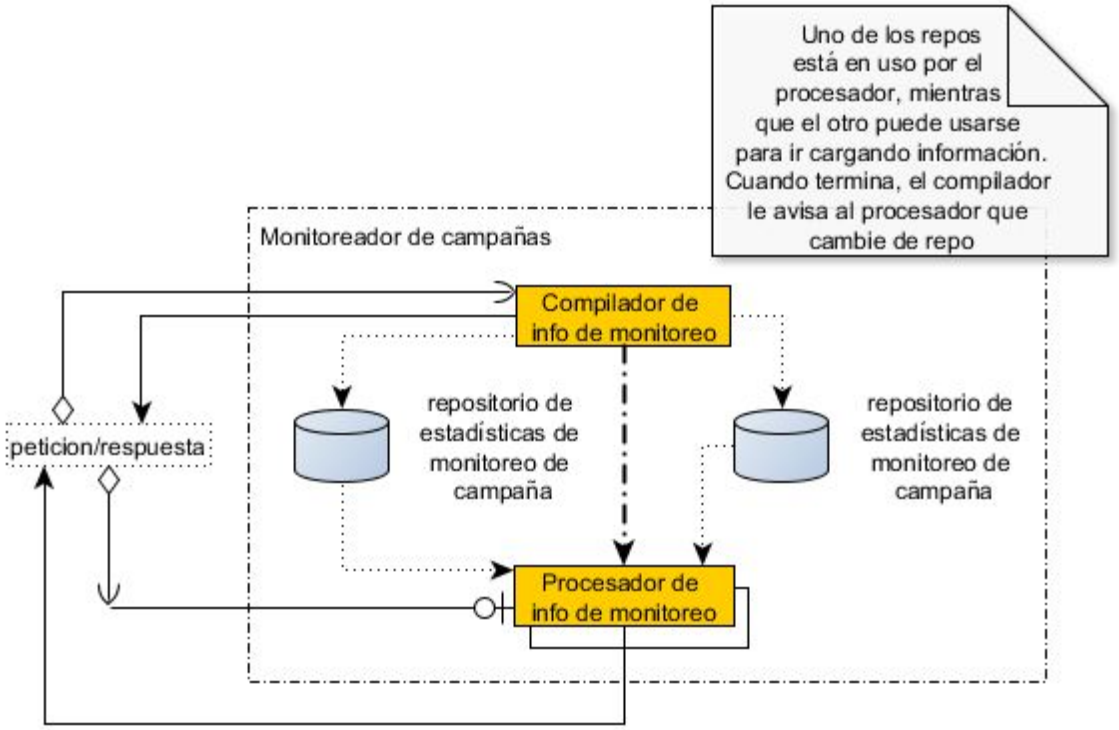
El *Procesador de Respuesta* estará pendiente del *Bus* y se suscribirá a mensajes del tipo “respuesta de información” y “respuesta de monitoreo”, las cuales hará saber al *Comunicador de Consumidores*. Lo mismo ocurre ante cualquier filtro que rechace el pedido. De esta forma, el *Consumidor* siempre tiene una respuesta, ya sea un error o la que esperaba.




El *Firewall* se encarga de cumplir con el Escenario S03. Para ello, utiliza un repositorio con la información de fallos y los orígenes bloqueados, con las fechas de caducidad apropiadas. El *Manager de Bloqueos* espera un mensaje del *Autorizador* para registrar el origen de la autenticación fallida y, en caso de ser repetirse, aplicar el bloqueo correspondiente.




Volviendo al esquema general, nos falta explicar el *Monitoreador de Campañas* y el *Controlador de Información*.




Se buscaba proveer un monitoreo rápido de las campañas, pero tratándose de un volumen masivo de datos, esto entra en conflicto con la performance. Como mencionamos en la parte de Escenarios, queremos que sea rápido y nos permitimos un grado de incerteza.

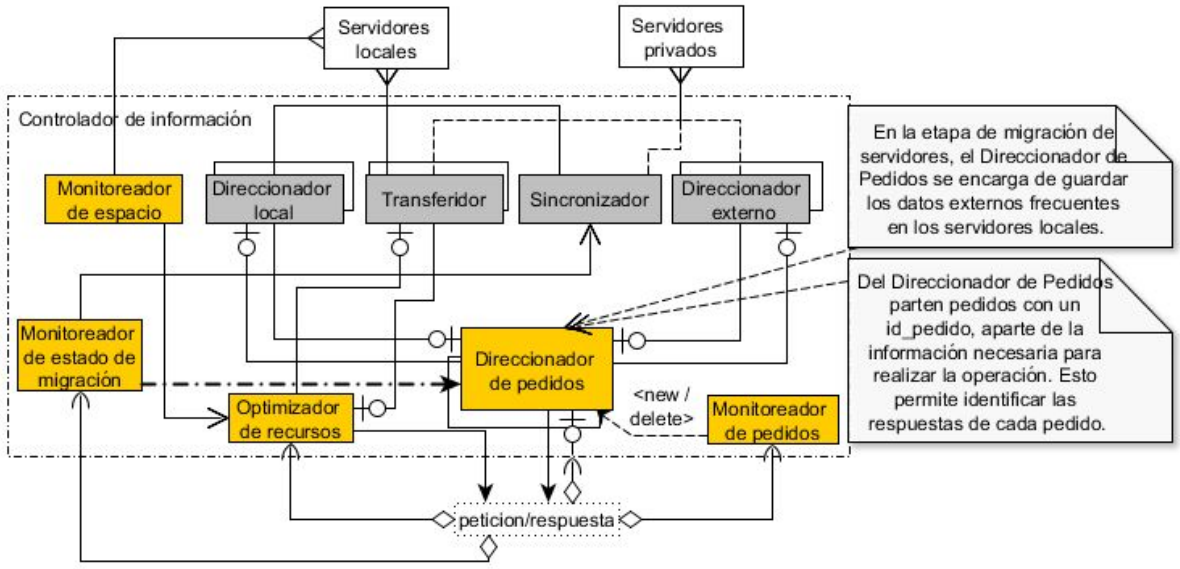
El Monitoreador de Campañas se encarga de cumplir con los Escenarios C02 y P04. Para ello, cuenta con dos repositorios locales, los cuales uno será consumido

constantemente por varios componentes *Procesador de Info de Monitoreo*, mientras el otro es completado con la información de campaña hasta cierto momento.

El *Procesador* utiliza un pipe, que está suscrito a los eventos de tipo “monitoreo”. A medida que llegan pedidos, cada *Procesador* tomará uno y lo sacará de la cola para que nadie más lo resuelva. Busca la información pedida en el repositorio en uso, y la pusheará al *Bus* como evento de tipo “respuesta de monitoreo”.

El *Compilador de Info de Monitoreo* se ejecuta cada 30 minutos, como especifica el Escenario C02. Pushea pedidos de “información de monitoreo” al Bus, y se suscribe a “respuesta de información de monitoreo”. A medida que va recibiendo resultados, los va compilando en el repositorio que no está en uso. Al terminar la compilación, avisa a todos los *Procesadores* que cambien el repositorio que están usando.

Utilizar dos repositorios permite que no esté disponible la información “a medias” mientras se está buscando, sólo de forma completa.




Pasando a la parte de manejo de los datos, tenemos los componentes *Direccionador de Pedidos*, que se encargan de ir leyendo y sacando del pipe los eventos enviados por los demás componentes, como “información”, “información de monitoreo”, “mensajes pendientes de envío”, “respuestas de mensaje”, buscar y publicar las respuestas con el tipo acorde.

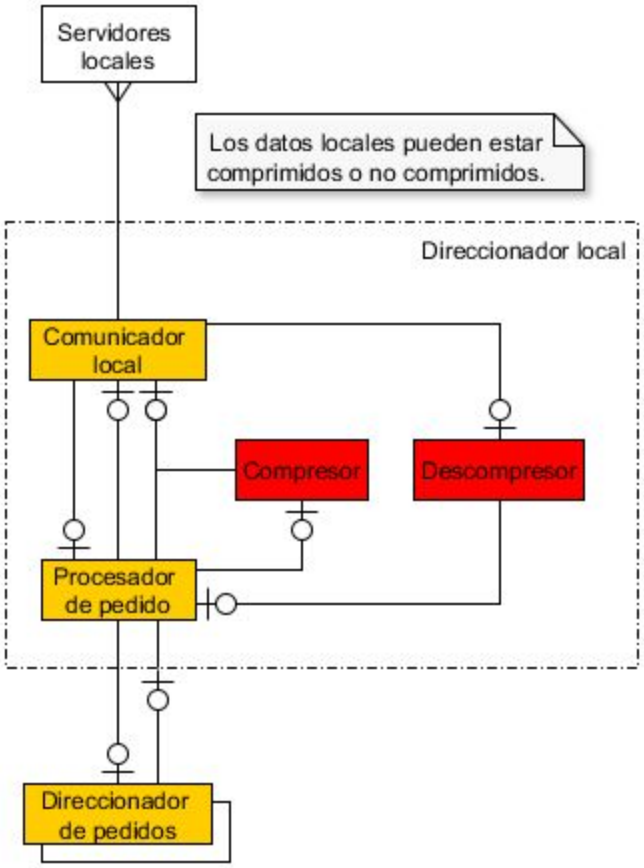
El *Monitoreador de Estado de Migración* se suscribe a eventos de “estado de migración”, y es el encargado de informar al *Sincronizador* y a los *Direccionador de Pedidos* el cambio de estado de la migración para que actúen acordes al mismo.

El *Monitoreador de Pedidos*, por otro lado, se suscribe a los mismos eventos que el *Direccionador* y a los tipos de eventos que publica. Su función es la de observar la tasa de pedidos/respuestas en el *Bus que* afectan al *Controlador de Información*, y crear o eliminar *Direccionadores de Pedidos* según se necesiten.

El *Direccionador de Pedidos* según el tipo de dato pedido sabe si mandar la consulta al *Direccionador Local* o al *Direccionador Externo*. También sabe que en caso de no encontrarlo en el servidor correspondiente tiene que buscarlo en el otro. Maneja internamente un id de pedido, para saber cuál es el resultado de una consulta y es el encargado de cumplir los Escenarios D01 y P01.

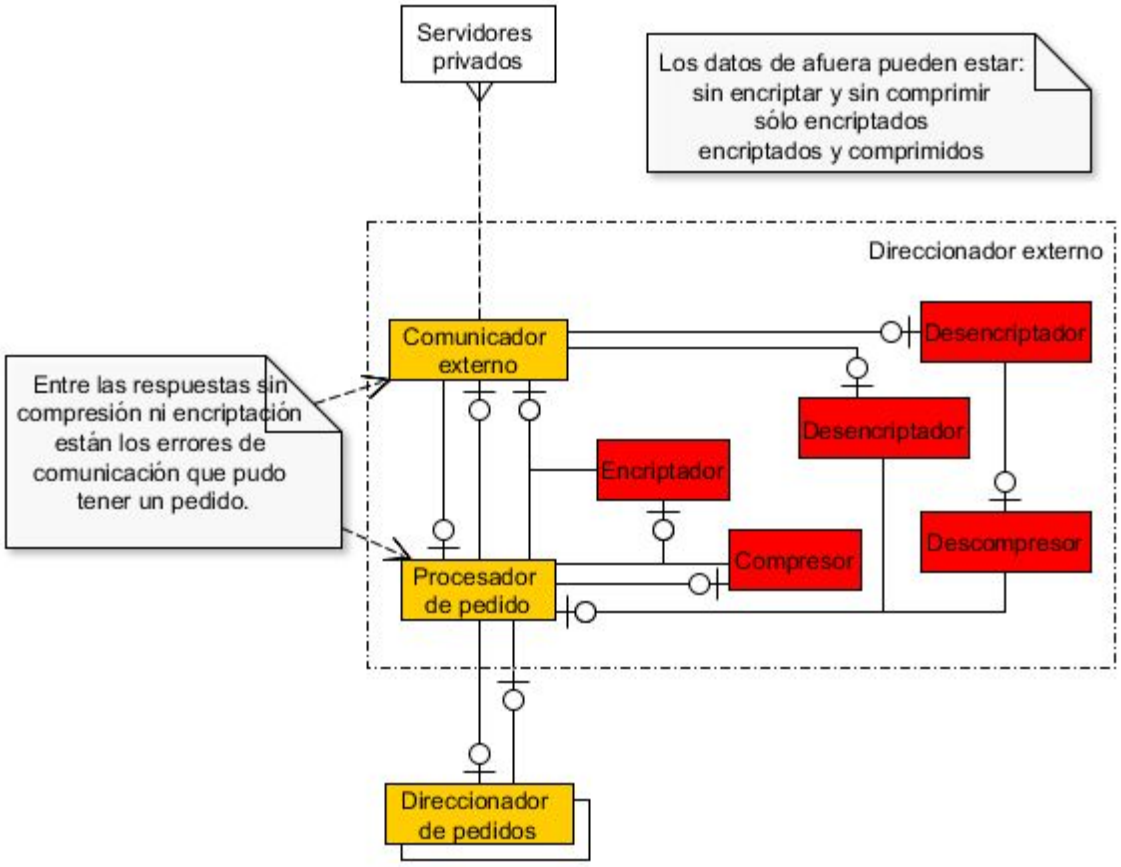
Los *Direccionadores Locales* y *Externos* están para comunicarse con los *Servidores Locales* y *Privados*, respectivamente. La concurrencia apunta al cumplimiento del Escenario P03, permitiendo procesar varios pedidos a la vez. Hay que observar que el conector con los *Servidores Privados* es un cliente/servidor donde puede fallar la comunicación.

El *Monitoreador de Espacio* se encarga de revisar que los *Servidores Locales* estén trabajando en un margen seguro. De notar que hay poco espacio disponible, se le avisa al *Optimizador de Recursos*, quien se encarga de cumplir los Escenarios D02 y D03. Para ello, pide información al *Bus* de tipo “estadística” acerca de la cantidad de *datos internos* comprimidos, y *datos externos* frecuentes. Una vez que obtiene la respuesta, organiza un plan de reestructuración de datos y comprime *datos internos*, o bien los traslada a los servidores externos mediante los componentes *Transferidor*.



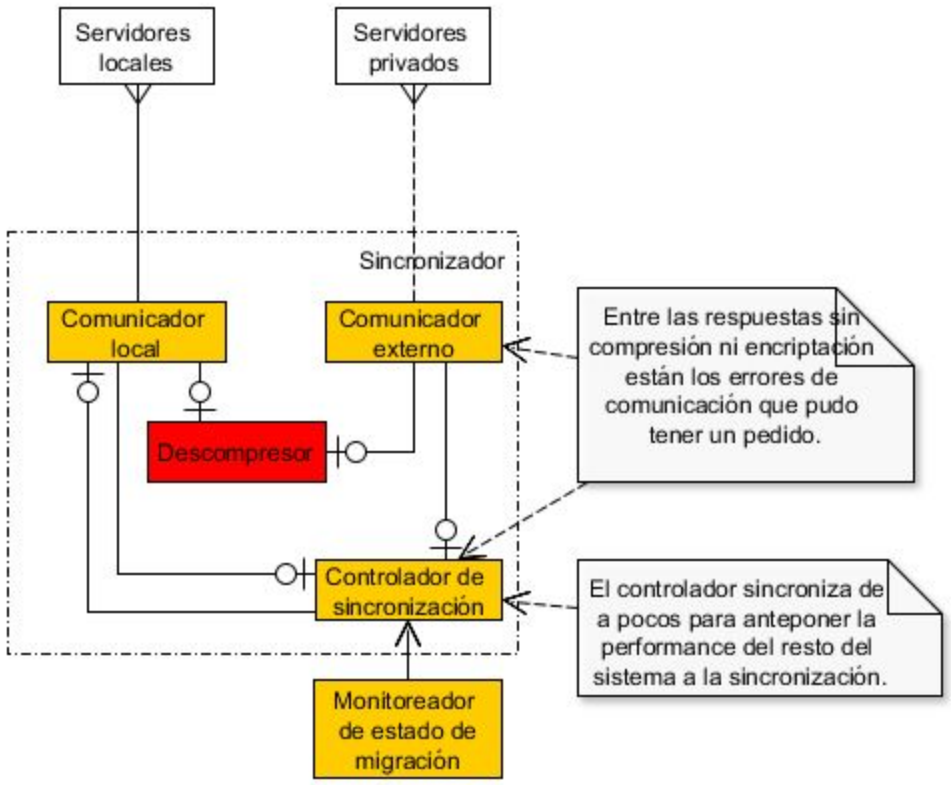

El *Direccionador Local* trata los pedidos a los servidores locales. Si se trata de almacenar un dato, el *Procesador de Pedido* puede necesitar comprimirlo, o no. Si se trata de una consulta a los servidores, se pasa directamente al *Comunicador Local*.

La respuesta puede requerir descomprimirse, o devolverse directamente al *Procesador*, quien la mandará al *Direccionador*.




El *Direccionador Externo* es similar, pero más complejo. Su *Procesador de Pedido*, aparte de mandar el dato plano, puede requerir encriptarlo, o bien comprimirlo y encriptarlo. A su vez, el *Comunicador Externo* tiene que permitir el camino inverso y, dado un dato, desencriptarlo, o bien desencriptarlo y descomprimirlo. También es posible que el *Comunicador* quiera devolver un dato plano, la respuesta a una consulta, o un mensaje de error por fallo de comunicación con los servidores, lo que se hace utilizando el mismo pipe.

Los *datos externos* no se comprimen, pero los *datos internos* podrían hacerlo, por lo que la compresión requiere de su posterior encriptación.

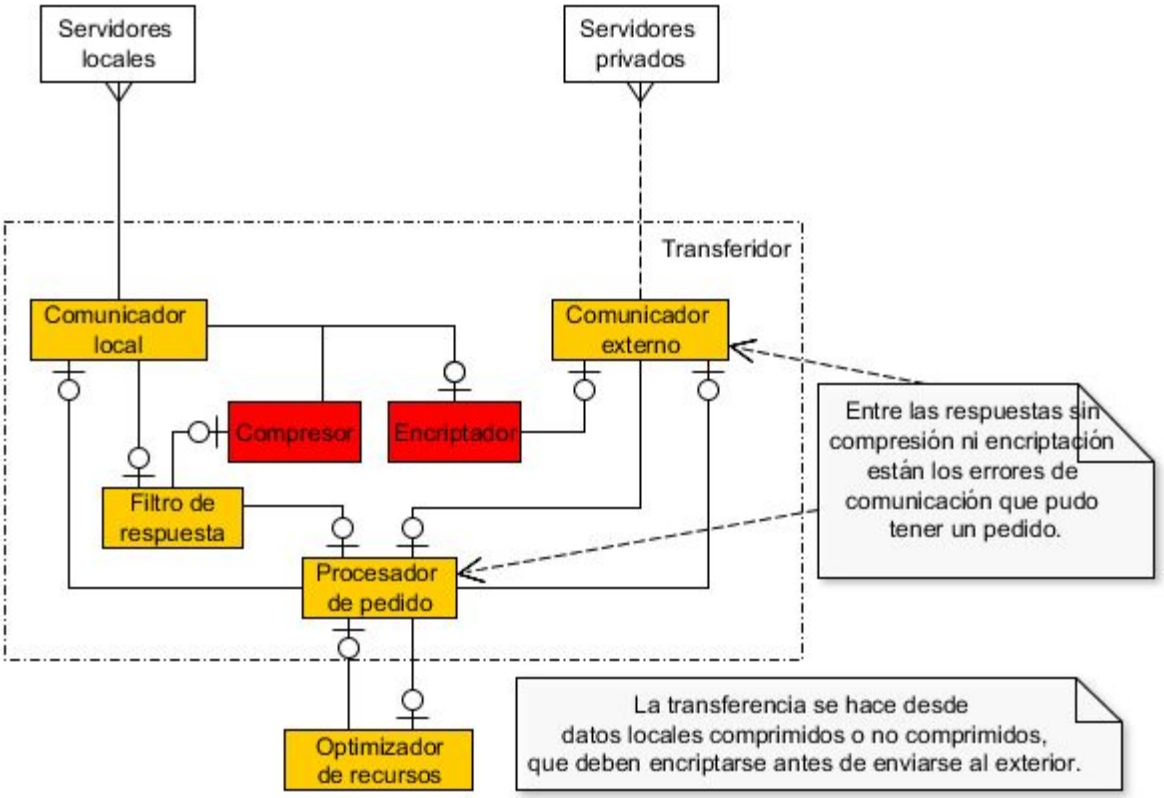



El *Sincronizador* es el encargado de cumplir el Escenario C01. El *Controlador de Sincronización* funciona sólo durante la Etapa de Migración de Servidores y corre con prioridad baja, anteponiendo la performance general del sistema la sincronización entre servidores.

Busca información del estado de los *datos externos* usando *Comunicador Local*. Luego cuando le pide algún dato al *Comunicador*, éste lo devuelve por el Descompresor, quien se lo pasa al *Comunicador Externo* para actualizar el dato en los *Servidores Privados*. Al ser un *dato externo* en los *Servidores Locales*, por cómo lo describimos al hablar de los Escenarios, siempre estará comprimido, y como estos datos en los servidores externos se encuentran planos, es necesario descomprimirlos antes de subirlos.

Ante un error de comunicación, el *Controlador* puede decidir reintentarlo o buscar otros datos para subir.

El *Monitoreador de Estado de Migración* le hará saber al *Controlador* del, valga la redundancia, estado de la migración a medida que cambia.



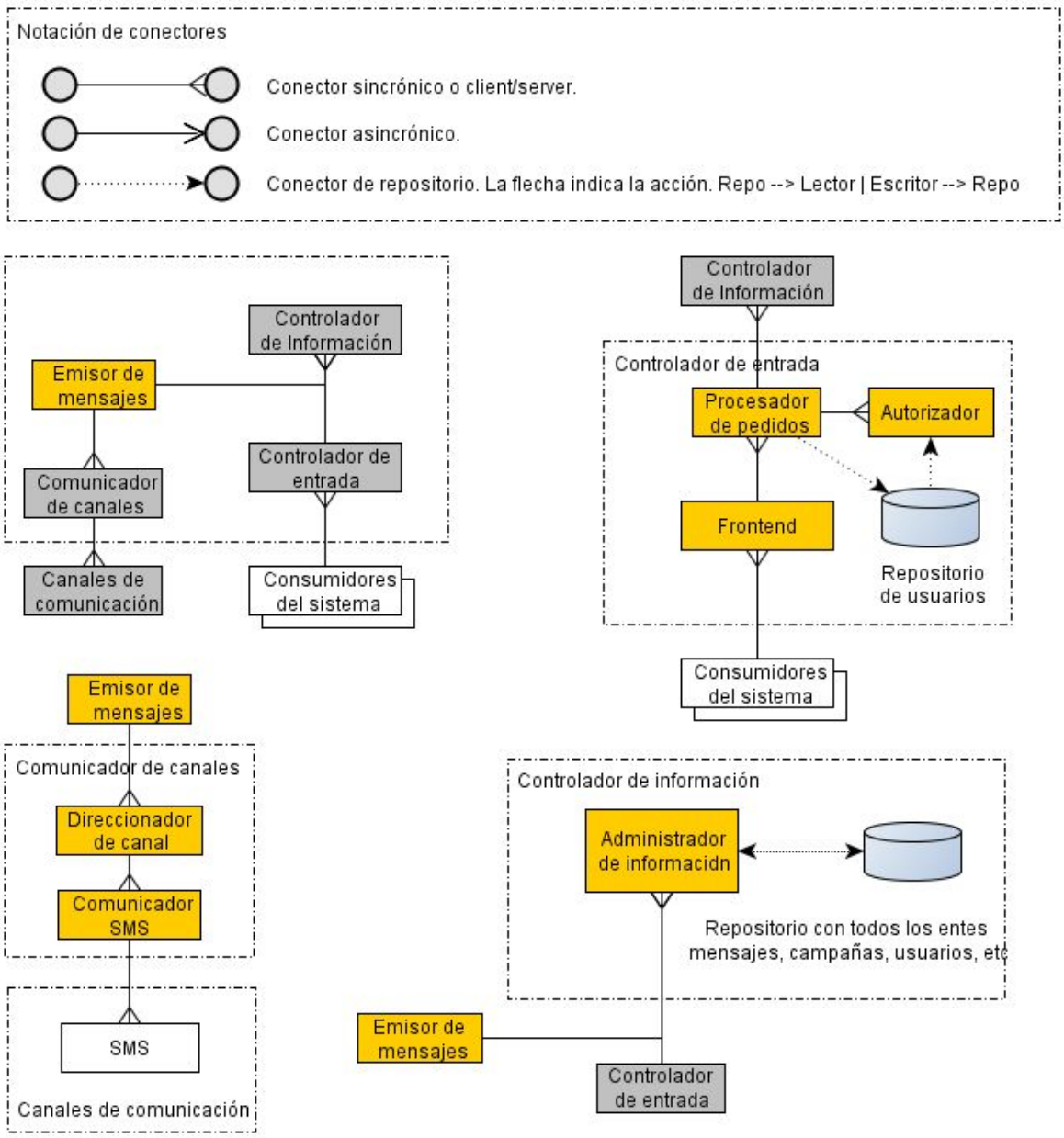

Finalmente, el *Transferidor*. Este componente se encarga de mover los datos desde los servidores locales y eliminarlos una vez se haya confirmado la transferencia a los *Servidores Privados*.

El *Procesador de Pedido* tomará una consulta del pipe (identificada con un id de pedido) y se la quedará para que ningún otro *Transferidor* la procese. Acto seguido, la buscará utilizando *Comunicador Local*. Ya vimos que este último puede devolver una respuesta plana, o bien un dato comprimido. De estar compreso, sólo falta encriptarse para enviarse a los servidores externos, siguiendo lo indicado en el Escenario S02. De ser una respuesta plana, el *Filtro de Respuesta* verifica si se trata de un dato, en cuyo caso se comprime y encripta antes de transferirse.

El *Comunicador Externo* podrá avisar de un fallo en la comunicación al *Procesador*, el cual se lo hará saber al *Optimizador de Recursos* mediante el id de pedido. Si no hubo fallo en la comunicación, el *Procesador* debe eliminar el dato localmente. En tal caso, la respuesta del *Comunicador Local* no sería un dato, sino la del pedido de borrado, por lo que se pasará al *Procesador de Pedido* y posteriormente se informará al *Optimizador* de que la transferencia fue exitosa.

.

# Arquitectura TP1




Evidentemente, la arquitectura del primer TP es mucho más simple. Algunos componentes se comparten, pero sus detalles son más sencillos:

La falta de servidores externos y, por lo tanto, de problemas de comunicación simplifica muchísimo el detalle del *Controlador de Información*.

Como el volumen de datos era muy bajo, no era necesario tener buses y todo podía resolverse con calls sincrónicos o client/server.

La parte de mensajería en el primer TP admitía una cierta facilidad para agregar otros canales de comunicación aparte del *SMS*, pero en principio sólo admitía éste.

No había auditabilidad de ningún tipo, ni necesidad de evitar el brute forcing de las contraseñas de los usuarios por lo que el *Controlador de Entrada* no requiere de un *Firewall* ni un *Logger*.

Finalmente, el *Monitoreo de las Campañas* no se basaba en inmensos volúmenes de datos, por lo que podía ser resuelto desde el *Controlador de Entrada* con varios llamados al *Controlador de Información*.

# Comparación y conclusiones

## Breve comparación de los métodos usados (UP vs Ágil)

El método *Unified Process* (UP) consiste en una versión iterativa incremental con elementos similares a los del tradicional método de desarrollo en cascada, agregando como atributo principal la iteración continua de las tareas en cuatro etapas esenciales: Inicio del Proyecto, Elaboración, Construcción y Transición. La intensidad de cada tarea (Modelado del Negocio, Requerimientos, Diseño, etc) varía de acuerdo a la etapa.

En cada etapa toma especial importancia la contención de los riesgos identificados en el Inicio del Proyecto, y el cumplimiento de la arquitectura diseñada, expresada en las diferentes *viewtypes*.

En el método Ágil, en cambio, la prioridad de las tareas no depende de los posibles riesgos, sino que se realizan equitativamente en cada iteración en forma de pequeños “pasos”, con reuniones periódicas al final de cada iteración (que suele tomar una o dos semanas), y los grupos no superan las 4 o 5 personas.

En términos de la planificación, también existen importantes diferencias. Las tareas en UP se diagraman en base a los riesgos considerados en la planificación, los casos de uso y en el QAW, mientras que en el método ágil estas tareas se diagraman a partir de las *User Stories* definidas por los *stakeholders*. Luego, la estimación de la duración de cada tarea se calcula en “Horas Hombre” (UP), a diferencia del método ágil donde se utilizan los *Story Points* para describir la complejidad de la tarea y no la duración específica, y sin una planificación rigurosa de la responsabilidad de cada desarrollador sobre la tarea (esto se define en cada iteración).

## “Programming in the small" vs. "Programming in the large”

El Diseño OO trata con una visión más detallada del sistema a crear, mientras que en la Arquitectura es más general. En el primer TP, modelamos entidades del mundo real, que luego se trasladaban muy directamente a código. En este TP, si bien no entramos en la Vista Modular, ésta trata más de módulos y su relación, no tanto de los detalles como qué representan o qué mensajes pueden responder.

Otro punto a destacar es la aparición de Atributos de Calidad y sus Escenarios. En este TP vimos que existen atributos no funcionales que se pueden describir en términos testeables y que pueden impactar en la Vista de Componente y Conector, y observarse en cierta forma desde la misma. En principio no se nos ocurre cómo estos Atributos y Escenarios se visualizarían en un Diseño OO.

## Conclusiones

Para hacer algo masivo como YouTube o Facebook es necesario pensar en grande, y ahí es importante la visión de Arquitectura. Si en cambio se busca un público más reducido, o una aplicación más sencilla, el Diseño OO podría ser suficiente.

Cabe destacar que no son excluyentes, es posible especificar usando tanto la Arquitectura como el Diseño OO; es más, podríamos decir que la Arquitectura está arriba del Diseño OO, como una capa que nos abstrae de sus detalles y nos permite observar “desde más arriba” cómo se comportaría nuestro sistema ante diversos escenarios.

Por otro lado, respecto a los métodos de desarrollo, ambos (y varios más) son utilizados en el mundo real y la elección puede ser afectada por el *scope* del proyecto. Adoptar metodologías ágiles con grupos de cientos de personas, o bien UP en proyectos chicos puede ser contraproducente. Por eso es importante elegir el método que mejor nos ayude a construir un producto de calidad en tiempo y forma.