ADR- Patrones de sincronización basados en eventos

ADR-BACKEND-EDA-SINCRO

<table>
<tr><td>Estado</td><td>VIGENTE</td></tr>
<tr><td>Partes interesadas</td><td>AREA GOBIERNO TECNOLOGICO</td></tr>
<tr><td>TAGS</td><td></td></tr>
<tr><td>Tomada el</td><td>
<table>
<tr><th>PRE-RELEASE</th><th>RELEASE</th><th>RETIRO</th></tr>
<tr><td>13 may 2024</td><td>20 jun 2024</td><td>-</td></tr>
</table>
<table>
<tr><th>Version</th><th>Date</th><th>Author</th><th>Comment</th></tr>
<tr><td>20</td><td>06-04-02026</td><td>ADMINISTRADOR 6</td><td>Actualización automática de enlaces Jira/Confluence: ws001 -> nuevos dominios</td></tr>
</table>
</td></tr>
<tr><td>Responsable</td><td>AREA GOBIERNO TECNOLOGICO</td></tr>
<tr><td>Solución Afectada</td><td>NUEVOS DESARROLLOS, PAES</td></tr>
<tr><td>Recursos Asociados</td><td>Spike

⚠ OTARQ-2139 - El proyecto de Jira no existe o no tienes permiso para verlo.</td></tr>
<tr><td></td><td>
<table>
<tr><th>Responsable</th><th>Aprobado por</th><th>Consultados</th><th>Informados</th></tr>
<tr><td>AREA GOBIERNO TECNOLOGICO</td><td></td><td>CARLOS GONZALEZ SANTIAGO

ALBERTO FUENTES GOMEZ

LUIS MARTINEZ FONTIVEROS</td><td>DESARROLLO SOFTWARE RRHH (AYESA)

DESARROLLO SOFTWARE ECONOMICO-FINANCIERO (NTTDATA)

DESARROLLO SOFTWARE ASISTENCIAL (NTTDATA)

AREA SOLUCIONES DIGITALES

AREA GOBIERNO TECNOLOGICO</td></tr>
</table>
</td></tr>
</table>

Asunto a Decidir	Se necesita tomar una decisión sobre cuál será el mecanismo de sincronización entre sistemas de información que requieren mantener una coherencia en el dato sincronizado, asumiendo los mecanismos de **consistencia eventual** .
	
	Los requisitos no funcionales que esta solución debe asegurar son los siguientes:
	
	• **Resiliente**: Se debe garantizar la entrega de la información actualizada (no puede haber pérdida de datos en la sincronización)
	• **Transaccional:** En caso de error como vuelvo al estado inicial.
	• **Idempotente**: Un cambio en los datos debe tener siempre la misma consecuencia
	• **Obsolescencia de la sincronía**: Es importante mantener la última versión del dato.
	• **Desacoplamiento:** implica que los componentes de un sistema están diseñados para interactuar entre sí con la menor cantidad de dependencias directas posible.
	• **Compatible con la Consistencia Eventual**
	• **Compatible con las tecnologías actuales** definidas en la arquitectura de referencia
	  ◦ Baremetal
	  ▪ Web Logic 12.2
	  ▪ Jdk 8
	  ▪ Oracle 19c
	  ◦ Contenedores
	  ▪ Open Liberty 20.0.0.12
	  ▪ Microprofile 3.3
	  ▪ Jdk 11
	  ▪ Oracle 19c

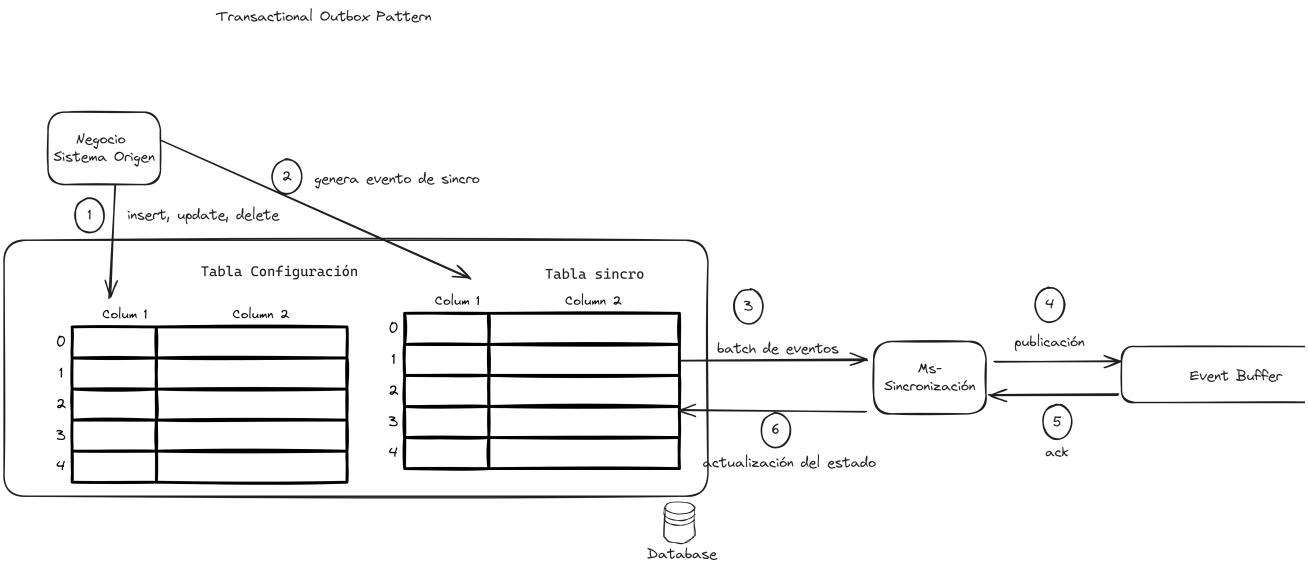
Identificación de la decisión	**Describe brevemente la decisión arquitectónica tomada.** La decisión final dependerá de factores adicionales del proyecto, donde se aplicarán los mecanismos descritos en este Architecture Decision Record. Entre los aspectos clave a considerar, se incluye: • No refactorizar el código legacy.

Contexto	Este caso de uso se divide en dos fases:
	• Fase 1: Generación del cambio
	  ◦ Persistir el cambio en el sistema origen
	• Fase 2: Sincronización
	  ◦ Sistema origen genera un evento de sincronización en base al cambio producido
	  ◦ Propagación del evento de sincronización
	  ◦ Persistir la sincronización en el/los sistema/s destino
	Hay diferentes escenarios que se ven impactados por esta sincronización:
	• Aplicaciones de escritorio
	• Aplicaciones Web Baremetal (WebLogic)
	• Aplicaciones Web Contenedores (OKD)
	Dentro de la gestión de los proyectos está el aspecto económico:
	• Con canal de para modificación
	• Sin canal para modificación

Alternativas Consideradas

Transactional Oubox Pattern

Este patrón implica escribir los eventos en una tabla de "outbox" dentro de la misma transacción que la operación de negocio. El negocio (*aplicación, servicio, microservicio*) que hace la persistencia (*nueva inserción, actualización o borrado*) y después en la misma transacción persiste el cambio en la BBDD. Otro servicio del mismo negocio debe leer esa tabla de eventos (técnica polling cada n ms) e ir generando eventos de sincronización a un bus y este se consume y se persiste el cambio en el sistema destino.



Transactional OutBox Pattern

Evaluación de pros/contras

Pros para cualquier tipo de proyecto
Asegura la creación del evento de sincronización. Esto es debido a la transacción local que se realiza al persistir el cambio en la tabla y su posterior persistencia en la tabla de eventos a publicar.
No hay una comunicación directa desde el sistema origen con el event buffer. Esto aporta varias ventajas: • Se puede contar con un microservicio reutilizable e independiente del sistema origen.
Pros para proyectos ya existentes

En relación a la comunicación del sistema origen con el event buffer, se aportan varias ventajas adicionales a los proyectos existentes: • No es necesario modificar el código del proyecto existente para tener esta funcionalidad,
Contras para cualquier tipo de proyecto
Implica cambios en la base de datos, se debe crear una tabla para persistencia de los eventos a generar.
Hay riesgo de impacto en la performance de la bbdd del sistema, al realizar consultas cada intervalo de tiempo x.
Contras para proyectos ya existentes
Implica cambios en el código para persistir en la tabla de eventos

Evaluación de requisitos

Requisito	OK/NOK	Detalle
Resiliente	OK	• Hay que implementar de mecanismos de fault-tolerance en cada uno cada paso • Al hacer uso de una tabla auxiliar y un event buffer puedes tener la garantía de entrega ante caídas. • Si el **Sistema Origen** se cae, al estar persistida la necesidad de sincronización, se puede volver a retomar desde la última sincronización correcta. • El event buffer al tener persistencia de mensajes y posibilidad de ack de entrega puedes garantizar simplemente la recepción o la lectura del mismo por algún consumidor según se defina la necesidad de confirmación de entrega.

Transaccional	OK	Se establecen los siguientes tramos transaccionales: • **Persistencia del cambio:** La persistencia tanto en la tabla origen como en la tabla de evento, se puede realizar como transacción local. • **Generación de Eventos:** El event buffer tiene diferente estrategias en la garantía de entrega.
Idemponente	OK	• Hay que asegurarse de que los eventos en la outbox se marquen como procesados. • Necesita lógica adicional para asegurar idempotencia
No invasivo - Código	NOK	Hay que hacer desarrollos para la implementación de la emisión de los eventos.
No invasivo - BBDD	NOK	• Hay que crear una nueva tabla que persista los eventos que después un servicio leerá para publicar. • Definir estrategia de lectura, en batch o individual. • Definir el tiempo de polling (ms).
Compatible con las tecnologías actuales	OK	Al no añadir ninguna tecnología extra al (la conexión con el event buffer es a través de un nuevo microservicio) no se aprecian incompatibilidades.

Trigger en BBDD

Los triggers son bloques de código que se ejecutan automáticamente en respuesta a ciertos eventos en la base de datos, como inserciones, actualizaciones o eliminaciones de registros. En términos de resiliencia y recuperación, Oracle ofrece varios mecanismos para manejar errores y garantizar la integridad de las transacciones en las que participan los triggers. A continuación, se detallan algunos aspectos clave:

1. Atomicidad y Consistencia

- **Atomicidad**: Los triggers en Oracle se ejecutan como parte de la transacción que los activa. Esto significa que si un trigger falla, la transacción completa se revierte, garantizando que los cambios no se apliquen parcialmente.
- **Consistencia**: Los triggers deben mantener la consistencia de los datos. Si una operación del trigger genera un error, la transacción completa se revierte para mantener la integridad de la base de datos.

2. Mecanismos de Manejo de Errores

- **Excepciones en PL/SQL**: Dentro de un trigger, se pueden manejar errores utilizando bloques `EXCEPTION`. Esto permite capturar errores específicos y tomar acciones adecuadas, como registrar el error en una tabla de auditoría, enviar notificaciones, o realizar limpiezas necesarias.

- **RAISE_APPLICATION_ERROR**: Permite generar errores personalizados dentro de un trigger, lo que puede ser útil para manejar situaciones específicas y proporcionar mensajes de error más claros.

3. Logs y Auditorías

- **Registros y Auditorías**: Los triggers pueden incluir lógica para registrar eventos y errores en tablas de auditoría. Esto proporciona un historial de eventos y errores que puede ser analizado posteriormente.

4. Garantías de Ejecución

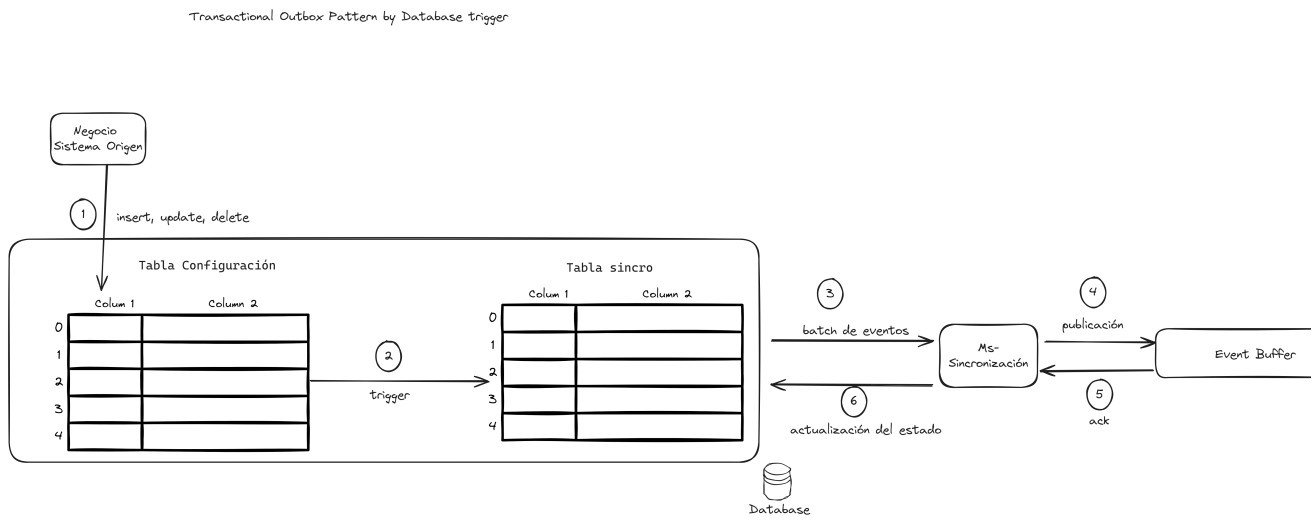
- **Control de Estados**: Oracle garantiza que un trigger se ejecuta en el estado adecuado de la transacción. Por ejemplo, los triggers BEFORE se ejecutan antes de que se realice la operación DML, y los triggers AFTER se ejecutan después de la operación DML pero antes de que se confirme la transacción.

5. Control de Fallos

- **Reversión Automática**: Si ocurre un error no manejado dentro de un trigger, la transacción entera que activó el trigger se revierte automáticamente. Esto incluye todas las operaciones DML que hayan ocurrido dentro de la transacción.

6. Eventos de Recuperación

- **Eventos de Reintento**: En algunos escenarios, es posible implementar lógica para reintentar operaciones fallidas. Aunque no es un mecanismo nativo de los triggers, se puede implementar mediante programación adicional, como paquetes PL/SQL que manejen la lógica de reintento.



Transactional Outbox Pattern by Database trigger

Evaluación de pros/contras

Pros para cualquier tipo de proyecto

Asegura la creación del evento gracias a su atomicidad y consistencia
No hay una comunicación directa desde el sistema origen con el event buffer. Esto aporta varias ventajas: • Se puede contar con un microservicio reutilizable e independiente del sistema origen.
Pros para proyectos ya existentes
No requiere cambio en el código del proyecto para la generación de los eventos.
En relación a la comunicación del sistema origen con el event buffer, se aportan varias ventajas adicionales a los proyectos existentes: • No es necesario modificar el código del proyecto existente para tener esta funcionalidad.
Contras para cualquier tipo de proyecto
Implica cambios en la base de datos, se debe crear una tabla para persistencia de los eventos a generar.
Hay riesgo de impacto en la performance de la bbdd del sistema, al realizar consultas cada intervalo de tiempo x.

Evaluación de requisitos

Requisito	OK/NOK	Detalle
Resiliente	OK	• Hay que implementar de mecanismos de fault-tolerance en cada uno cada paso • Al hacer uso de una tabla auxiliar y un event buffer puedes tener la garantía de entrega ante caídas. • Si el **Sistema Origen** se cae, al estar persistida la necesidad de sincronización, se puede volver a retomar desde la última sincronización correcta. • El event buffer al tener persistencia de mensajes y posibilidad de ack de entrega puedes garantizar simplemente la recepción o la lectura del mismo por algún consumidor según se defina la necesidad de confirmación de entrega.

Transaccional	OK	Se establecen los siguientes tramos transaccionales: • **Persistencia del cambio:** La persistencia tanto en la tabla de origen como en la tabla de evento, se puede realizar como transacción local. • **Generación de Eventos:** El event buffer tiene diferente estrategias en la garantía de entrega.
Idemponente	OK	• Hay que asegurarse de que los eventos en la outbox se marquen como procesados. • Necesita lógica adicional para asegurar idempotencia
No invasivo - Código	OK	No requiere cambio de código
No invasivo - BBDD	NOK	• Requiere la creación de una tabla para persistir los eventos de sincronización • Requiere la creación de un trigger
Compatible con las tecnologías actuales	OK	Al no añadir ninguna tecnología extra al (la conexión con el event buffer es a través de un nuevo microservicio) no se aprecian incompatibilidades.

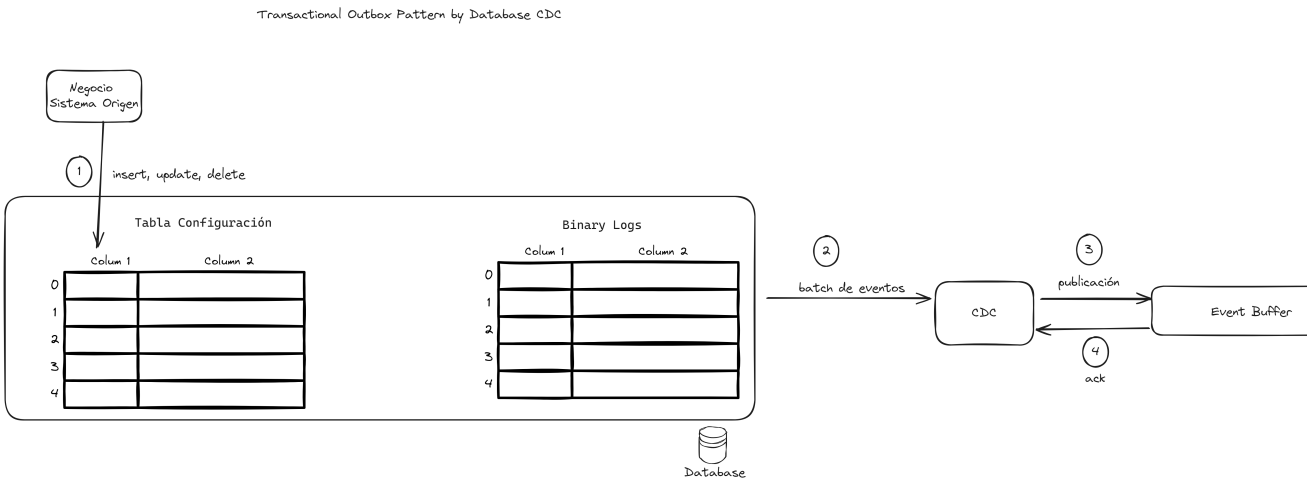
CDC (Debizum)

La captura de datos de cambios (CDC) es un patrón de datos que permite producir eventos a partir de los cambios en un almacén de datos.

CDC proporciona un camino para obtener las ventajas de las arquitecturas basadas en eventos reduciendo el esfuerzo de extraer bases de datos de aplicaciones.

La implementación de CDC dependerá de su base de datos, sus casos de uso y el conector que seleccione.:

- **JDBC**: sondean continuamente la base de datos en busca de cambios en las tablas de origen especificadas.
  - *Sondear una tabla mutable:* puede no ser óptimo, ya que inevitablemente perderá actualizaciones que se completen entre los intervalos de sondeo.
  - *Sondear una tabla inmutable*.
    - La primera es simplemente hacer que la tabla sea inmutable y cambiar el código de la aplicación para manejar esta vista de los datos.
    - La segunda opción es adjuntar un disparador a la tabla mutable primaria. Este activador añade una segunda entrada a una tabla de auditoría inmutable. Luego, el conector puede sondear esta tabla de auditoría para detectar cambios sin que falten registros. En este caso, para ORACLE la opción es el GOLDEN-GATE que no está dentro de la licencia disponible para el SAS



Transactional Outbox Pattern by Database CDC

Evaluación de pros/contras

Pros para cualquier tipo de proyecto
Garantiza la creación del evento de sincronización a través de los mecanismos de Change Data Capture
No hay una comunicación directa desde el sistema origen con el event buffer, si no que se realiza a través del CDC
No tienes que desarrollar ningún microservicio para la publicación de eventos. La propia herramienta de CDC lo hace.
Minimizas el impacto en BBDD
Permite extracciones de información mas complejas, abarcando varias tablas por ejemplo.
Se externaliza la gestión de algunos aspectos de funcionamiento, a destacar: • At-least-once delivery • Idempotencia
Pros para proyectos ya existentes
No requiere cambio en el código del proyecto para la generación de los eventos.
Contras para cualquier tipo de proyecto

En caso de que la base de datos de sincronización sea Oracle implica:

- La activación del CDC de forma nativa en la BBDD (ORACLE GOLDEN GATE y CDC ORACLE CONFIG) está fuera de soporte por lo que se tiene que hacer uso de otro sistema externo (Debizum)
- añadir una pieza adicional OpenLogReplicator
- implica añadir algunas configuraciones extras
- La solución al no ser nativa, y sobre una tecnología que no es opensource (ORACLE), aumenta el riesgo de incompatibilidades

Requisito	OK/NOK	Detalle
Resiliente	OK	• Hay que implementar de mecanismos de fault-tolerance en cada uno cada paso • Depende de la robustez de la herramienta de CDC • Altamente resiliente, ya que captura los cambios directamente del log de transacciones • Si el **Sistema Origen** se cae, al estar persistida la necesidad de sincronización, se puede volver a retomar desde la última sincronización correcta. • El event buffer al tener persistencia de mensajes y posibilidad de ack de entrega puedes garantizar simplemente la recepción o la lectura del mismo por algún consumidor según se defina la necesidad de confirmación de entrega.

Transaccional	OK	Se establecen los siguientes tramos transaccionales:
		• **Persistencia del cambio:**
		  ○ Captura cambios de manera transaccional porque lee directamente del log de transacciones.
		  ○ Puede tener un ligero retraso en la captura de eventos
		• **Generación de Eventos:** El event buffer tiene diferente estrategias en la garantía de entrega.

Idemponente	OK	Partiendo de la base del uso de DeBizum, este garantiza la idempotencia a través de los siguientes mecanismos: • Uso de identificadores únicos de mensajes (UUID). • Registro de mensajes procesados para evitar duplicados. • Confirmaciones de recepción (ACKs) para asegurar la entrega y procesamiento. • Persistencia de mensajes para asegurar la recuperación en caso de fallos.
No invasivo - Código	OK	• No requiere cambios en el código de la aplicación • Necesita configurar y mantener la herramienta de CDC.
No invasivo - BBDD	A MEDIAS	• No requiere cambios en la estructura de la base de datos. • Requiere la creación de un trigger

Compatible con las tecnologías actuales	A MEDIAS	Al ser Oracle el motor de BBDD mayoritaria en el SAS se encuentran los siguientes problemas:
		• Se necesita el uso de debizum
		  ◦ Es compatible con single instance y rac
		  ◦ Limitaciones de compatibilidad con OracleDecisión tomada, a falta de confirmación y aclaraciones al equipo, así como a falta de la redacción de las HU's adicionales en base a las necesidades indicadas en la sección de Consecuencias.
		  ▪ Logminer: Deprecated
		  ▪ XStreams: unsoported
		  ▪ OpenLogReplicator: Según Debezium : "The OpenLogReplicator ingestion adapter is currently in incubating state, i.e. exact semantics, configuration options etc. may change in future revisions, based on the feedback we receive."
		  • Ficha técnica Debezium-OpenLogReplicator:
		  ◦ Database: 12c, 19c, 21c
		  ◦ Driver: 12.2.x, 19.x, 21.x
		  ◦ OpenLogReplicator: 1.3.0
		  ◦ Supported versions: 11.2, 12.1, 12.2, 18c, 19c, 21c.
		  ◦ Supported editions: XE, SE, SE2, PE, EE
		  ◦ Database must be in single instance mode (non RAC)
		  ◦ The database must be working in ARCHIVELOG mode and having enabled MINIMAL SUPPLEMENTAL LOGGING.
		  ◦ Supported platforms: Linux64, Solaris x64, Solaris Sparc
		  ◦ Supported storage: file system (ext4, btrfs, zfs, xfs, sshfs)
		  ◦ Supported database character set, es probable que que esté soportada (estándar UTF está incluido)

Resumen Comparativo de aporte de valor

Criterio	Transactional Outbox	Outbox by Trigger	Outbox by CDC
Resiliente	Alto	Alto	Alto
Transaccional	Alto	Alto	Alto
Idempotente	Medio	Medio	Alto
Minimizar cambio en el código	Medio	Alto	Alto
Minimizar cambio en la estructura	Medio	Medio	Alto
Obsolescencia de la sincronía	Alto	Alto	Alto

Desacoplamiento entre sistemas	Alto	Alto	Alto
Consistencia Eventual	Alto	Alto	Alto
Compatible con tecnologías actuales	Alto	Medio	Alto

Conclusión

- **Transactional Outbox Pattern** es robusto y flexible, pero puede requerir cambios moderados en el código y la estructura de la base de datos.
- **Transactional Outbox Pattern by Database Trigger** minimiza cambios en el código pero puede complicar la administración y depuración debido a la lógica de los triggers.
- **Transactional Outbox Pattern by CDC** ofrece alta resiliencia y minimiza cambios tanto en el código como en la base de datos, pero requiere la gestión de una herramienta de CDC.

El impacto sobre los siguientes requisitos es el mismo para todas las alternativas

Requisito	OK/NOK	Detalle
Desacoplamiento entre sistemas	OK	En todas las alternativas se contempla que la fase de sincronización sea a través de un event buffer por lo que el desacoplamiento con el sistema destino está garantizado siempre y cuando la definición de los eventos de sincronización sean agnósticos al **Sistema Destino**.

Obsolescencia del Dato	NO APLICA	No hay ninguna solución que garantice la correcta gestión de la obsolescencia del dato. Posibles mitigaciones:
		• Uso y transmisión del timestamp
		  ◦ En caso de ser inferior a la última sincronización confirmada se descartaría el evento.
		• Garantía del orden
		  ◦ Esto puede añadir una complejidad no necesaria

Compatible con las tecnologías actuales	OK	En todas las alternativas se contempla que la fase de sincronización sea a través de un event buffer por lo que la compatibilidad de esta pieza aplica a todas por igual: • Versiones de interoperabilidad • Matriz de compatibilidades • Compatibilidad Java • Soporte Cliente Java

Decisión Tomada

La decisión por defecto atendidendo al cumplimiento de RNF y costos de implementación es
CDC

.

Consecuencias

Las consecuencias previstas para CDC :

- Análisis de viabilidad en base a los motores de persistencia a sincronizar.
- Valoración de capacidad de aprovisionamiento en sistemas de la solución CDC identificada.
- Análisis pormenorizado de las necesidades de información a sincronizar.

Estado Actual

Describe brevemente la decisión arquitectónica tomada.

Activa