Facultad de Ingeniería Universidad de Buenos Aires

75.61 Taller de Programación III

TP N°2 - Ejercicio de Colas (RabbitMQ)
(Correciones)

Profesor: Andrés Veiga
JTP: Pablo Roca

Integrantes:

Padrón	Nombre	Email
89579	Torres Feyuk, Nicolás R. Ezequiel	ezequiel.torresfeyuk@gmail.com

Índice

1. Introducción

El presente trabajo práctico consiste en diseñar e implementar un sistema de recepción de pedidos masivos. Para realizar esta tarea, se deben implementar procesos que modelen diferentes partes/tareas del sistema. Para comunicar los procesos, se deben utilizar cola de mensajes distribuidas.

RabbitMQ es un middleware orientado a mensajes el cual permite conectar diferentes procesos a través de colas de mensajes distribuídas. El objetivo del presente trabajo consiste en:

- Focalizar el desarrollo de sistema en la recepción fluída de **Orders** y **Queries** realizadas por los clientes
- Comprender el funcionamiento básico de RabbitMQ
- Paralelizar la mayor cantidad de tareas posible, exprimiendo en el máximo nivel posible el middleware de colas a utilizar
- Crear y diseñar una arquitectura escalable horizontalmente de forma de responder de forma eficaz y eficiente ante nuevos requerimientos

2. Arquitectura 4 + 1

2.1. Casos de Uso

En la figura 1 se exhibe el diagrama de casos de uso del sistema. Se detalla a continuación cada uno de las entradas del mismo:

- **Enviar Orden:** El cliente envía una orden de compra, la cual contiene un identificador único de la orden más la cantidad que desea obtener un determinado producto. Por el momento el sistema no acepta más de un producto por orden.
- **Consultar Orden:** Un cliente consulta en que estado se encuentra su orden. El sistema puede responderle con alguno de los siguientes estados:
  - *RECEIVED*
  - *ACCEPTED*
  - *REJECTED*
  - *DELIVERED*
- **Procesar Orden:** Como se considera a los Empleados externos al sistema, el procesamiento de una orden es válido como caso de uso. Los empleados obtienen los pedidos a procesar (aquellos que fueron aceptados), y al terminar de trabajar con los mismo proceden a cambiar su estado a *DELIVERED*
- **Incrementar Stock:** Un proveedor se encarga de aumentar el stock de los productos ofrecidos por la aplicación

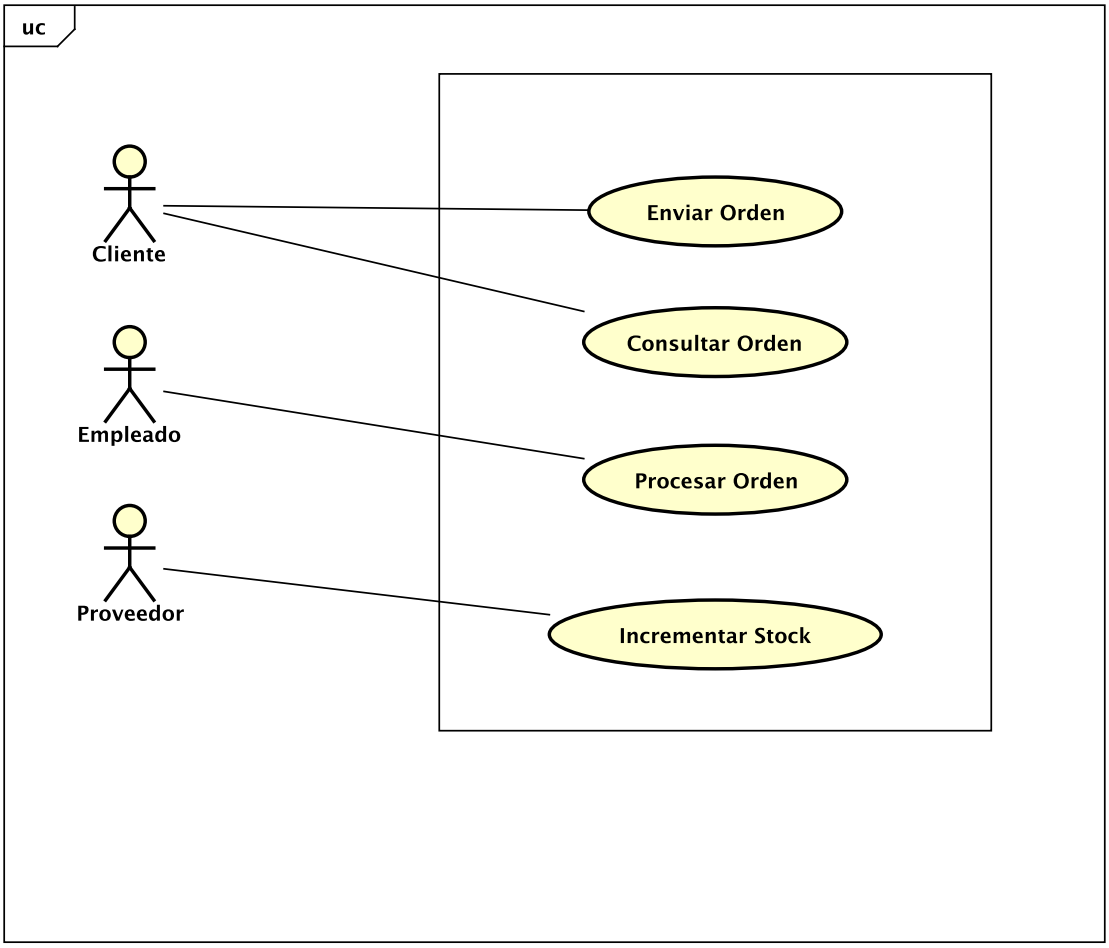

Figura 1: Diagrama de Casos de Uso

2.2. Vista Lógica

En la figura 2 se exhibe el diagrama de clases, el cual muestra la vista lógica del sistema. A continuación se adjunta una breve explicación de cada una de las clases:

- **DefaultConsumer:** Clase abstracta proveída por RabbitMQ para consumir los mensajes recibidos en las colas. Cada proceso que recibe mensajes desde una cola crea un objeto que hereda de esta clase. Las clases que poseen como Base a esta clase son las siguientes:
  - *RequestDispatcher*
  - *StockManager*
  - *QuerySolver*
  - *AuditLogger*
  - *OrderManager*
- **StockDB:** Esta clase encapsula la persistencia del Stock de los Productos del sistema. La misma almacena cada producto en un archivo binario como registros de tamaño fijo.
- **Order:** Entidad que modela a los Pedidos efectuados por los clientes del sistema. La misma está compuesta por los siguientes atributos:
  - *Key:* UUID que representa unequívocamente a cada *Order*.
  - *Product:* Tipo de producto que el cliente desea obtener.
  - *Amount:* Cantidad de productos del mismo tipo que se desea obtener.
  - OrderState: Estado de la orden. El mismo es modificado por cada proceso del sistema durante todo el ciclo de vida del objeto en el sistema.
- **OrderDB:** Clase que expone una API(add, alter) para efectuar ABMs sobre las *Orders* que lleguen al sistema. La misma almacena las órdenes en diferentes archivos, agrupando las mismas en función de los 8 bits más significativos del ID de los *Pedidos*. Al igual que *StockDB*, esta clase almacena cada *Pedido* como un registro de tamaño fijo.
- **OrderDBEntry:** Entidad que posee la responsabilidad de serializar/deserializar *Orders*.
- **OrderState:** Enum con los diferentes estados que posee un producto
- **Product:** Enum que identifica a cada producto en función del nombre que posee el mismo. Además de esto, posee en su firma un factory method que permite crear un producto al azar entre todos los posibles.

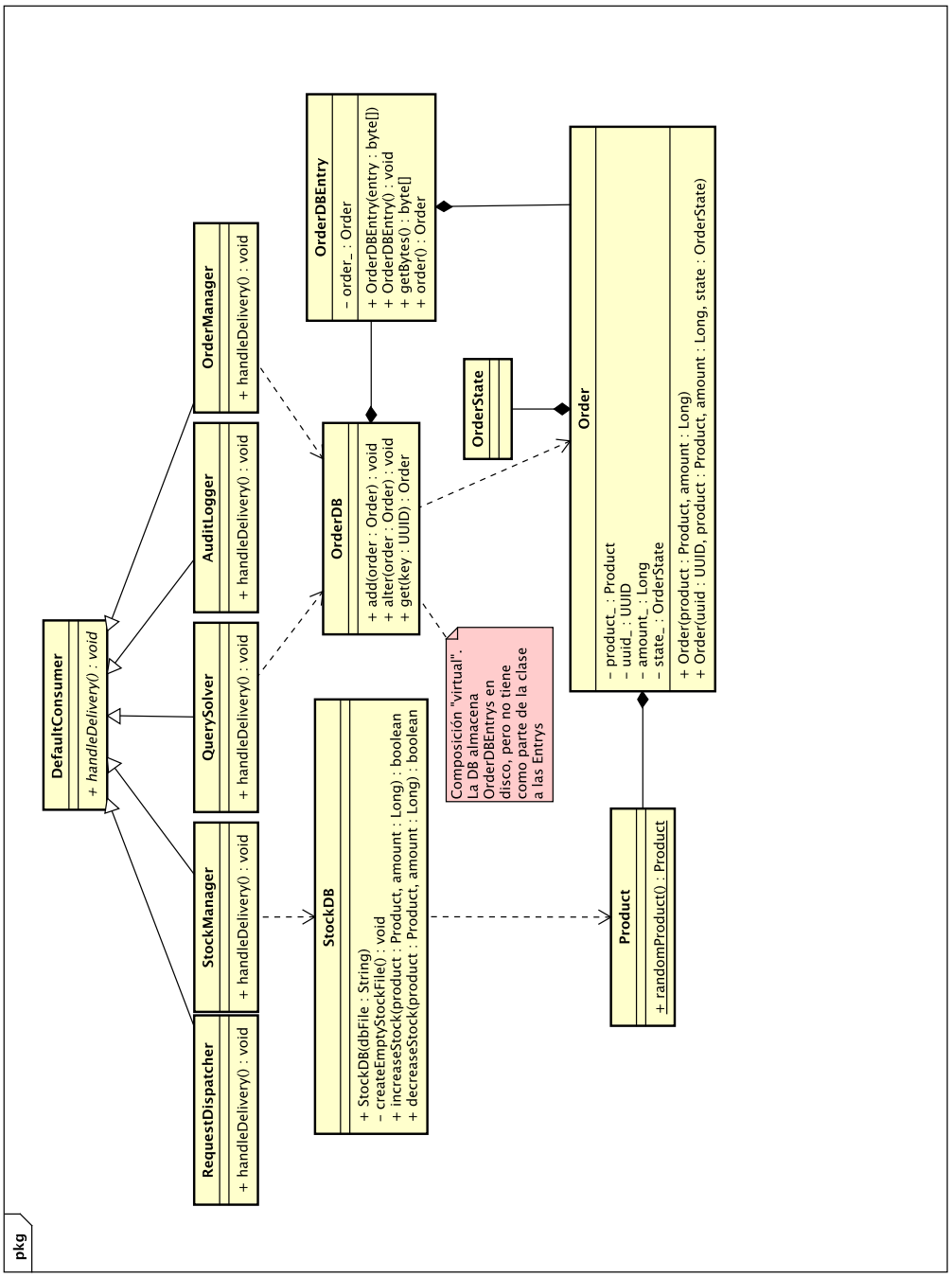

Figura 2: Diagrama de Clases

2.3. Vista de Despliegue

Para la vista de despliegue se decidió realizar un diagrama de robustez y un diagrama de despliegue. El primero explica como los diferentes procesos del sistema se comunican entre sí a través de diferentes colas de mensaje, mientras que el segundo intenta representar la arquitectura física de la aplicación.

2.3.1. Diagrama de Robustez

El mismo se puede visualizar en la figura 3. El presente diagrama posee símbolos extra para explicar de mejor manera la interacción entre los procesos del sistema y las colas de Rabbit. Se decidió además utilizar cilindros en vez de *Entity Objects* para visualizar aquellas entidades que persisten información.

Las flechas entrantes a las colas de Rabbit simbolizan que la *Entidad* o *Controlador* están colocando un mensaje en la cola. Las flechas salientes a las colas de Rabbit simbolizan que la *Entidad* o *Controlador* están sacando mensajes de la cola. En el caso de las DBs no se cumple esta regla. Las flechas solo implican que el *Controlador* en cuestión se encuentra realizando operaciones en la misma.

En el gráfico se puede observar que las entradas del presente sistema están representadas por los actores presentes en el Diagrama de Casos de Uso exhibido en la figura 1.

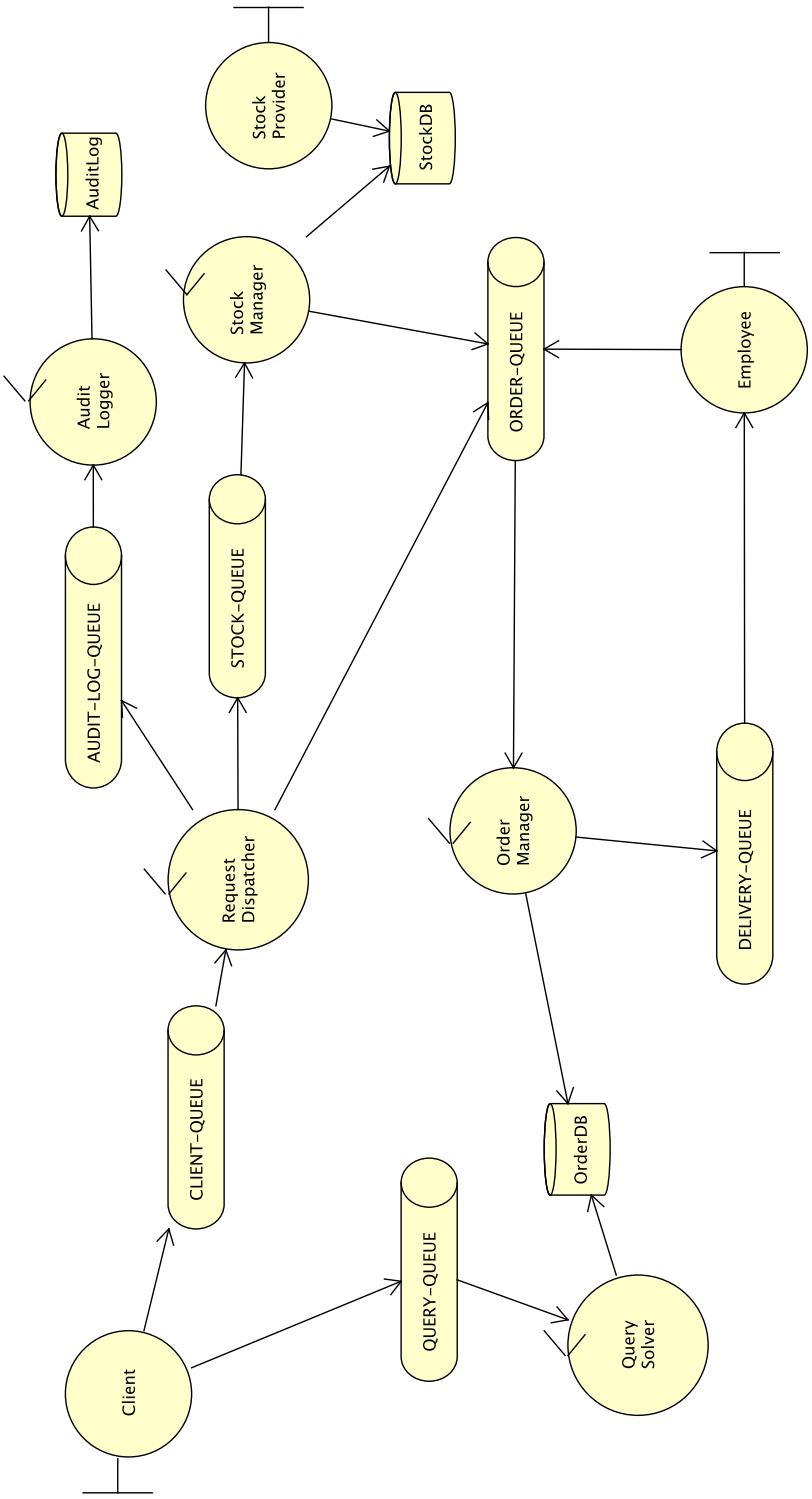

Figura 3: Diagrama de Robustez

2.3.2. Diagrama de Despliegue

El diagrama se despliegue se exhibe en la figura 4. Cada nodo representa a un servidor físico independiente, el cual corre aplicaciones y puede o no tener Base de Datos. Se procede a detallar cada uno de los nodos:

- **DispathServer:** En este nodo corren los procesos RequestDispatcher que se encargar de recibir las **Orders** de los clientes para derivarlas a otros nodos. En el diagrama se puede ver que existen dos nodos de este tipo. Con esto se quiere representar que se puede escalar este nodo agregando N de los mismos de forma transparante al sistema.
- **AuditLogServer:** Nodo donde se encuentra el log de auditoría. Como los logs deben ser almacenados en orden, solo puede haber un proceso corriendo de este tipo. El proceso debe correr en un servidor aparte debido a que este es un punto sensible del sistema y debe ser optimizado en la mayor medida posible.
- **StockServer:** En el presente nodo se encuentra la base de datos en donde se almacena el Stock de los productos. En el nodo se pueden ver más de una aplicación, lo cual indica que se pueden correr N instancias de este proceso en el dispositivo físico, pero que no se puede agregar más nodos de este tipo. Esto último se debe a que los procesos a través de I/O syscalls a la DB las cuales deben correrse en el mismo servidor físico donde se encuentra la DB en cuestión.
- **OrderServer:** En el presente nodo se encuentran los procesos OrderManager y QuerySolver. Se cumplen los mismos requisitos que para el proceso StockManager, dado que estos procesos también realizando un acceso físico a un DB (OrderDB).

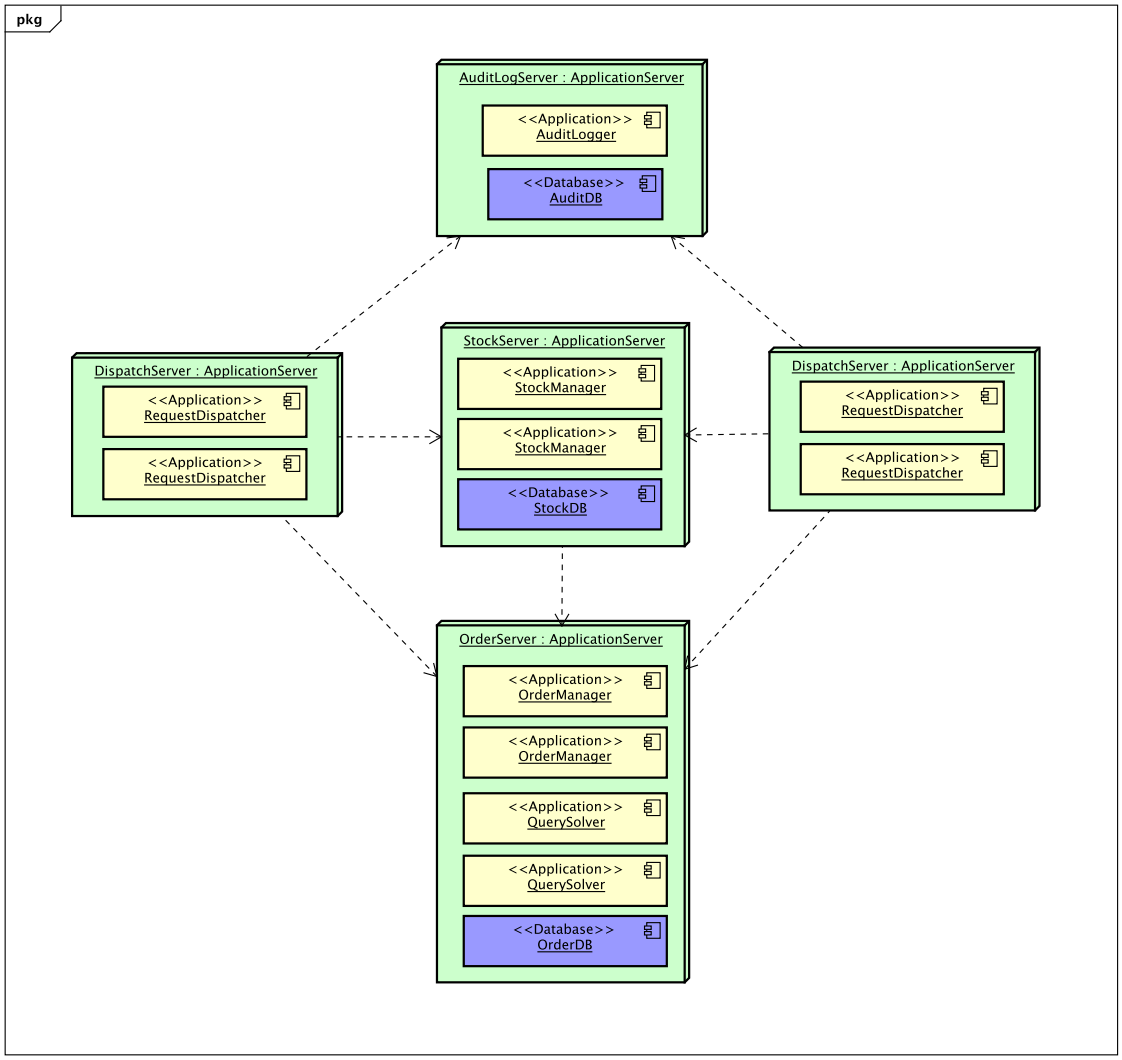

Figura 4: Diagrama de Despliegue