Primer Proyecto: Restorrente - El restaurant concurrente

Joaquin Segui, Juan Manuel Romera Ferrio

Resumen. Se presenta en el siguiente informe la descripción de procesos, análisis y funcionamiento del sistema construido que simula el funcionamiento de un Restaurante.

1 Introducción

En la materia Técnica de Programación Concurrentes I se nos planteó como primer trabajo practico la construcción de un sistema que simule parcialmente el funcionamiento de un restaurante. El mimo debe ser construido con la noción de concurrencia aprendida en la materia y usando exclusivamente los elementos que se vieron en la primera parte de la misma.

2 Requerimientos Funcionales

Los requerimientos funcionales son los siguientes:

1. Al restaurant ingresan grupos de comensales que son recibidos en la puerta por los recepcionistas.
2. Cada recepcionista atiende a un grupo de personas a la vez y los ubica en una mesa disponible. Si no hay más mesas disponibles, los hace pasar al living del restaurant, donde esperarán a que se desocupe una mesa.
3. Si todos los recepcionistas están ocupados, las personas esperan en la puerta del restaurant.
4. Cuando las personas están en la mesa, le ordenan al mozo un pedido de comida. El mozo toma nota de lo solicitado y le solicita la comida al cocinero. Hay un sólo cocinero que atiende a los mozos de a uno por vez. El cocinero le entrega el pedido al mozo y éste a los clientes de la mesa, quienes luego comerán.
5. Los clientes pueden repetir el ciclo de ordenar comida y comer, cuantas veces lo deseen.
6. Una vez que terminaron de comer, los clientes le solicitan al mozo “la cuenta". Le pagan el importe al mozo, quien lo deposita en la caja del restaurant. Una vez que pagaron, los clientes se retiran del restaurant.
7. De forma inesperada y sorpresiva, se corta el suministro de energía eléctrica en el restaurant, quedando totalmente a oscuras. Los clientes, furiosos, se van del restaurant sin pagar lo que hayan consumido hasta el momento. También se van los

que estaban esperando en el living. Cuando se resuelve el problema eléctrico, el restaurant vuelve a funcionar normalmente, recibiendo nuevos clientes.

8. La simulación finaliza cuando todas las personas que iban a comer ese día en el restaurant se retiraron.
9. Periódicamente, el Gerente del restaurant consulta:
    a. La cantidad de dinero en la caja.
    b. La cantidad de dinero que el restaurant perdió de facturar debido a los cortes eléctricos.
    c. La cantidad de grupos de personas esperando en el living.
10. Los siguientes parámetros deben ser configurables sin necesidad de recompilar el código:
    a. La cantidad de recepcionistas.
    b. La cantidad de mesas.
    c. La cantidad de mozos.
    d. El menú del restaurant: el listado de platos y el precio de cada uno.
11. Durante la ejecución de la simulación se deberá poder lanzar manualmente el corte de energía eléctrica.

3 Requerimientos No Funcionales

Los requerimientos no funcionales son los siguientes:

1. El proyecto deberá ser desarrollado en lenguaje C o C++, siendo este último el lenguaje de preferencia.
2. La simulación puede no tener interfaz gráfica y ejecutarse en una o varias consolas de línea de comandos.
3. El proyecto deberá funcionar en ambiente Unix / Linux.
4. La aplicación deberá funcionar en una única computadora.
5. El programa deberá poder ejecutarse en “modo debug", lo cual dejara registro de la actividad que realiza en un único archivo de texto para su revisión posterior. Se deberá poder seguir el recorrido de cada uno de los grupos de personas. Se deberá también poder observar el momento en que ingresan y se retiran del restaurante.
6. Las facilidades de IPC que se podrán utilizar para la realización de este proyecto son las que abarcan la primera parte de la materia, es decir, hasta el primer parcial. Dichas facilidades son:
    a. Memoria compartida
    b. Señales
    c. Pipes y Fifos
    d. Locks
    e. Semaforos

4 Procesos

Para la resolución del problema se decidió dividir la simulación en procesos. Como entendíamos a cada actor del sistema como un actor particular con funciones específicas decidimos que cada actor sea un proceso del sistema. Como resultado de lo expresado los procesos que se encuentran en el sistema son los siguientes.

4.1 Restaurant

El proceso **padre** de la aplicación es el proceso **restaurant** este proceso se crea cuando se quiere iniciar la simulación. Las tareas de este proceso son iniciar la memoria compartida y los semáforos que se usaran en el sistema, y también crear a todos los elementos que son parte del restaurant, **Recepcionistas**, **Mozos**, **Cocinero**, **Encargado**. Este proceso también se encarga de la eliminación de los recursos.

4.2 Recepcionistas

El proceso **Host**, es el nombre de la clase en el proyecto, tiene las siguientes funciones:

- Buscar comensales en la puerta, estableciendo una comunicación con los procesos de **Comensales.**
- Asignarle una mesa al comensal en caso de que haya mesa libre, esto implica acceder a la memoria compartida y luego comunicarse con el proceso del **Comensal.**
- Llevar al comensal al living en caso de que no haya mesa disponible, esto implica la creación de un nuevo proceso hijo del recepcionista llamado **AddDinerToLivingLineAction.**
- Echar al comensal del restaurant en caso de que ya haya entrado toda la gente que se configuro para la simulación, esto implica acceder a la memoria compartida y luego comunicarse con el proceso del **Comensal.**

4.3 Mozos

El proceso **Waiter**, es el nombre de la clase en el proyecto, tiene las siguientes funciones:

- El mozo escucha alguna orden que le es dada, este mecanismo de escucha se hace a través de una **fifo** que le permite establecer una comunicación con los procesos de **Comensal y Cocinero.** El mozo puede recibir 3 tipos de ordenes:
    - Comensal le ordena su plato.
    - Cocinero le avisa que está listo un plato.
    - Comensal le paga la cuenta.

- En el caso de recibir el mensaje del **Comensal** ordenándole un plato el mozo crea un proceso hijo que será el encargado de enviar la orden al cocinero, este proceso se llama **SendOrderToCookAction.**
- En el caso de recibir el mensaje del **Cocinero** avisándole que un plato está listo, el mozo se comunica con el proceso del **Comensal** al cual le pertenece ese plato y se lo entrega.
- En el caso de recibir el mensaje del **Comensal** pagándole la cuenta el mozo actualiza la plata en la caja.

4.4 Cocinero

El proceso **Cook**, es el nombre de la clase en el proyecto, representa al cocinero del restaurante y tiene las siguientes funciones:

- El cocinero espera que le llega una orden para cocinar, esto lo hace leyendo de una **fifo.**
- Cuando le llega una orden, la cocina esta acción es simulada y se simula con un tiempo de espera.
- Cuando termina la orden envía la orden con la marca de cocinada a una **fifo** de la que leen los **Mozos.**

4.5 Encargado

El proceso **Attendant**, es el nombre de la clase en el proyecto, representa a un encargado y tiene las siguientes funciones:

- La función del encargado es asignarle una mesa a algún **Comensal** del living cuando un **Comensal** deja el restaurant. Para hacer esto se utiliza un semáforo que se activa cuando un **Comensal** deja el restaurant, permitiendo de este modo actualizar la memoria compartida y asignarle una mesa a un **Comensal.**

4.6 Comensales

El proceso **Diner**, es el nombre de la clase en el proyecto, representa a un grupo de Comensales en el restaurant. Este proceso no se inicia desde el proceso **padre restaurant** sino que se crea desde otro proceso. Las funciones son las siguientes:

- Entrar al **restaurant,** esto es escribir en una **fifo** que es leído por los **Recepcionistas.**
- Esperar respuesta**,** el comensal espera la respuesta. La respuesta puede ser que es echado porque ya entraron todas las personas al restaurante o la asignación de una mesa.
- En el caso de que pueda sentarse el comensal hará lo siguiente:

- Hace el pedido al mozo, esto establece una comunicación con el proceso del **Mozo** mediante una **fifo.**
- Espera su comida leyendo de una **fifo,** que tiene que ser escrita por un **Mozo.**
- Paga su cuenta, esto también establece una comunicación con el proceso del **Mozo** mediante una **fifo.**
- Se va del restaurant.

4.7 AddDinerToLivingLineAction

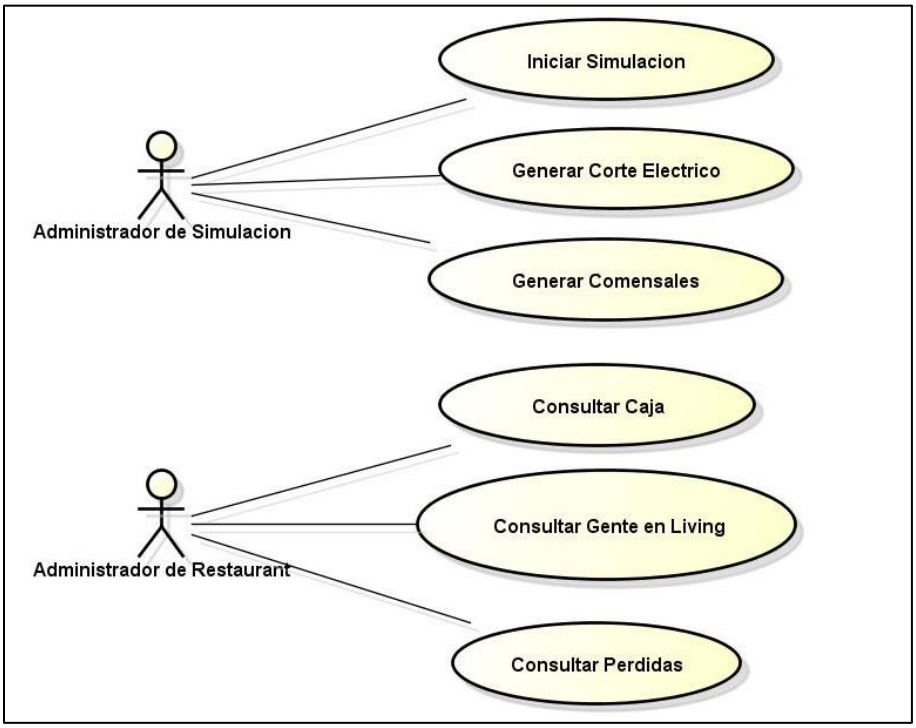
Este proceso es hijo del proceso **Recepcionista** y su función es únicamente actualizar la cantidad de gente en el living en la memoria compartida y escribir en una **fifo** que contiene los comensales que están esperando en el living.

4.7 SendOrderToCookAction

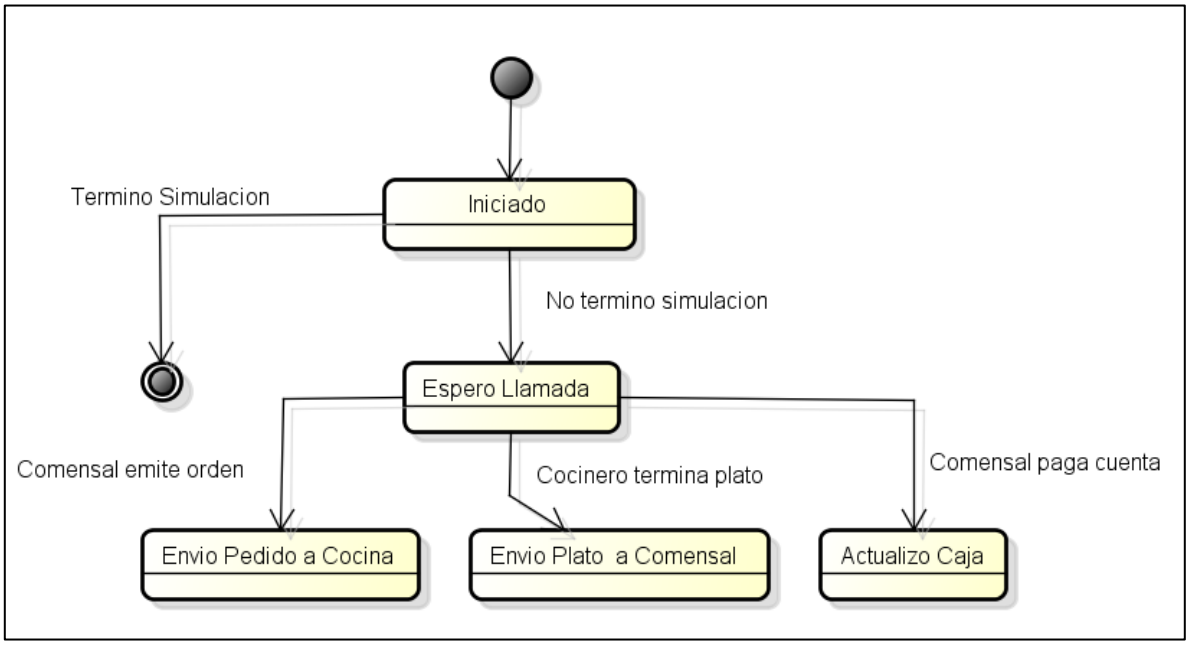
Este proceso es hijo del proceso **Mozo** y su función es únicamente escribir en una **fifo** que contiene los pedidos al cocinero.

5 Diagramas

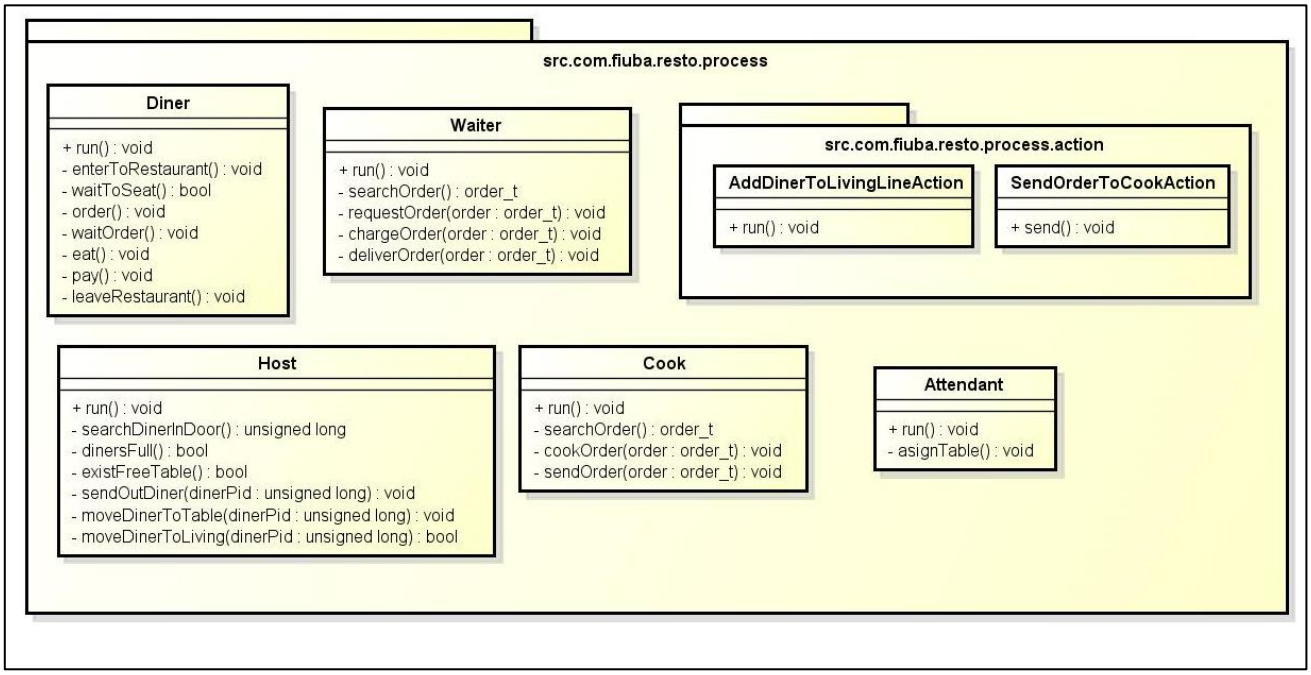
5.1 Casos de Uso



5.2 Estado de Mozo



5.3 Clases



5.4 Secuencia

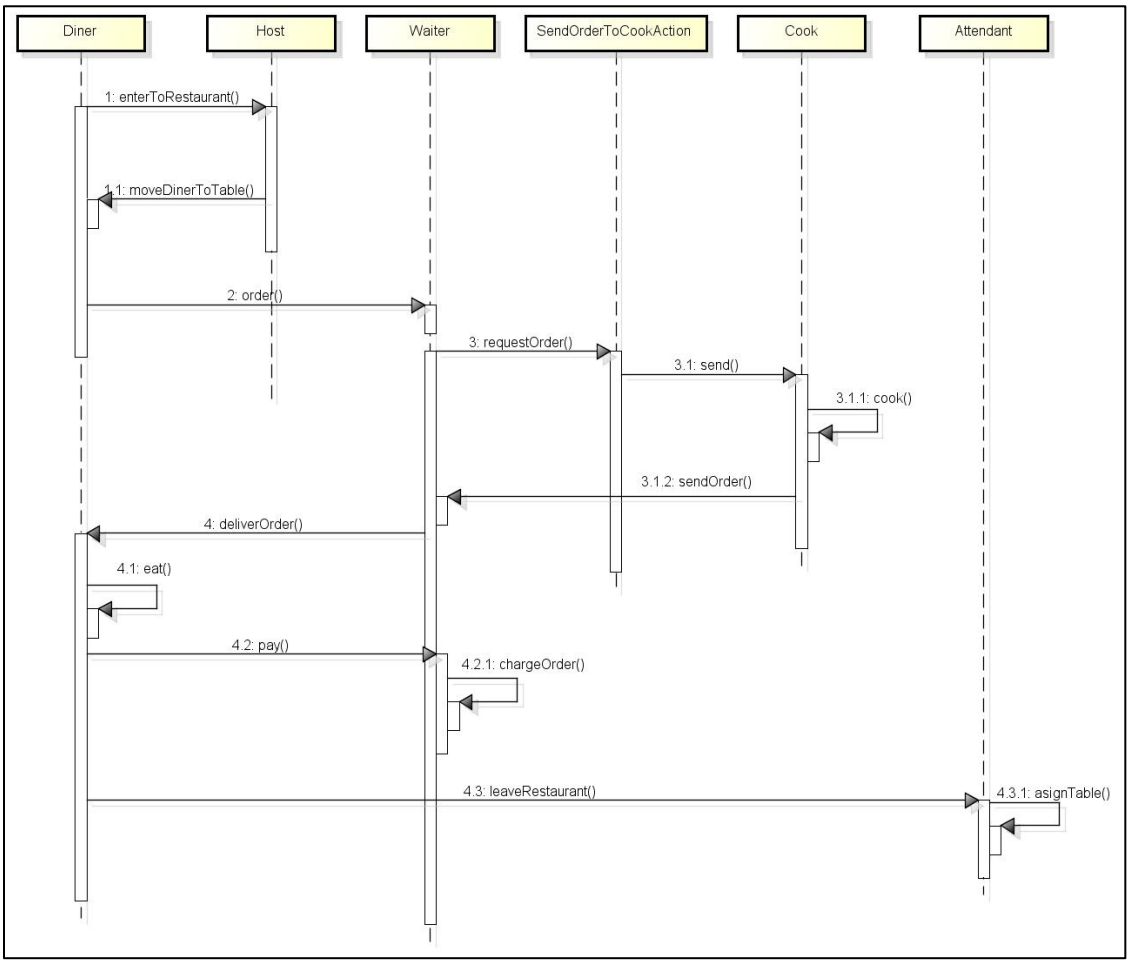


Diagrama de Secuencia para comensal que no tiene que esperar en el living.

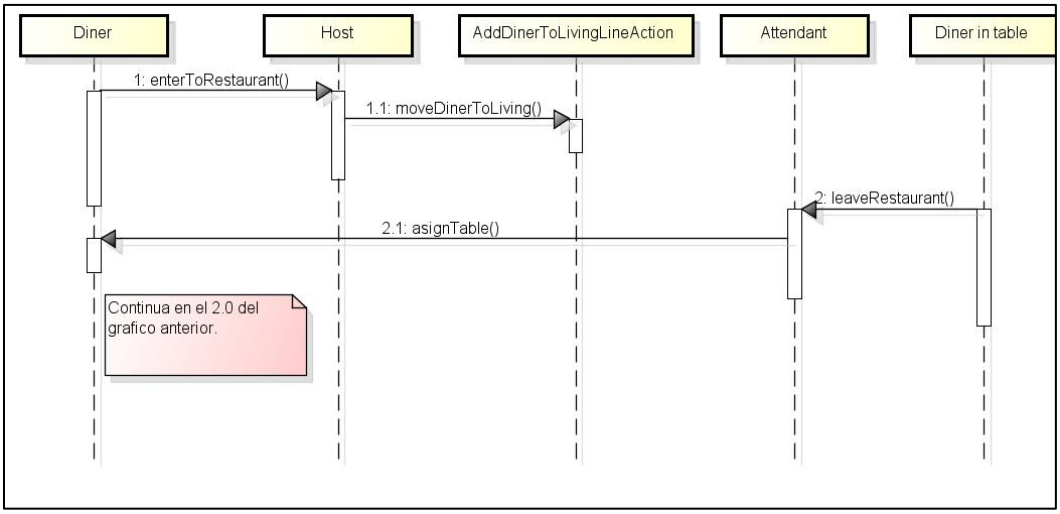


Diagrama de Secuencia para comensal que tiene que esperar en el living.

6 Comandos

La aplicación cuenta con diferentes comandos que se ejecutan en función de la opción ingresada. Los comandos son los siguientes.

6.1 Inicio

Para iniciar la aplicación debemos ejecutar lo siguiente:

```
>  ./resto-simulation-c++ -i
```

6.2 Creación de Comensales

Para crear los comensales el comando es el siguiente:

```
>  ./resto-simulation-c++ -d 10
```

El numero 10 indica el número de comensales que se generaran.

6.3 Consultas

Para las consultas los comandos son los siguientes:

- Para consultar el estado de la caja
  ```
  >  ./resto-simulation-c++ -q c
  ```
- Para consultar gente en el living
  ```
  >  ./resto-simulation-c++ -q l
  ```
- Para consultar perdidas
  ```
  >  ./resto-simulation-c++ -q p
  ```