
# *Framework de Simulación de Planificación de Procesos*

*por Emmanuel Luján*

## INFORMACIÓN GENERAL

**Fecha:** Noviembre del 2010
**Tema:** Framework de Simulación de Planificación
**Materia:** Taller de Tiempo Real para Control Robótico
**Carrera:** Ingeniería de Sistemas
**Universidad:** Universidad Nacional del Centro de la Provincia de Bs. As.
**Docentes:** Dr. Acosta Nelson
Mg. Aciti Claudio
**Autor:** APU Luján Emmanuel
**Email:** info@emmanuellujan.com.ar

## ÍNDICE DE CONTENIDO

## ÍNDICE DE FIGURAS

# 1. REQUERIMIENTOS

En la siguiente sección se detallarán los objetivos del proyecto y una descripción general del funcionamiento del sistema.

## 1.1. OBJETIVOS

- Diseñar, implementar y documentar un sistema de simulación de ejecución de procesos sobre dispositivos de un computador.
- Se debe lograr independencia en
  - Carga de datos inicial del sistema (procesos y dispositivos). Para hacer diferentes pruebas.
  - Los algoritmos de planificación. Para poder comparar su funcionamiento en base a los mismos datos iniciales.
- Se debe generar un reporte con los resultados de la ejecución.

## 1.2. DESCRIPCIÓN GENERAL

Típicamente el estado de los procesos planificados en la CPU sigue el siguiente diagrama:

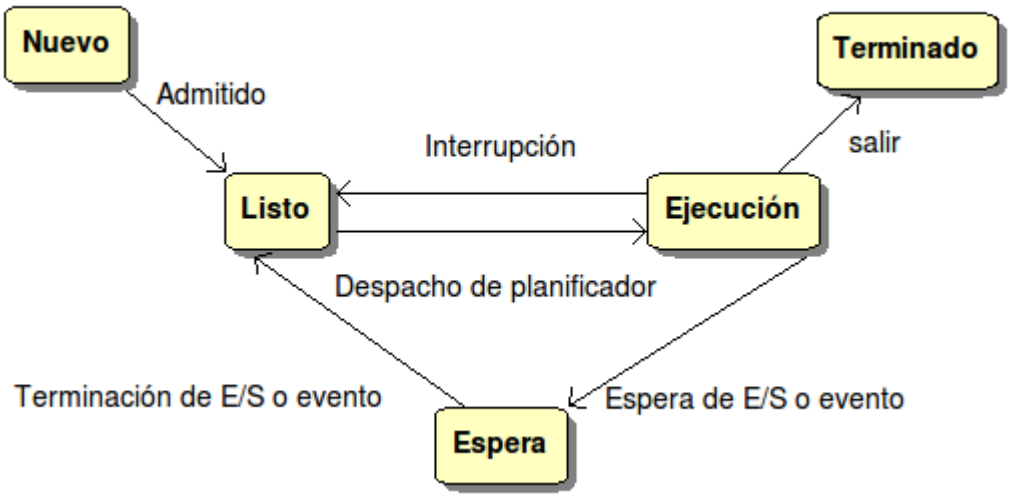


*Figura 1.1. – Diagrama de Estados de la CPU*

En el estado de ejecución puede haber sólo un proceso, en el resto de los estados puede haber varios. Un proceso en estado de ejecución o activo está corriendo en la CPU, pero en los demás estados los procesos deben ser almacenados en listas. Cuando un proceso cambia de estado va hacia otra lista o bien a ejecución.

Cuando la CPU no tiene un proceso activo entonces el planificador elige uno de la lista de listos mediante un algoritmo de planificación. El despachador se encarga de alojar dicho proceso en la CPU, o sea ponerlo como activo. Algunos algoritmos de planificación, como Round Robin, no dejan que un proceso esté en estado activo hasta que termine sino que luego de una determinada

cantidad de tiempo el proceso es cambiado por otro. Para realizar esto se utiliza un temporizador que notifica cuando la cantidad de tiempo que el proceso debe estar en ejecución ha terminado. Luego de esto el planificador elige un nuevo proceso para planificar de la lista de listos. Y finalmente el despachador realiza un cambio de contexto, o sea desaloja al proceso actual de la CPU, lo introduce en la lista de listos, toma el proceso elegido por el planificador y lo aloja en la CPU.

Cuando un proceso en ejecución necesita hacer una operación de entrada/salida sobre un dispositivo pasa al estado de espera, y por consiguiente a una lista de espera. No existe una sola lista de espera, cada dispositivo posee una propia. Cuando el proceso termina de ejecutarse en el dispositivo retorna a la lista de listos del CPU.

El CPU consta también de una lista de interrupciones. Cuando una interrupción llega a la CPU ésta se almacena en la lista de interrupciones. Luego se elige una interrupción con un algoritmo de planificación y se ejecuta dicha interrupción. Si había un proceso ejecutándose cuando esto ocurre el despachador hace un cambio de contexto y lo pasa a estado listo. Si el proceso activo es una interrupción entonces se ejecuta un algoritmo de planificación para decidir si la interrupción activa debe continuar ejecutándose o debe ser desalojada.

## 1.3. CASOS DE USO

En la siguiente sección se analizarán los requerimientos expuestos anteriormente, para ello se describirán los principales casos de uso relacionados. Si bien en este documento se presenta una visión del desarrollo del sistema top-down, éste fue realizado de acuerdo a una visión bottom-up. Primero se codificó el mismo y luego se hizo reingeniería para obtener la documentación.

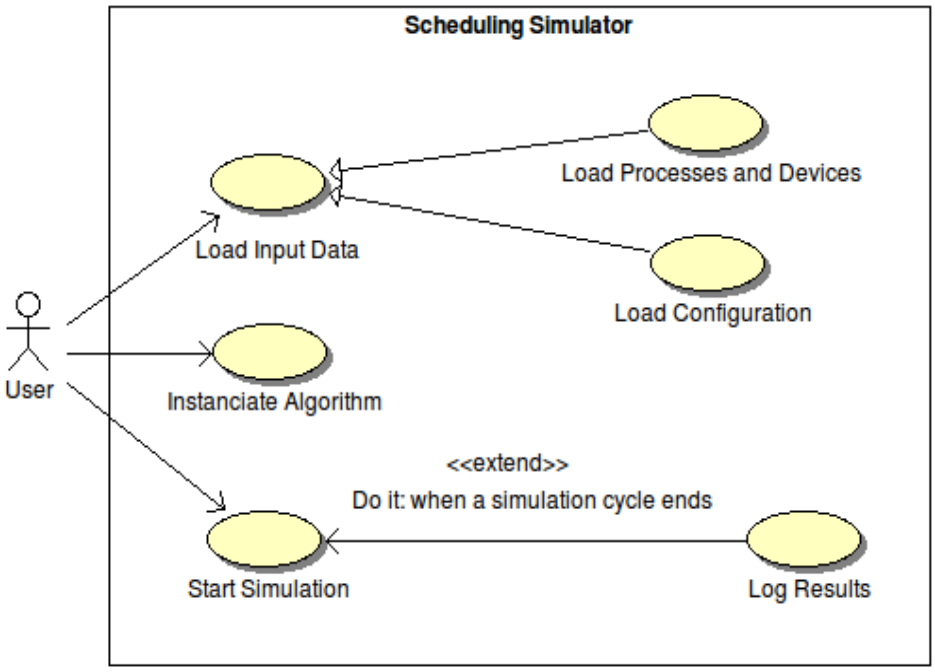


*Figura 1.2. – Diagrama de Casos de Uso del Sistema*

### 1.3.1. CARGAR DATOS DE ENTRADA (LOAD INPUT DATA)

Flujo de Eventos Principal: El caso de uso comienza cuando el usuario carga los datos que el sistema necesita para realizar la simulación ( procesos, dispositivos y opciones de configuración) y termina cuando finaliza dicha acción.

#### 1.3.1.1. CARGAR PROCESOS Y DISPOSITIVOS (LOAD PROCESSES AND DEVICES)

Flujo de Eventos Principal:

1. El caso de uso comienza cuando el usuario carga al sistema los procesos que se ejecutarán (cada uno con identificador, prioridad, un conjunto de unidades computacionales, etc).
2. El usuario carga al sistema los dispositivos que ejecutarán a los procesos (cada dispositivo tiene identificador, algoritmo de planificación, tiempo de ciclo de interrupción (quantum), etc).
3. El caso de uso termina.

Flujo de Eventos Alternativo: El usuario puede no realizar la carga de datos, el sistema provee un conjunto de datos por defecto.

#### 1.3.1.2. CARGAR CONFIGURACIÓN (LOAD CONFIGURATION)

Flujo de Eventos Principal: El caso de uso comienza cuando el usuario carga al sistema las opciones de configuración (ejemplo: cuál será la entrada de datos y cuál la salida), y termina cuando se finaliza está acción.

### 1.3.2. INSTANCIAR ALGORITMO (INSTANCIATE ALGORITHM)

Flujo de Eventos Principal:

1. El caso de uso comienza cuando el usuario diseña un algoritmo de planificación.
2. El usuario usa los puntos calientes (ver sección 4) necesarios provistos por el sistema para integrar el algoritmo a dicho sistema de simulación.
3. El caso de uso termina.

### 1.3.3. COMENZAR SIMULACIÓN (START SIMULATION)

Flujo de Eventos Principal:

1. El caso de uso comienza cuando el usuario ejecuta el sistema.
2. El sistema realiza la simulación.
3. El sistema loguea cada ciclo de simulación.
4. El sistema genera estadísticas con los datos obtenidos.
5. El caso de uso termina.

Precondición: Se debe haber hecho la carga inicial de datos antes de comenzar la simulación.

### 1.3.4. LOGUEAR RESULTADOS (LOG RESULTS)

1. El caso de uso comienza cuando el sistema termina un ciclo de simulación y necesita loguearlo.
2. El sistema almacena dicho ciclo en un formato estándar en memoria.
3. Cuando la simulación ha terminado los datos se almacenan en memoria secundaria.
4. El caso de uso termina.

Caso de Uso Comenzar Simulación
1. Punto de extensión: Do It

## 2. ANÁLISIS Y DISEÑO

### 2.1. ARQUITECTURA GENERAL

En la sección anterior se habla principalmente acerca del CPU, sin embargo los demás dispositivos del sistema tienen similares características. En pos de abstraer conceptos se supondrá que todos los dispositivos (incluyendo la CPU) pueden ejecutar un sólo proceso a la vez. Cada dispositivo posee los siguientes elementos:

- Un proceso activo.
- Una lista de listos (en la descripción anterior la lista de listos de los dispositivos de entrada/salida era llamada lista de espera).
- Una lista de interrupciones.

y responsabilidades:

- Planificar (con un algoritmo específico) la lista de listos.
- Planificar (con un algoritmo específico) la lista de interrupciones.
- Realizar los cambios de contexto.
- Gestionar interrupciones basadas en tiempos.

El siguiente diagrama de componentes muestra la arquitectura general del sistema. El mismo se divide en dos grandes partes, un controlador que comprende el sistema que gestiona la planificación, asociado con un sistema de algoritmos de planificación. Y por otro lado un modelo que comprende un sistema de entrada de datos, un log y el modelo de datos necesario.

El sistema de planificación, entonces, queda descripto de la siguiente forma:

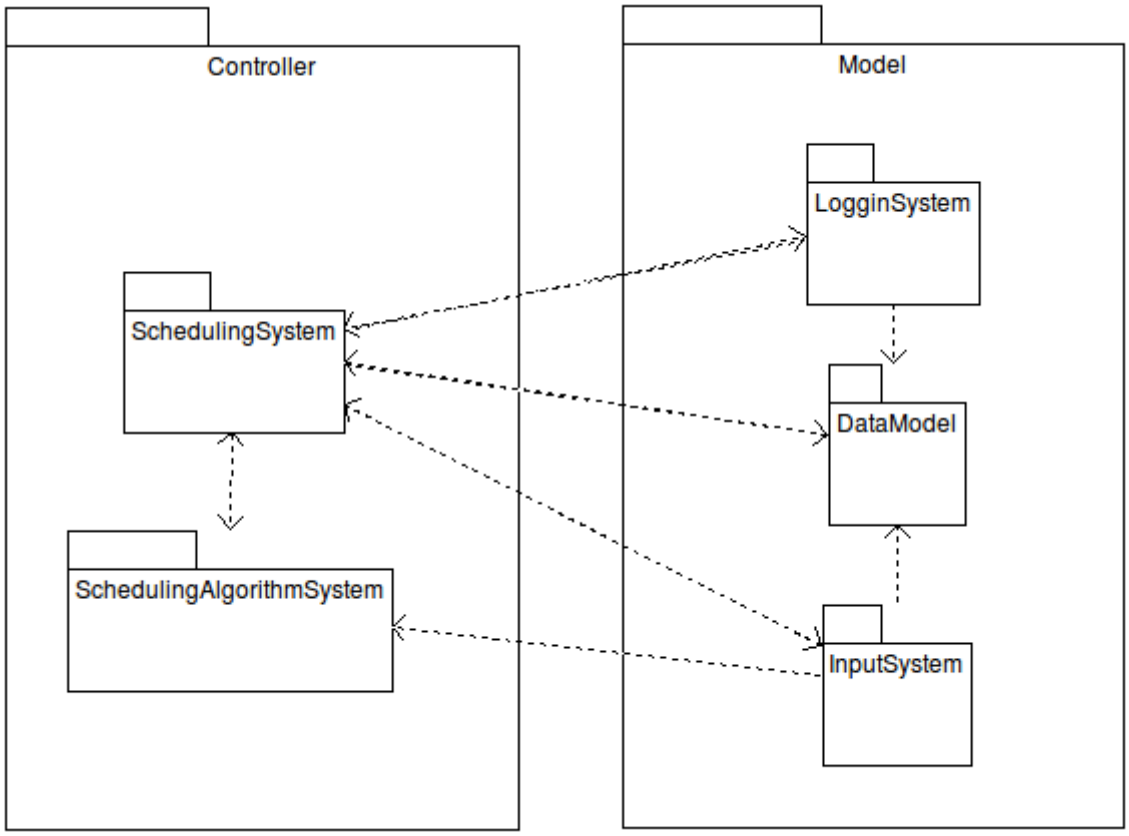


Figura 2.1 – Diagrama de Paquetes del Sistema

## 2.2. ARQUITECTURA DETALLADA

En la siguiente sección se explicarán los distintos sistemas que conforman al sistema principal. Para ello se hará uso de diagramas de clase, código, pseudocódigo y prosa.

### 2.2.1. SISTEMA DE ENTRADA (INPUT SYSTEM)

Este sistema se encarga de obtener los procesos y dispositivos para realizar la simulación. Estos datos de entrada se encuentran en un almacenamiento físico. Los datos obtenidos son almacenados en listas.

La clase InputSystem tiene dos métodos abstractos: *loadNewsList* y *loadDevicesList*. La clase XMLInputSystem hereda de la anterior e implementa dichos métodos obteniendo la información de archivos XML. Esto facilita la modificación de la forma de almacenamiento de los procesos y dispositivos.

La clase Configurator es necesaria para obtener la dirección del archivo de entrada. Las demás clases son necesarias para crear las listas.

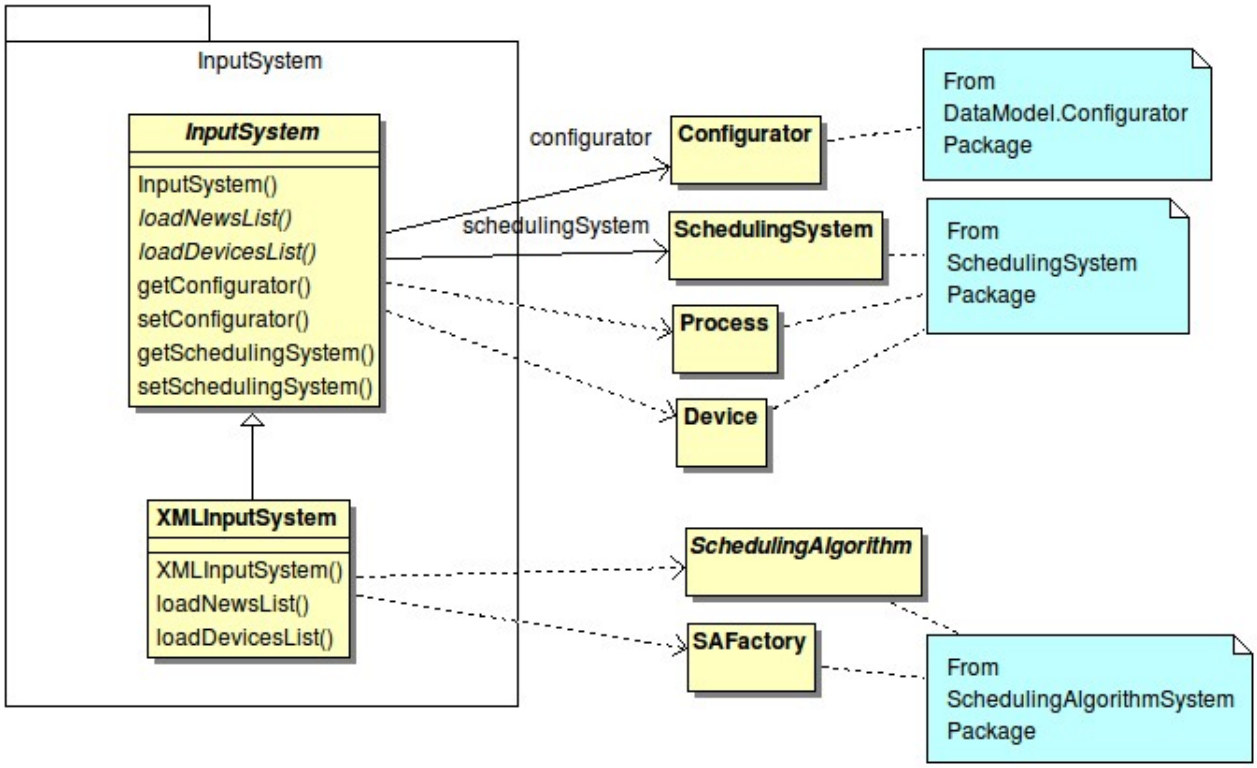


*Figura 2.2 – Diagrama de Clases, Sistema de Entrada*

### 2.2.2. SISTEMA DE ALGORITMOS DE PLANIFICACIÓN

Este sistema contiene los algoritmos de planificación que puede usar el simulador. La clase SchedulignAlgorithm posee el método abstracto *schedule* que recibe como parámetro una lista de procesos y brinda como salida al proceso escogido en la planificación.

Las clases PrioritiesSA y FCFS heredan de SchedulingAlgorithm e implementan, como sus nombres lo indican, un algoritmo por prioridades y el primero en llegar es el primero en salir respectivamente. El usuario del simulador puede heredar clases de SchedulingAlgorithm para probar sus algoritmos.

La clase SAFactory se encarga de asociar una etiqueta a cada algoritmo. El Sistema de Entrada utiliza esto en la creación de la lista de procesos nuevos. Por tanto cuando se agrega un algoritmo también se debe agregar una entrada en SAFactory. La etiqueta es usada en los archivos XML que guardan la información de entrada. Esta clase puede dejar de ser usada si se automatiza la creación de objetos mediante etiquetas.

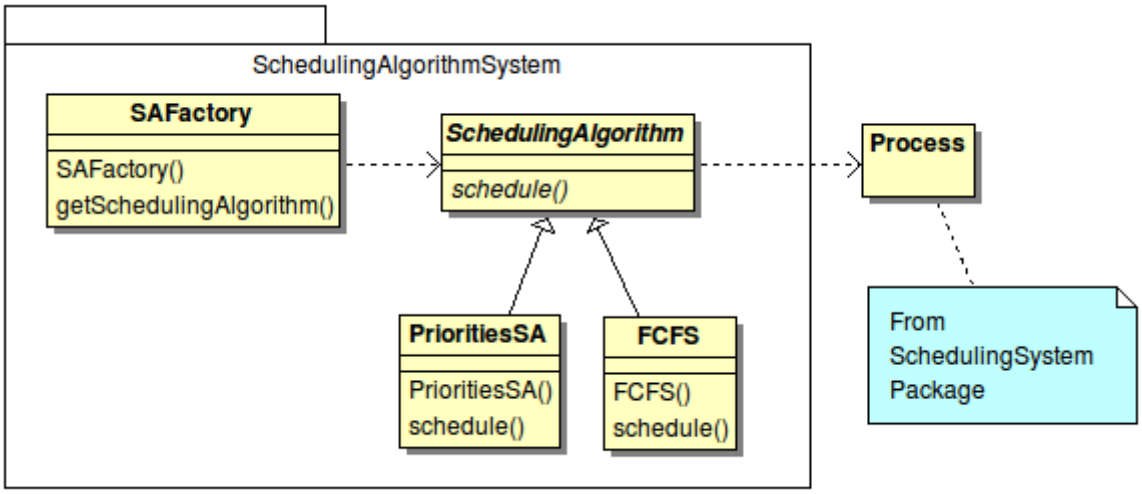


*Figura 2.3 – Diagrama de Clases, Algoritmos de Planificación*

### 2.2.3. SISTEMA DE PLANIFICACIÓN

El proceso de simulación debe emular la ejecución paralela de los dispositivos. Para ello se realizan ciclos de simulación. En cada ciclo cada dispositivo se ejecuta una fracción de tiempo. Debido a que los dispositivos interactúan entre ellos es necesario que estén sincronizados. Para lograr ese sincronismo las ejecuciones antes nombradas deben tardar la misma cantidad de tiempo t0. El algoritmo desarrollado asume que en cada ciclo un dispositivo puede realizar una de estas ejecuciones:

- Planificar y pasar a estado activo el proceso o interrupción planificado/a .
- Interrupción por temporizador (desalojar proceso activo y reiniciar temporizador) .
- Alojar un proceso en el dispositivo en dónde debe ejecutarse .
- Ejecutar un proceso (una unidad computacional) .
- Finalizar un proceso .
- No hacer nada .

A fines prácticos es irrelevante qué es lo que ejecuta un proceso. Por ello se abstrae dicho comportamiento y se asume que los procesos realizan computaciones. Cada proceso está compuesto por un conjunto de unidades computacionales. Cada unidad computacional requiere ser ejecutada en un dispositivo específico y tarda una cantidad de tiempo t0.

La clase SchedulingSystem administra los diferentes sistemas. Es la encargada de realizar los ciclos de simulación y de sincronizar las interacciones entre los dispositivos (por ejemplo que los pasajes de un proceso de una lista a otra se haga al fin del ciclo). La clase Device contiene la información de un dispositivo y se encarga de realizar una ejecución en cada ciclo. La mayor parte del algoritmo de simulación se encuentra allí. Finalmente la clase Process contiene la información referente a un proceso.

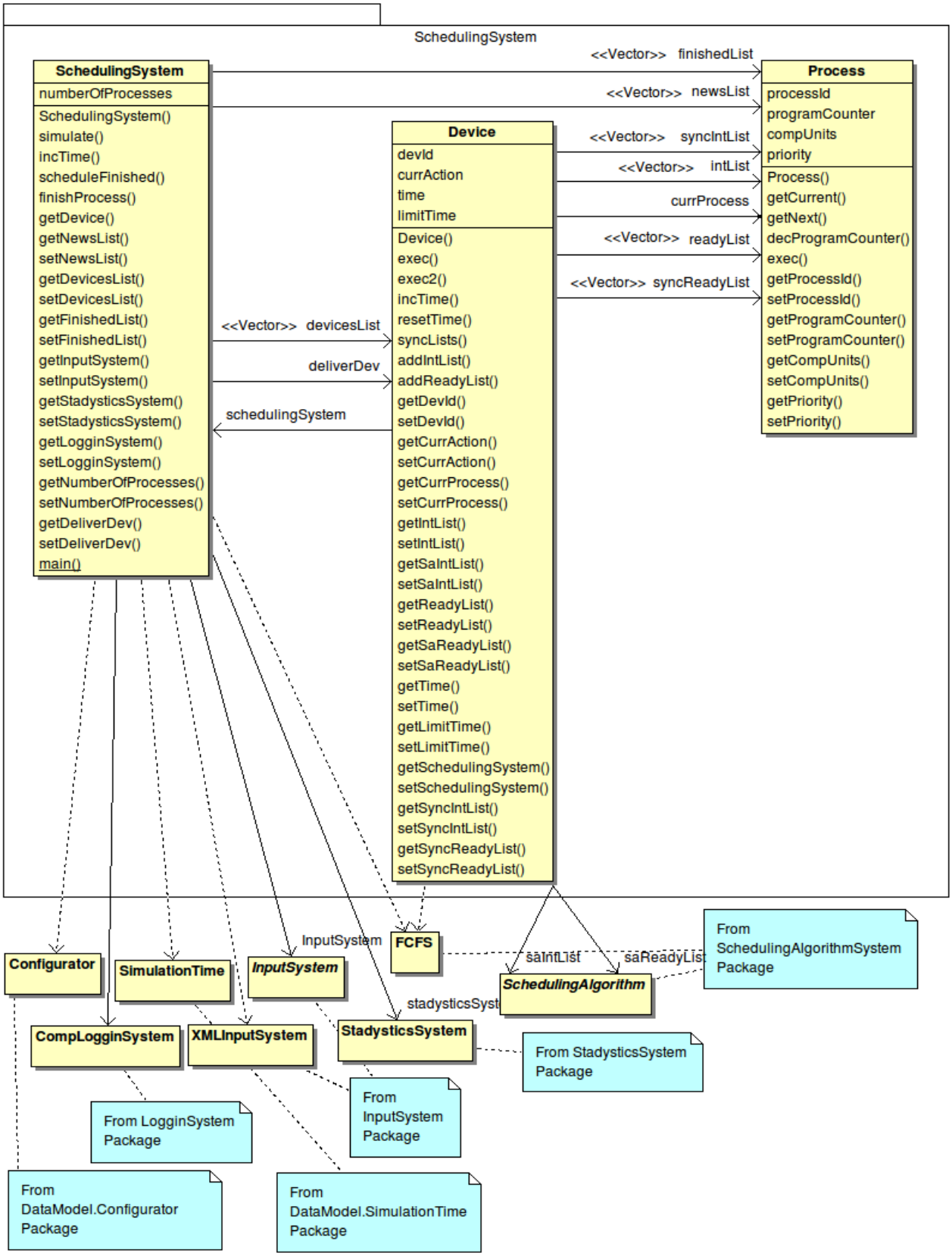


*Figura 2.4 – Diagrama de Clases, Simulación*

#### 2.2.3.1. ALGORITMO DE SIMULACIÓN

Para simular la ejecución en paralelo de los dispositivos y sus procesos se utilizará un algoritmo iterativo. Como se dijo antes, en cada ciclo cada dispositivo se ejecuta una fracción de tiempo t0. El algoritmo de simulación se ejecuta mientras haya procesos que deban ejecutarse y por cada dispositivo. En azul se establecen las sentencias de control y en verde las ejecuciones:

*Simular:*

- *Hasta que no haya más procesos por ejecutar hacer:*
  - *Por cada dispositivo hacer:*
    - *Si no hay un proceso activo*
      - *Si el temporizador ha terminado:*
        - *Se reinicia el temporizador*
      - *Sino, si la lista de interrupciones no está vacía:*
        - *Se ejecuta el algoritmo de planeamiento de interrupciones para elegir un proceso.*
        - *Se elimina dicho proceso de la lista de interrupciones.*
        - *El proceso pasa a estado activo.*
      - *Sino, si la lista de listos no está vacía:*
        - *Se ejecuta el algoritmo de planeamiento para elegir un proceso de la lista de listos.*
        - *Se elimina dicho proceso de la lista de listos.*
        - *El proceso pasa a estado activo.*
      - *Sino:*
        - *No hacer nada*
    - *Sino hay un proceso activo*
      - *Si el temporizador ha terminado:*
        - *Se obtiene la unidad computacional (que informa cuál es el próximo dispositivo en el que se ejecutará el proceso)*
        - *Si el proceso en ejecución no es una interrupción:*
          - *Se lo anexa a la lista de listos (pasa a estado listo).*
        - *Se reinicia el temporizador.*
      - *Sino, si la lista de interrupciones no está vacía:*
        - *Se ejecuta el algoritmo de planeamiento de interrupciones para elegir un proceso.*
        - *Si el proceso no es una interrupción o bien sí lo es y además tiene mayor prioridad que el proceso actual*
          - *Se elimina dicho proceso de la lista de interrupciones.*
          - *Se agrega el proceso actual a la lista de listos o a la lista de interrupciones dependiendo de lo que sea*
          - *El proceso pasa a estado activo.*
        - *Sino*
          - *Ir a Simular2*

- *Sino*
  - *Ir a Simular2*

*Simular2:*

- *Si el proceso en ejecución ha llegado a su fin:*
  - *Terminar proceso (pasa a estado finalizado).*
- *Sino, si el proceso en ejecución debe continuar ejecutándose en este dispositivo:*
  - *Ejecutar proceso.*
- *Si es una interrupción:*
  - *Se decrementa el contador de programa*
  - *Se lo anexa a la lista de interrupciones del dispositivo indicado (el proceso pasa a ser una interrupción y pasa a estado de espera).*
  - *Se desaloja el proceso en ejecución.*
- *Sino, si el proceso en ejecución debe continuar ejecutándose en otro dispositivo:*
  - *Se decrementa el contador de programa*
  - *Se lo anexa a la lista de listos del recurso indicado (pasa a estado de espera).*
  - *Se desaloja el proceso en ejecución.*

### 2.2.4. SISTEMA DE LOGUEO

Este sistema está encargado de registrar todas las actividades que se realizan en cada ciclo de simulación. El Sistema de Planificación usa a este sistema para dicho fin. La información es almacenada en memoria principal hasta que se pide que la misma pase a memoria secundaria. Para realizar esto último el sistema cuenta con más de una codificación disponible, XML y TXT. En caso de ser requerido se pueden agregar nuevos medios de almacenamiento, como una base de datos.

La clase LogginSystem es abstracta, tiene implementado el metodo *log*, que es el que guarda la información de un ciclo en memoria principal. Tiene un método abstracto *writeLog*, que es el que almacena la información en memoria secundaria. La clase FileLogginSystem abstrae el comportamiento para almacenar información en disco en forma de archivos. Las clases XMLLogginSystem y TXTLogginSystem, como es esperado, almacenan la información en estos formatos. Por último la clase CompLogginSystem posee un conjunto de LogginSystem que le permite almacenar en disco la información en varios formatos. En este caso en particular cuando el LogginSystem ejecuta el método *writeLog* de la instancia CompLogginSystem la información que se pasa a disco es en formato XML y TXT.

La clase Configurator es usada para conocer dónde se deben almacenar los resultados. La clase Device es necesaria para obtener la información de ese dispositivo en un momento particular. Finalmente la clase SimulationTime almacena un ciclo de simulación.

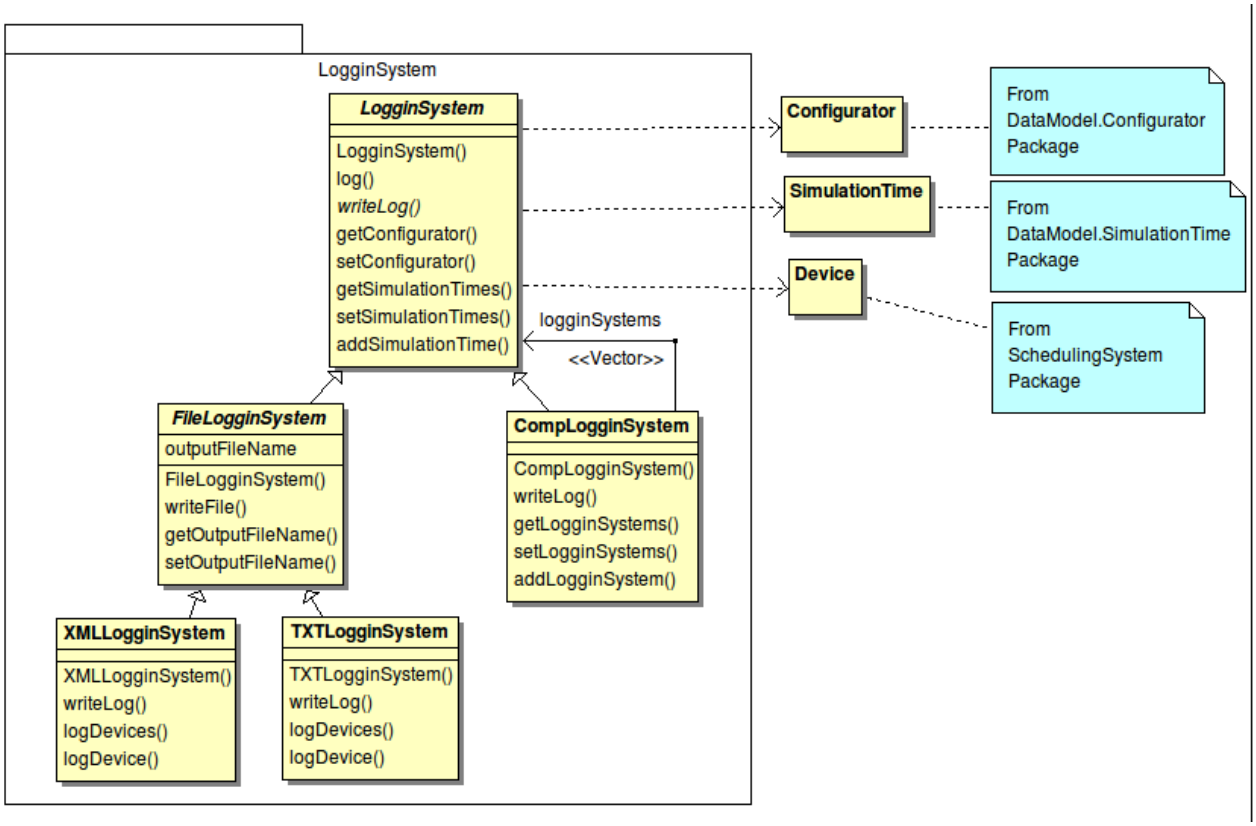


*Figura 2.5 – Diagrama de Clases, Logueo de Resultados*

### 2.2.5. MODELO DE DATOS

El modelo de datos representa la información que será almacenada. Por un lado la clase Configurator tiene la información relacionada con las direcciones de la información de entrada (procesos y dispositivos), y de salida (reportes generados en base a la simulación). Por otro lado la clase SimulationTime contiene la información necesaria para almacenar un ciclo de simulación. Cada ciclo almacena el estado de cada dispositivo, se hace necesaria la clase SimulationDevice. Estas clases dependen de la clase Device y Process.

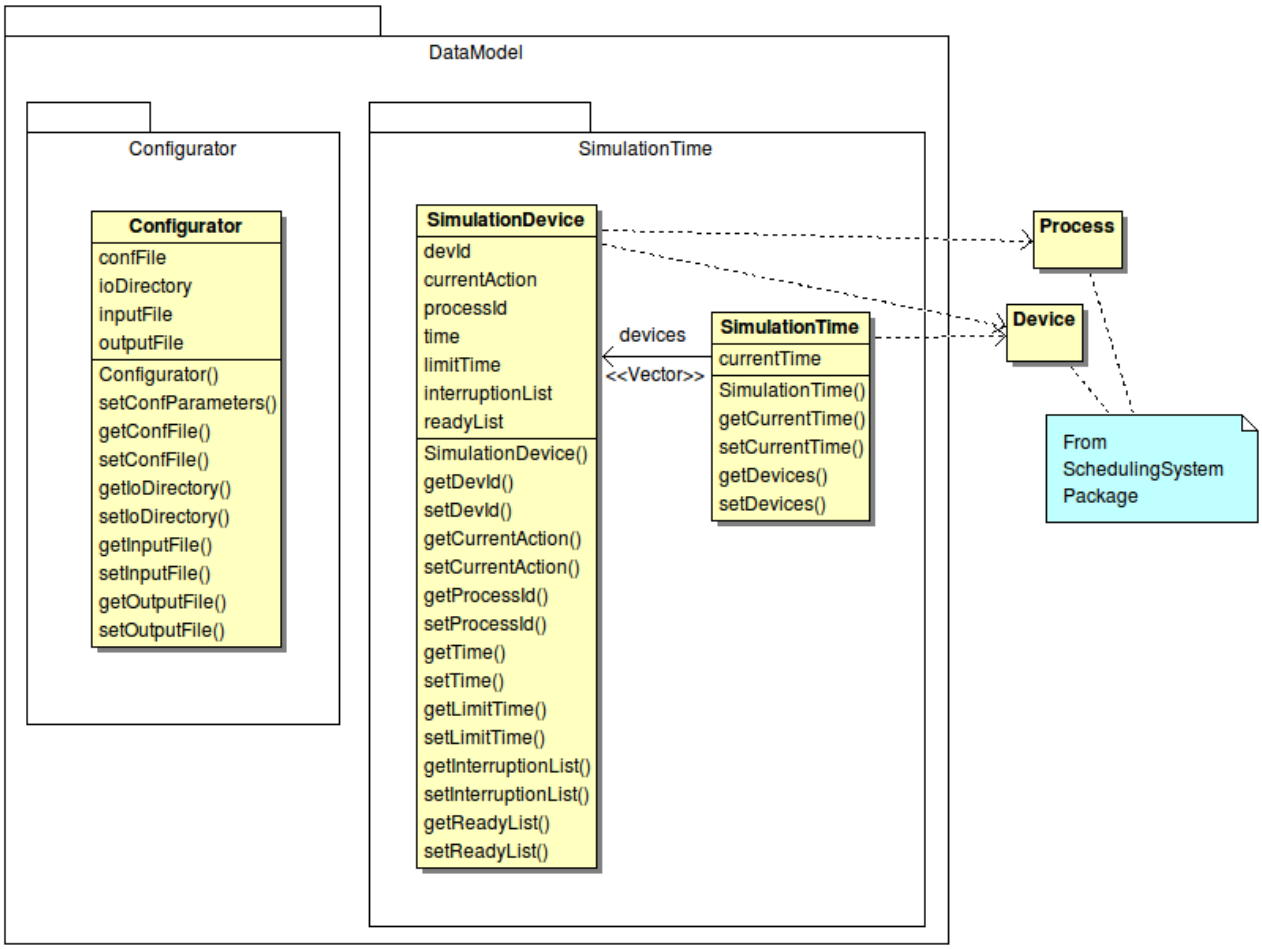


*Figura 2.6. – Diagrama de Clases, Resultados*

# 3. USO DEL FRAMEWORK

## 3.1. ACERCA DE FRAMEWORKS

Un framework es un conjunto de clases cooperativas que construyen un diseño parcial y reusable con el cual las aplicaciones dentro de un dominio son creadas a través de especializaciones. De la misma forma que una clase abstracta auna una familia de clases con características en común, un framework auna un conjunto de aplicaciones con características en común.

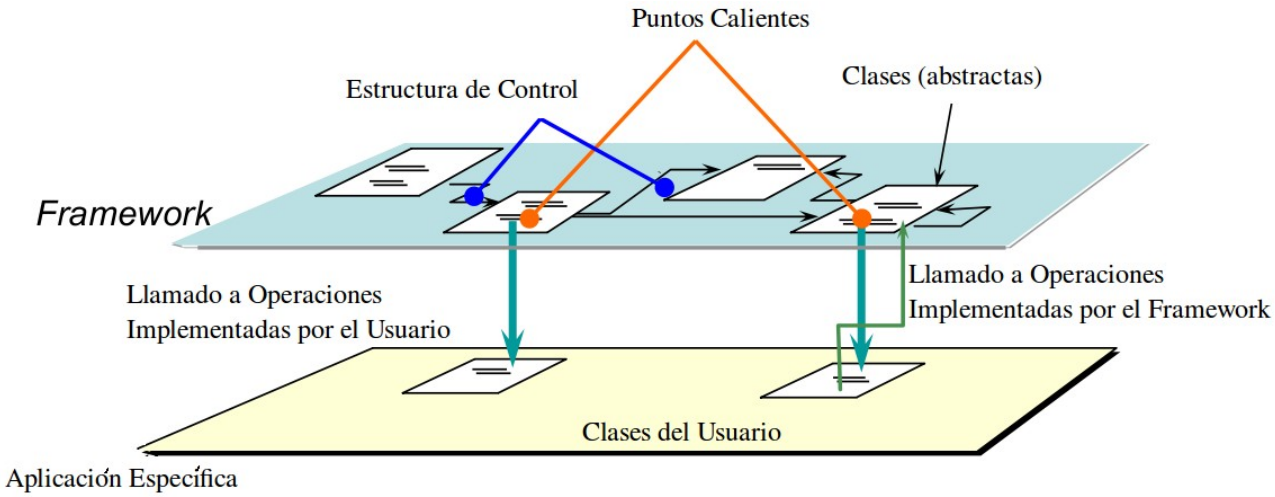


*Figura 3.1. – Conceptos Básicos de Frameworks*

Al instanciar un framework se genera una aplicación particular. El control principal de la aplicación no se encuentra en la instanciación, sino en el framework. Como se puede ver en el gráfico de arriba las estructuras de control están en el framework. Esta propiedad se llama inversión de control. Por otro lado para instanciar un framework se necesita implementar métodos abstractos definidos dentro del framework, estos métodos se llaman puntos calientes.

A veces los métodos que necesitan ser instanciados poseen cierta estructura que deben respetar. Para solucionar esto dichos métodos (que están dentro del framework) se implementan, por lo que no son abstractos, en función de otros métodos que sí son abstractos. El método implementado parcialmente (en pos de que se respete la estructura requerida) se denomina método plantilla y los métodos que usa el método plantilla, que sí son abstractos, se llaman métodos gancho. La clase que pertenece al framework se denomina clase plantilla y la que implementa los métodos gancho se llama clase gancho.

Por último los frameworks de caja blanca son aquellos en donde en la especialización se usa herencia; mientras que en los de caja negra en la especialización se usa composición.

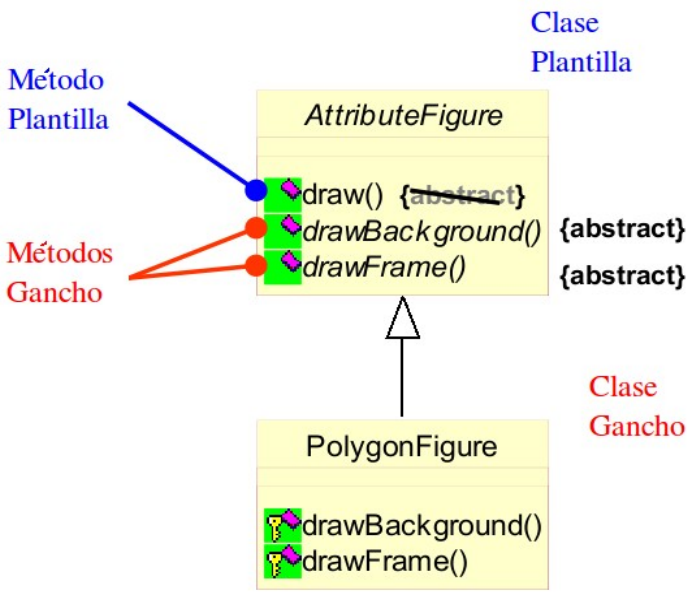


*Figura 3.2. – Métodos y Clases Plantilla y Gancho*

## 3.2. CREAR UN ALGORITMO DE PLANIFICACIÓN

Básicamente se debe subclasear de SchedulingAlgorithm. A manera de ejemplo se mostrará la implementación de la clase FCFS (primero en llegar primero en ser servido):

```
public class FCFS extends SchedulingAlgorithm {

        public FCFS(){}

        public Process schedule(Vector<Process> processes){
                if(processes.size()>0)
                        return processes.get(0);
                else
                        return null;
        }

}
```

## 3.3. INTRODUCIR DATOS DE ENTRADA

Los datos de entrada comprenden la descripción de los procesos y dispositivos que serán ejecutados. Los procesos tienen id, prioridad, un conjunto de unidades computacionales, etc. Los dispositivos tienen id, algoritmo de planificación, quantum, etc. Para ello se debe crear un archivo XML con dicha información. A continuación se muestra un ejemplo puntual:

```
<?xml version="1.0"?>
<!-- dev0 is the cpu -->
<!-- priority must be >= 0 -->
<!-- if quantum is not needed set it to -1 -->
<inputData>
        <description>  Introducir descripción </description>
        <processes>
                <process>
                        <processId>proc0</processId>
                        <priority>0</priority>
                        <compUnits>
                                <compUnit>dev0</compUnit>
                                <compUnit>dev1</compUnit>
                                <compUnit>dev0</compUnit>
                        </compUnits>
                </process>
                <process>
                        <processId>proc1</processId>
                        <priority>2</priority>
                        <compUnits>
                                <compUnit>dev1</compUnit>
                                <compUnit>dev1</compUnit>
                                <compUnit>dev0</compUnit>
                        </compUnits>
                </process>
        </processes>
        <devices>
                <device>
                        <deviceId>dev0</deviceId>
                        <schedulingAlgorithm>PrioritiesSA</schedulingAlgorithm>
                        <quantum>5</quantum>
                </device>
                <device>
                        <deviceId>dev1</deviceId>
                        <schedulingAlgorithm>FCFS</schedulingAlgorithm>
                        <quantum>-1</quantum>
                </device>
        </devices>
</inputData>
```

### 3.4. LEER DATOS DE SALIDA

Se generan dos archivos con distinto formato, XML y TXT, pero con la misma información. El primer archivo está pensado para si se quiere trabajar con esa información, por ejemplo para la realización de estadísticas. El segundo para ser legible si se quiere hacer una lectura de la ejecución. A continuación se mostrará un fragmento del resultado de una ejecución de en TXT:

```
...
Time: 21
        DeviceId: deliverDev
                Current Action: None
                Active Process: None
                Current Time: 21
                Limit Time: -1
                Interruption List:
                Ready List:

        DeviceId: dev0
                Current Action: Select a process from the ready list and put that process as active.
                                The process selected is proc1
                Active Process: proc1
                Current Time: 11
                Limit Time: 5
                Interruption List:
                Ready List:

        DeviceId: dev1
                Current Action: The process proc0 pass to the ready list of the device dev0
                Active Process: None
                Current Time: 21
                Limit Time: -1
                Interruption List:
                Ready List:

        DeviceId: dev2
                Current Action: Processing active process proc2
                Active Process: proc2
                Current Time: 21
                Limit Time: -1
                Interruption List:
                Ready List:

Time:22
…
```

# 4. PRÓXIMOS REQUERIMIENTOS

- Manejo de errores. (ejemplo: carga de datos inválida)
- Interfaz gráfica. Sistema de entrada y salida de datos más intuitivos.
- Completar la salida de datos en base a lo requerido por estadísticas.
- Estadísticas. En base a los datos producidos obtener información como:
  - El tiempo y cantidad de ejecuciones del algoritmo de planificación de cada recurso.
  - El tiempo y cantidad de desalojos de cada recurso.
  - El tiempo y cantidad de ejecuciones de cada proceso de cada recurso.
  - El tiempo de inactividad de cada recurso.
  - El tiempo total de la ejecución.
  - El tiempo total de cada proceso.
- Mejorar el algoritmo de simulación. Se propone el agregado de una lista de procesos dormidos, la lógica para el manejo de deadlines y el agregado de dos funciones, al comienzo y fin de cada ciclo para que el usuario pueda instanciar el framework de una manera más flexible. Los siguientes diagramas de actividades ilustran como podría ser el algoritmo:

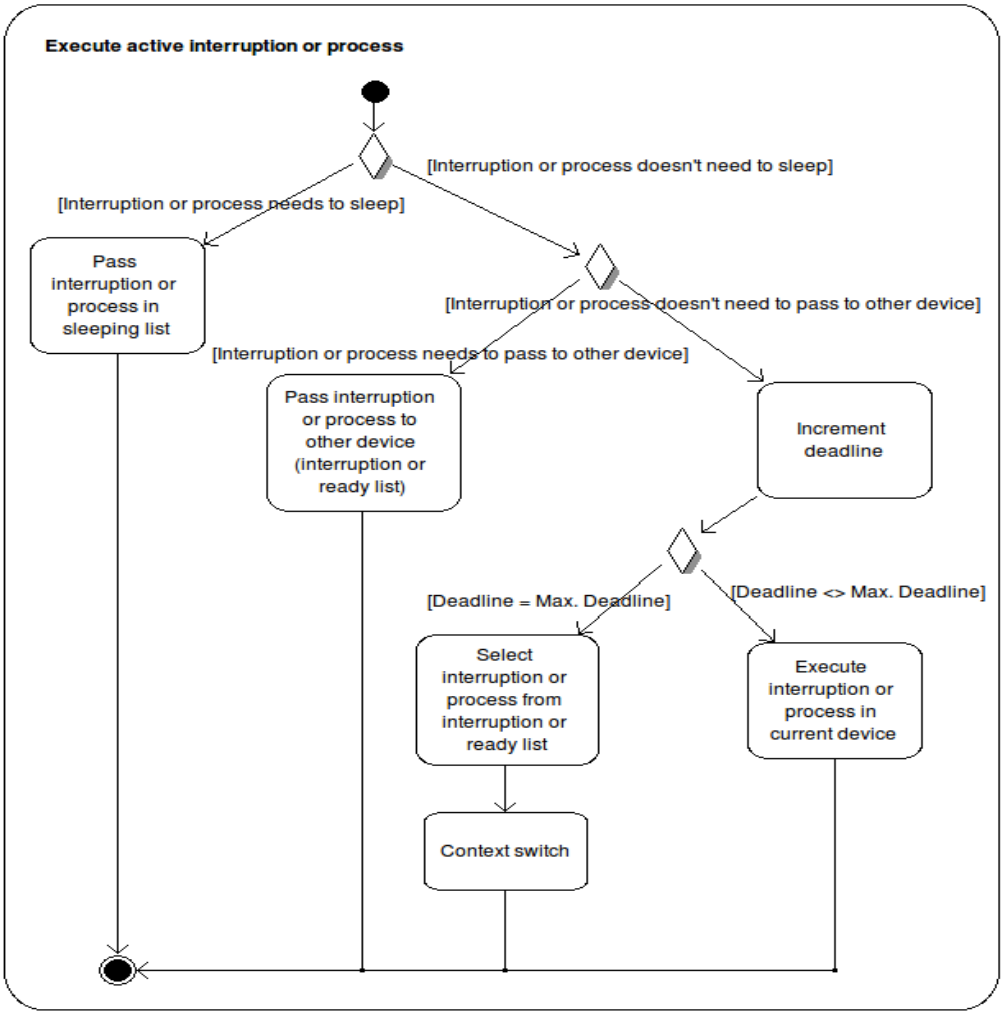


*Figura 4.1. – Diagrama de actividades: Ejecutar interrupción o proceso activo*

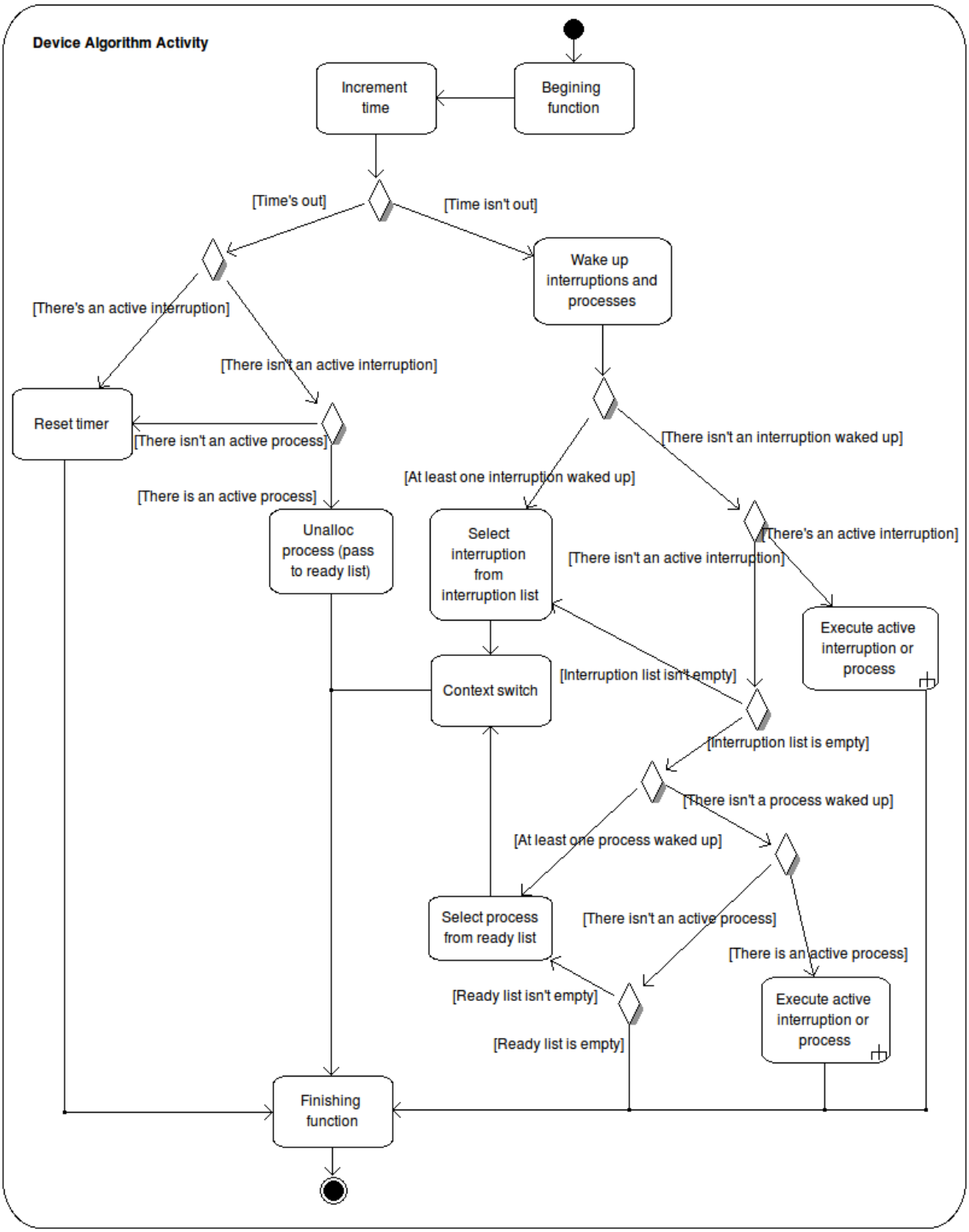


*Figura 4.2. – Diagrama de actividades: Algoritmo del Dispositivo*

## 5. CONCLUSIÓN

Se realizó un framework de simulación de planificación de procesos sobre dispositivos. El mismo se puede instanciar gracias al agregado (mediante subclaseo) de algoritmos de planificación.

La información de entrada (procesos y dispositivos) se almacena en formato XML y la de salida en formato XML y TXT. El diseño deja extender y cambiar las funcionalidades en varios puntos, como los formatos de almacenamiento de información o el agregado de nuevos algoritmos de planificación. También quedan establecidas nuevas funcionalidades por realizar, como un sistema de entrada de datos más completo y un sistema de procesamiento de la información de salida capaz de generar estadísticas para comparar las diferentes simulaciones de planificación.

## 6. BIBLIOGRAFÍA


- Página web de la materia Taller de Tiempo Real para Control Robótico. UNICEN. http://www.exa.unicen.edu.ar/catedras/rtlinux/
- Página web de la materia Diseño de Sistema de Software. 2007. UNICEN. http://www.exa.unicen.edu.ar/catedras/disenio/
- Sistemas Opertivos. 6a. Edición. Silberschatz, Galvin, Gagne.
- Wikipedia.