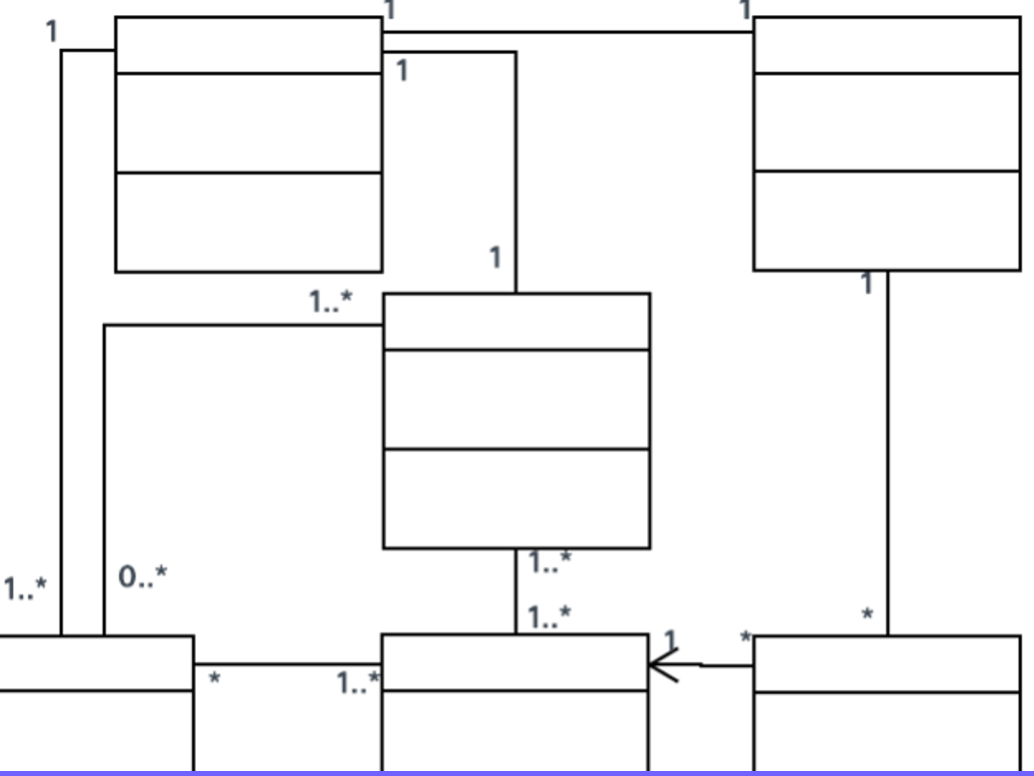
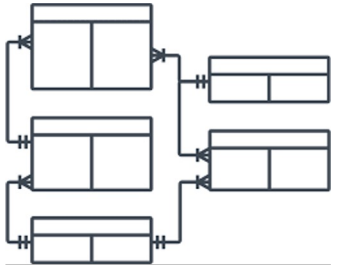


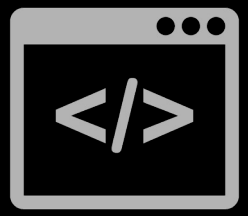


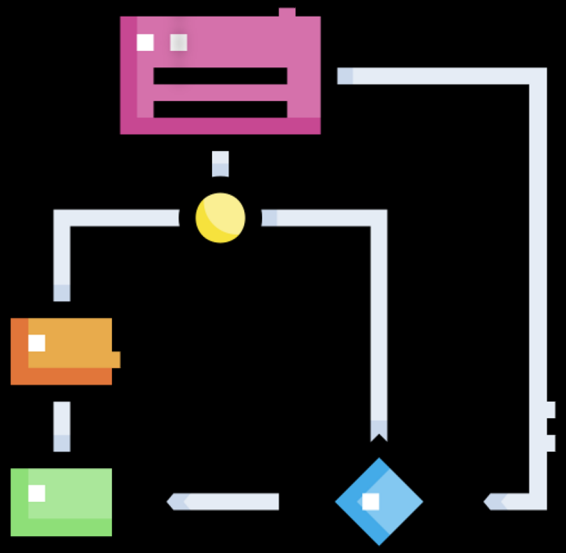
Diseño de Sistemas

Agenda

- Interfaces entrantes & salientes
- Cuestiones de Sincronismo. Cron Task
- Patrones creacionales (Patrón Singleton, Patrón Factory Method, Patrón Simple Factory)

Integración de Sistemas: Interfaces entre componentes



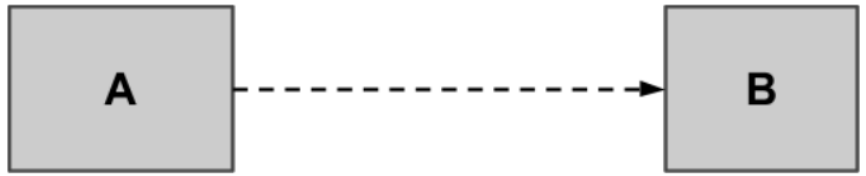


UNIFIED MODELING LANGUAGE™

Interfaces entre Componentes

Sabiendo que un Sistema es un conjunto de Componentes que se relacionan para cumplir un objetivo en común:

¿Cómo se relacionan dichos componentes?



Interfaces entre Componentes

Consideraciones al momento de Diseñar:

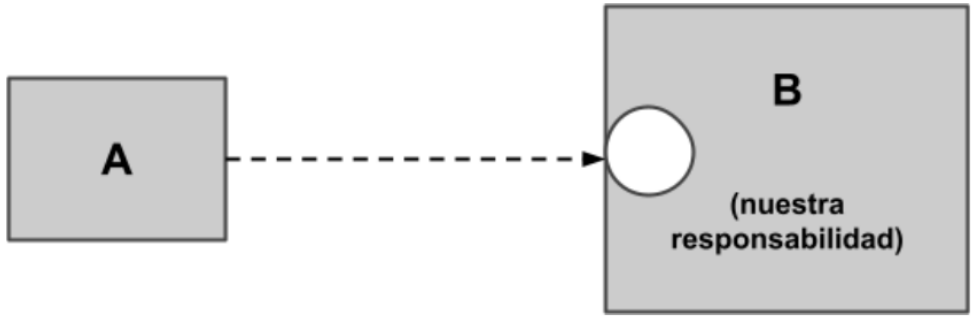
- *Dirección y sentido de la comunicación*
  - *A* llama a *B*
  - *B* llama a *A*
  - La comunicación es ***bidireccional***

- *Ambiente en el que residen los componentes*
  - Los componentes están en el mismo ambiente
  - Los componentes están en ambientes separados

- Sincronismo/Asincronismo en peticiones

Interface Entrante

La ***interface entrante*** es aquel contrato que define qué ***datos*** necesita ***nuestro componente*** (componente B en el gráfico) para poder realizar determinada tarea/ejecutar cierta funcionalidad.
Se debe tener en cuenta cómo utilizarán los terceros nuestro componente.



Interface Saliente

La ***interface saliente*** es aquel contrato que define qué ***datos*** necesita el ***componente*** al cual nos queremos integrar (componente B en el gráfico) para poder realizar determinada tarea/ejecutar cierta funcionalidad.
En este caso, se considera que nuestro componente A está "consumiendo" al componente B.

Entorno de Ejecución

¿Qué es un entorno de ejecución?

"Es un contexto que refiere a hardware y software donde se ejecuta una aplicación".

Es importante destacar que no necesariamente todos los componentes de un aplicativo/Sistema ejecutan en el mismo entorno.

Entorno de Ejecución

Al integrar componentes debemos tener en cuenta dónde están ubicados físicamente los mismos:

- En el mismo Sistema – mismo entorno
- En el mismo Sistema – diferente entorno
- En Sistemas diferentes – diferentes entornos

Entorno de Ejecución – Mismo Entorno

En el caso que ambos componentes se encuentren en el mismo entorno (mismo Sistema), la integración entre ellos podrá resultar sencilla, aunque:

- Se debe pensar en qué mecanismo de integración es el adecuado para la comunicación entre ambos componentes.
- Se debe prestar atención al grado de acoplamiento que se podría generar entre ambos componentes, reduciendo el mismo al máximo posible.
- Se debe evitar la ruptura del encapsulamiento del componente integrado.
- Se debe diseñar una integración Testeable.

Entorno de Ejecución – Entornos diferentes

En el caso que los componentes se encuentren en entornos diferentes, la integración entre ellos resultará –quizás- más compleja que el caso anterior. En este caso:

- Debemos pensar cuál será el mecanismo para que ambos componentes se comuniquen (API REST, por ejemplo.), en el caso que no estén definidas las interfaces.

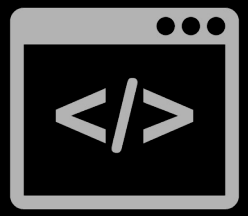
- Si la interfaz del componente ya está definida, debemos integrarnos de la forma en que lo exige.

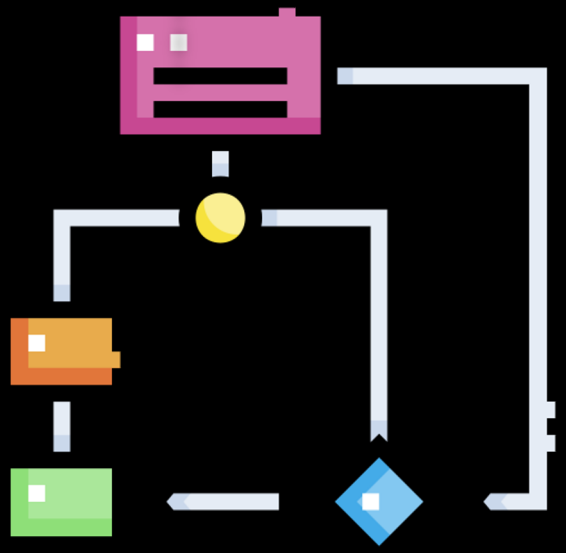
Testing con Integraciones

Si estamos testeando un componente propio que tiene una integración con otro que no está ejecutando en el mismo entorno (sea propio o no), se debe considerar:

- La interface saliente debe estar mockeada, porque de no hacerlo estaríamos llamando a dicho componente en cada test con todo lo que ello implica *(¿qué sucedería si el componente nos cobra por cada llamada?)*

Cuestiones de Sincronismo





Cuestiones de Sincronismo

¿Qué debemos hacer cuando llamamos a un componente?

¿Nos quedamos esperando la respuesta en ese momento?

¿Seguimos trabajando en otra cosa hasta que se resuelva nuestro llamado?

Integraciones Sincrónicas

*Cuando un componente **A** hace uso de alguna funcionalidad expuesta por el componente **B** y **se queda esperando la respuesta**, entonces A está integrándose de forma **sincrónica** con B para la utilización de dicha funcionalidad.*

- Es la forma más simple de trabajar porque es fácil de razonar.
- Es la forma en la que trabaja el envío de mensajes entre objetos por defecto.

Integraciones Sincrónicas

Pero … ¿qué sucedería si consideramos que, en el siguiente caso, el Cotizador necesita realizar una Request a un componente externo y sabemos que la latencia puede ser amplia?

```
dolarHoy = cotizador.obtenerCotizacion()
```

Estaríamos ***penalizando a todo el resto del Sistema por tan solo esperar la respuesta*** del cotizador para poder continuar con la labor.

Integraciones asincrónicas

*Cuando un componente **A** hace uso de alguna funcionalidad expuesta por el componente **B** y **no se queda esperando la respuesta**, entonces A está integrándose de forma **asincrónica** con B para la utilización de dicha funcionalidad.*

- No es la forma en la que estamos acostumbrados a pensar el común de las cosas.
- Es una buena manera de no penalizar toda la ejecución del Sistema.

Integraciones asincrónicas

El componente A puede no quedarse esperando la respuesta de B por alguno de los siguientes motivos:

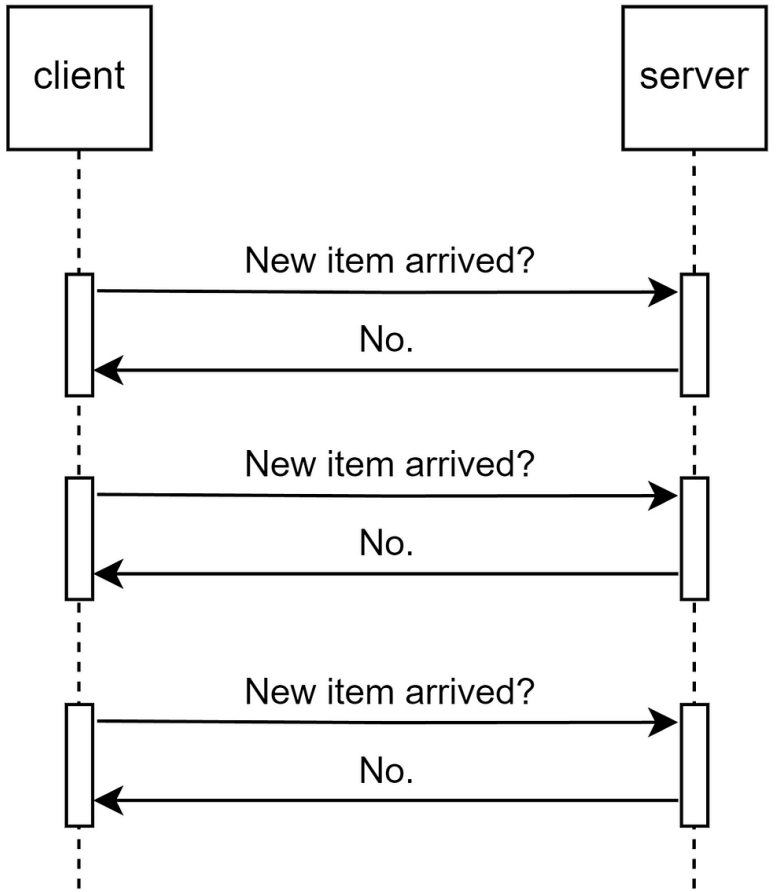
- No necesita el resultado/la respuesta en ese momento para continuar con la ejecución.
- No le interesa el resultado/la respuesta.
- No le estaba pidiendo nada, sino que le estaba "avisando de algo".

Integraciones asincrónicas - Respuestas

Algunas de las formas que tiene A de enterarse, en un momento posterior, el resultado que generó el componente B, ante la ejecución de dicha funcionalidad, son:

- Preguntarle a B "*cuál es el estado de la ejecución*" o "*si ya terminó*"
    - Ante una mala utilización de este mecanismo podríamos generar una ***espera activa*** en A.
    - A esta forma se la conoce como "***pull based***"

Integraciones asincrónicas - Respuestas



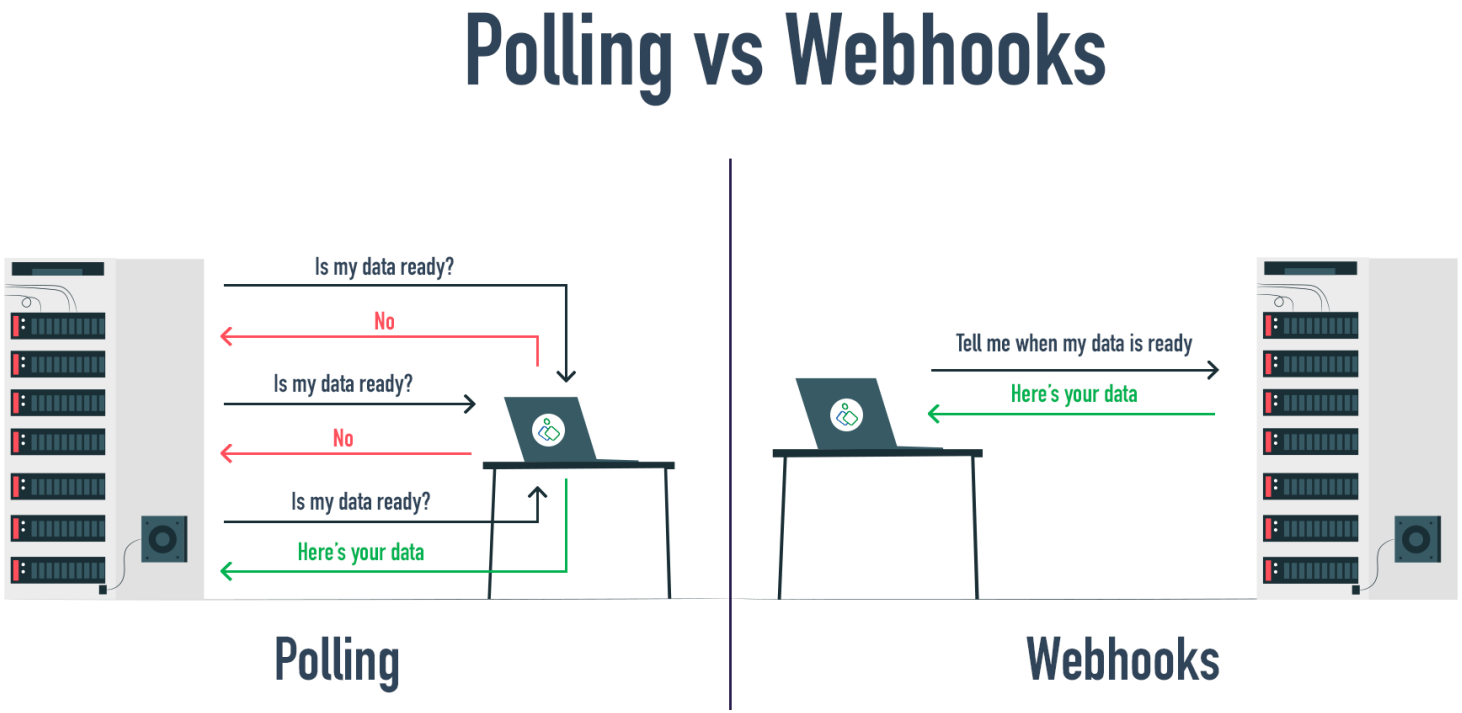
Integraciones asincrónicas - Respuestas

- Leer periódicamente algún espacio de memoria compartido con B, en el cual B depositará el resultado.
    - Nuevamente, ante una mala utilización de este mecanismo podríamos generar una espera activa en A.

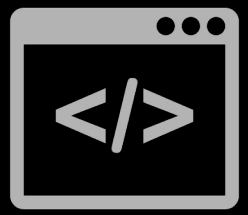
Integraciones asincrónicas - Respuestas

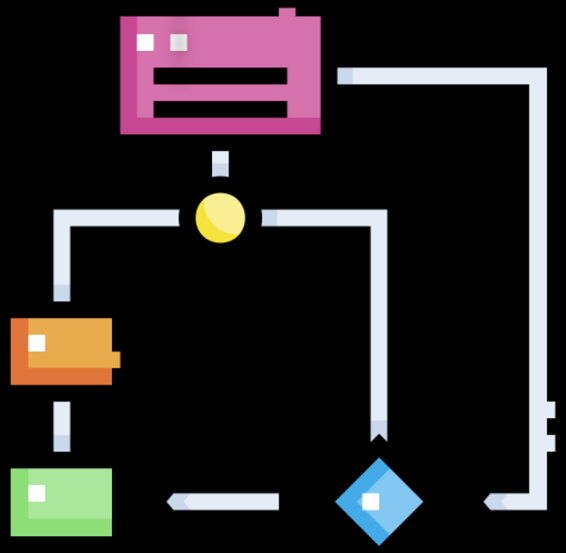
- Pasarse como parámetro para que B le avise cuando haya terminado (si A y B están en el mismo ambiente).
- Enviarle a B un ***callback*** para que ejecute luego de haber terminado la ejecución
    - A este mecanismo se lo conoce bajo el nombre "***Webhook***", en el caso de que ambos componentes estén bajo una Arquitectura Web.
    - A esta forma se la conoce como "***push based***"

Integraciones asincrónicas - Respuestas



Ejecución asincrónica: CronJobs





CronJobs

Analicemos los siguientes requerimientos de ejemplo:

- *"Todas las noches el Sistema debe enviar un email a los clientes para (…)"*
- *"Cada semana el Sistema debe analizar los gastos del cliente y generar un reporte (…)"*
- *"Todos los jueves el Sistema debe enviar un voucher de descuento en hamburguesas a los empleados(…)"*

¿Qué tienen en común todos ellos?

CronJobs

Requerimiento	Acción a realizar	Frecuencia	Evento disparador
"Todas las noches el Sistema debe enviar un email a los clientes para (…)"	Enviar email	Todas las noches	El paso del tiempo
"Cada semana el Sistema debe analizar los gastos del cliente y generar un reporte (…)"	Generar reporte	1 vez por semana	El paso del tiempo
"Todos los jueves el Sistema debe enviar un voucher de descuento en hamburguesas a los empleados(…)"	Enviar voucher	1 vez por semana	El paso del tiempo

CronJobs

¿Cómo podemos lanzar las ejecuciones requeridas en el momento exacto? ¡CronJobs!

*Un **CronJob es una tarea programada** que se ejecuta de manera **periódica** y **automática** en Sistemas Unix-like (como Linux o MacOS). Se utiliza para automatizar comandos o scripts en intervalos de tiempo definidos por el usuario.*

Si bien la definición está pegada a los Sistemas basados en Unix, también se pueden crear tareas programadas en Windows (desde el programador de tareas).

CronJobs

Elementos de un CronJob

Un CronJob se define en una línea dentro de un archivo llamado "crontab" y tiene la siguiente estructura:

```
* * * * * comando
| | | | |
| | | | +----- Día de la semana (0 - 7) (Domingo puede ser 0 o 7)
| | | +------- Mes (1 - 12)
| | +--------- Día del mes (1 - 31)
| +----------- Hora (0 - 23)
+------------- Minuto (0 - 59)
```

CronJobs

Como comando podríamos pasarle, por ejemplo, la siguiente línea:

```
”java –jar enviarEmails.jar”
```

En este caso estaríamos mandando a ejecutar un archivo compilado y empaquetado “jar” de Java, el cual debe contener un main específico para que pueda ejecutarse.

Gracias