INFORME TECNICO

**Utilizar RabbitMQ como intermediario de mensajes en las interfaces de datos de Dynamics AX**

# Entendiendo sobre colas de mensajes con RabbitMQ

¿Qué son las colas de mensajes?

Es posible que alguna vez se hayan topado con el nombre *message queue* o su acrónimo *MQ* navegando por la red, si volvemos a los conceptos básicos de la programación de pilas y colas sabremos que es un concepto bastante sencillo donde los emisores producen mensajes y para que estos lleguen a su destinatario deben ser entregados a un intercambiador que los colocará en la cola del respectivo destinatario, finalmente el destinatario puede ir progresivamente desencolando y procesando los mensajes o dejar que el intercambiador se los haga llegar, esto por medio de diferentes tipos de rutas.

Por lo tanto podemos ver el concepto de las colas de mensajes como un intermediario entre los emisores y los destinatarios, o si lo orientamos a un escenario más real, entre clientes y servidores o publicadores y consumidores.

Ventajas

Desde luego estarás esperando que te demos argumentos que te hagan considerar implementar este enfoque de intercambio, veamos algunas:

* **Redundancia** – Si tu aplicación falla mientras está procesando alguna petición no te debes preocupar de que esta se pierda para siempre ya que esta estrategia le permite a la cola persistir el mensaje hasta que el mismo sea procesado por completo.
* **Naturaleza asíncrona** – Dependiendo de la manera en que funcione tu sistema puedes necesitar que los mensajes se vayan acumulando para procesarlos en lotes, la cola irá manteniendo tus mensajes para cuando decidas (o programes) su procesamiento.
* **Garantía de entrega y ordenamiento** – Se garantiza que el orden en el que llegan los mensajes será el orden en el que sean procesados, de igual manera un emisor y/o consumidor puede estar seguro de que este ha sido recibido y se procesará una única vez mediante la implementación de intercambio de banderas de reconocimiento.
* **Disponibilidad** – Si parte de tu arquitectura falla, los mensajes no se perderán ya que estos seguirán en la cola hasta que se les indique lo contrario, al mismo tiempo la cola podrá seguir recibiendo mensajes para el procesamiento posterior a la recuperación del sistema.
* **Elasticidad** – En situaciones donde tu sistema pudiera llegar al tope de su capacidad de recepción de peticiones y volverse incapaz de responder por un anormal flujo de mensajes, el hecho de tener una cola o *buffer* de peticiones permitirá balancear, filtrar y normalizar el flujo, lo cual evitará que tu sistema sea inundado de peticiones que no pueda responder causando perdida de los mismos o incluso el colapso del sistema.
* **Desacoplamiento** – El hecho de tener una capa intermedia de comunicación entre procesos nos da la flexibilidad en la definición de arquitectura de cada uno de ellos de manera separada, mientras cada uno se mantenga alineado a los mismos requerimientos de interfaz que representa la cola de mensajes no abran mayores problemas de compatibilidad ni mucho que cambiar de lado y lado.
* **Escalabilidad** – Por la misma ventaja de desacoplamiento podemos escalar fácilmente nuestro sistema, solo debemos agregar más unidades de procesamiento y el sistema de colas se encargará de balancear la carga entre ellos.

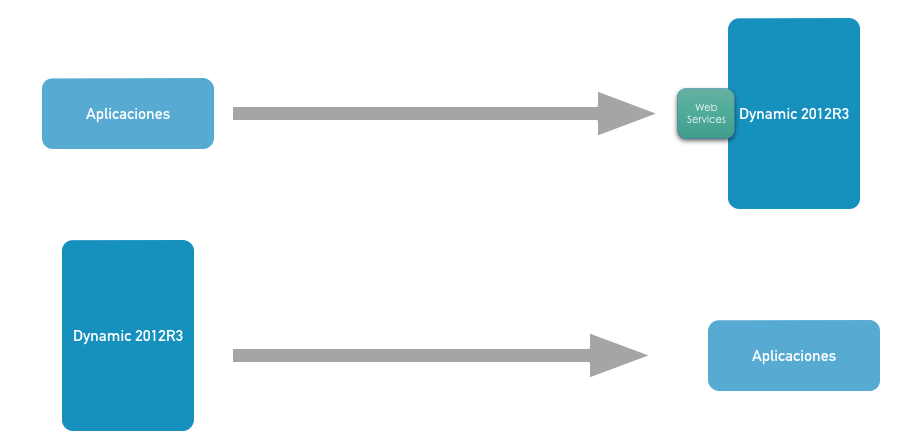
RabbitMQ

Luego de conocer un poco sobre lo que son las colas de mensajes podemos introducir la solución que utilizaremos aquí, RabbitMQ. Este programa nos ofrece las ventajas antes mencionadas de las colas de mensajes además de ofrece también otras características que lo hacen más flexible aún:

* Soporta múltiples protocolos, muchas soluciones solo manejan uno solo como AMQP.
* Dispone de librerías en casi todos los lenguajes de programación.
* Rastreo de mal comportamiento en las colas.
* *Clusterización* de varios servidores de colas para evitar que el sistema de encolamiento falle.
* Múltiples tipos de enrutamiento, incluso puedes crear tu propia ruta.
* Soporta *plugins* para extender su comportamiento.
* Amigable interfaz gráfica.

Existen algunas alternativas a RabbitMQ muy interesantes en el ámbito *cloud* o *SaaS* (software como servicio) como IronMQ y Amazon SQS; siempre promovemos el uso de herramientas de código abierto y por ello optamos por RabbitMQ, la solución que consideramos más robusta de manejo de colas de mensajes.

Utilizar RabbitMQ como intermediario de mensajes en las interfaces de datos de Dynamics AX

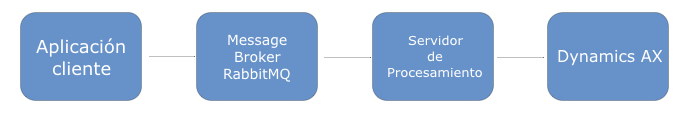
Una de las cosas que me encanta de Dynamics AX es lo fácil que es crear interfaces de datos para permitir la integración con otros sistemas. Si ha trabajado con Dynamics AX durante un período de tiempo prolongado, probablemente haya visto varias integraciones de servicios web que permiten la interoperabilidad con otros sistemas de línea de negocio y heredados. Un par típico de integraciones entrantes y salientes podría verse como la imagen siguiente.   


El uso de una conexión estrechamente acoplada entre los sistemas de origen y destino suele ser la forma más fácil (por lo tanto, la más rápida y económica) de establecer una integración, pero a menudo es una mala idea. Tenga en cuenta el escenario de entrada en el que una aplicación externa está enviando datos a Dynamics AX. ¿Qué sucede si la aplicación de llamada se comporta mal y comienza a enviar miles de solicitudes por segundo? Esto tiene el potencial de sobrecargar su servidor AX y hacerlo completamente inutilizable. Ahora considere el escenario de salida en el que un complemento de AX llama a un servicio web externo. Si el servicio web de la aplicación de destino está fuera de línea durante unos minutos, la actualización del complemento de AX no se recibirá a menos que exista algún tipo de error al manejar y reintentar la lógica incorporada en la aplicación de AX.

Un enfoque alternativo

Por estas razones, y por muchas otras (registro, seguridad, escalabilidad, solo por nombrar algunas), se considera una mejor práctica crear integraciones poco integradas que dependan de un intermediario de mensajes que se ubica entre los sistemas de origen y de destino. Aunque la definición formal es más complicada, para nuestros propósitos un intermediario de mensajes puede considerarse como una colección de colas que contienen mensajes. Los editores escriben mensajes en las colas y luego los consumidores recogen los mensajes y los procesan de manera adecuada. Además, el intermediario de mensajes se puede configurar para mantener los mensajes en sus colas hasta que los consumidores brinden confirmación del procesamiento exitoso.

Aquí hay un ejemplo de cómo se verían las integraciones con un intermediario de mensajes.



Para la llamada saliente desde la aplicación cliente, la aplicación cliente escribe el mensaje en un intermediario. El mensaje se enruta a una cola donde espera ser procesado. Una aplicación de servicio de procesamiento por separado recupera el mensaje de la cola y lo envía a AX mediante la invocación de un Web Services SOAP. Para la llamada saliente desde AX, el proceso funciona exactamente igual, excepto que las aplicaciones de origen y de destino se invierten.

Conclusión

¿Por qué es mejor un intermediario de mensajes?

En el escenario de llamadas entrantes, típicamente se esperaría que un intermediario de mensajes eficaz manejara un volumen mayor de mensajes entrantes que Dynamics AX porque todo lo que hace es recibir y enrutar los datos sin ningún procesamiento adicional. El servicio de procesamiento puede procesar los mensajes en la cola a una velocidad que no sobrecargue el servidor de Dynamics AX. En el caso de la llamada saliente, la combinación de un intermediario de mensajes y el servicio de procesamiento puede habilitar la lógica de reintento compleja y el registro personalizado sin tener que almacenarlo en la capa de complemento. Como una ventaja adicional para cualquier escenario, un intermediario de mensajes puede proporcionar una garantía de que los mensajes no se pierden entre los sistemas de origen y destino, siempre y cuando el mensaje se publique exitosamente en el intermediario.

Se seleccionó RabbitMQ por varias razones:

1. Es de código abierto.
2. Se ejecuta en múltiples plataformas.
3. Es fácil de instalar y configurar.
4. Es rápido en el procesamiento de mensajes.