Metodologías, procesos y entornos para sistemas de tiempo real

Master de Computación

Patrones de diseño para aplicaciones de tiempo real

José M. Drake
Computadores y Tiempo Real

Santander, 2010

1

Cocepto de patrón de diseño

- Un patrón de diseño de tiempo real es una solución probada que se puede aplicar con éxito en el diseño de aplicaciones de tiempo real.
- Es un esqueleto básico que cada diseñador adapta a las peculiaridades de su aplicación y deben cumplir las siguientes características:
  - Se debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores.
  - Deben ser reutilizables, esto es, se han de poder aplicar a diferentes problemas de diseño en distintas circunstancias.
- Hay diferentes puntos de vista para definir lo que es un patrón de diseño:
  - Pueden abordar aspectos de muy bajo nivel, como ocurre en el diseño de algoritmos de búsqueda u ordenamiento de listas.
  - Otros patrones son complejos subsistemas parametrizados de determinados dominios, scadas (*Supervisory Control And Data Acquisition*), comunicaciones, etc.
  - Aquí, los patrones de diseño son descripciones de conjuntos reducidos de clases y objetos que colaboran para resolver un problema de diseño general.

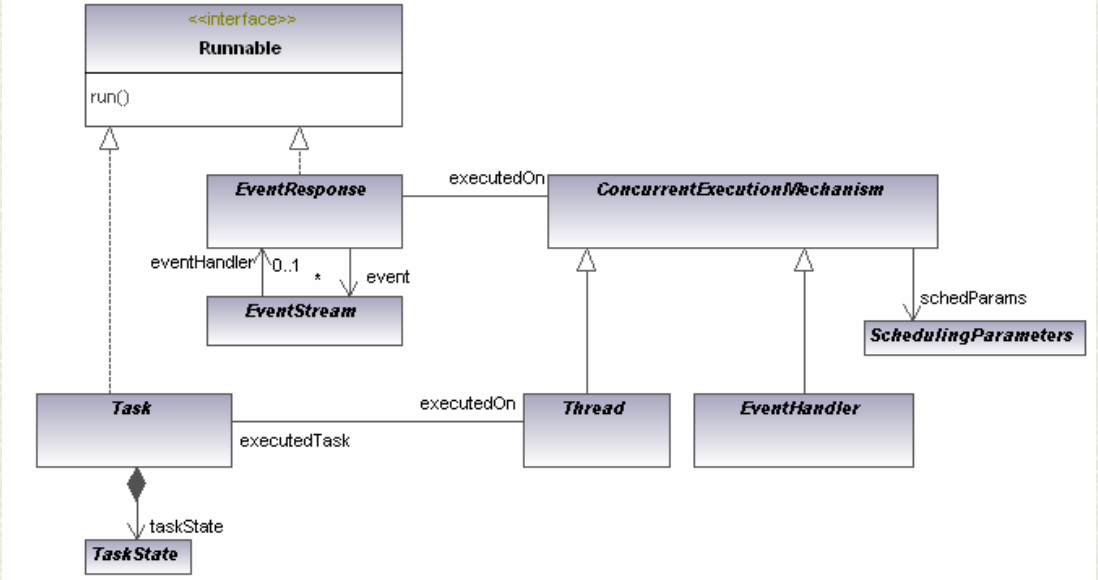
Elementos de un patrón de diseño.

- **El nombre**: permite describir mediante una o dos palabras un problemas de diseño junto con sus soluciones y consecuencias.
- **El problema**: describe el contexto en el que aplicar el patrón.
- **La solución**: describe los elementos que constituyen el diseño, las relaciones entre ellos, sus responsabilidades y sus colaboraciones.
- **Las consecuencias**: son los resultados que obtienen con su aplicación, haciendo hincapié en sus ventajas e inconvenientes.

Términos relativos a la concurrencia

- **Tarea (*Task*)**: actividad de duración prolongada, con estado y con intervalos de habilitación y deshabilitación. Mientras está habilitada, compite con otras tareas por el acceso a recursos compartidos.
- **Stream de eventos (*EventStream*)**: secuencia potencialmente infinita de eventos de un determinado tipo, que provocan la ejecución de respuestas o tareas.
- **Interrupción (*Interrupt*)**: Tipo especial de evento generado por una línea hardware del procesador.
- **Respuesta a evento (*EventResponse*)**: Actividad de vida corta, sin estado y con características de *run-to-completion*, esto es, una vez lanzada no se puede deshabilitar hasta su terminación. Así mismo, entre la respuesta a un evento y a otro no se guarda estado ni información.
- **Thread**: Hilo de ejecución concurrente proporcionado por el sistema operativo para la ejecución tanto de tareas, como en ocasiones, de respuestas a eventos.
- **Manejador de evento (*EventHandler*)**: Mecanismo de ejecución concurrente proporcionado por el sistema operativo, diferente a los *threads*, que generalmente se utilizan para implementar respuestas a eventos hardware o temporizados.
- **Trabajo (*Job*)**: Cada una de las ejecuciones de una tarea o una respuesta a un evento en un *thread* o manejador de evento del sistema.

Elementos que implementan la concurrencia.



Términos relativos al despliegue y ejecución

- **Plan de despliegue *(Deployment Plan)***: Información que describe la asignación de los thread o módulos a los nudos de procesamiento de la plataforma distribuida en la que se ejecuta la aplicación. Incluye el mecanismo de comunicación entre thread o módulos si la plataforma tiene mecanismos de comunicación alternativos.
- **Código ejecutable *(Execution Code)***: Código de la aplicación en un formato adecuado para que pueda ser ejecutado por el ejecutivo (run-time) de un nudo procesador. Puede recibir como parámetros de ejecución la configuración de despliegue y/o la configuración de planificabilidad.
- **Partición *(Distributed Partition)***: Sección del código ejecutable de la aplicación que se ejecuta en un determinado nudo de la plataforma de ejecución, cuando ésta es distribuida.
- **Configuración de despliegue *(Deployment Configuration)*:** Conjunto de datos que son pasados en la ejecución de una aplicación y que establecen la asignación de los recursos de la plataforma a los elementos lógicos de la aplicación. Cuando la aplicación es distribuida pueden existir una configuración de despliegue específico por cada partición.
- **Configuración de planificabilidad *(Scheduling Configuration)*:** Conjuntos de datos que son pasados en la ejecución de una aplicación y que establecen los valores de los parámetros de planificación que deben ser asignados en a los threads, manejadores de eventos, sesiones de comunicación y mecanismos de sincronización para que sea planificable la aplicación que se ejecuta. Puede ser una sección de la configuración de despliegue.

Términos relativos al modelo de tiempo real

- **Situación de tiempo real (*Real-time situation*):** Modo de operación de una aplicación de tiempo real. Definida por: (1) Los eventos que atiende, (2)el patrón de activación de los eventos, y (3) los requisitos temporales de las respuestas. Es el objeto del análisis planificabilidad.
- **Modelo de tiempo real *(Real-time model)*:** Abstracción que describe el comportamiento temporal de un sistema de tiempo real, de la plataforma en que se ejecuta y del requerimiento de actividad que le requiere el entorno. Se utiliza como base del análisis de planificabilidad de la aplicación.
- **Modelo de la plataforma (Real-time platform model):** Sección del modelo de tiempo real que es independiente de la aplicación en sí. Describe la capacidad de los recursos de la plataforma, así como la parte de la esta capacidad que es consumida por tareas de background (atención a los timers, drivers, planificador, etc.) que se ejecutan en ella, e incluso incluye el modelo de otras aplicaciones diferentes a la que se analiza, que se va a ejecutar en la plataforma concurrentemente con ella.
- **Modelo de los módulos lógicos (*Real-time logic model*):** Sección del modelo de tiempo real, que contiene los modelos de los elementos que constituyen la aplicación. Describe la capacidad de procesamiento que requiere la ejecución de su código, las características de los elementos de sincronización que garantiza el acceso seguro a sus datos, y en ciertos casos de los threads y tareas que se introducen con la instanciación de los módulos de la aplicación.
- **Modelo reactivo (*Real-time reactive model*):** Modelo parametrizado compuesto por la descripción que puede atender una aplicación. Es la parte del modelo que se deriva de la aplicación y no del entorno sobre el que opera.
- **Modelo de carga de trabajo (*Workload model*):** Sección del modelo reactivo que describe el patrón temporal de generación de eventos que va a ser requerido a la aplicación. Es un modelo del entorno.
- **Transacción (*End to End Flow Transaction*):** Sección del modelo reactivo que describe la respuesta de la aplicación a un determinado tipo de evento. Describe el tipo de evento del que es respuesta, las tareas que componen la respuesta y las relaciones de flujo que existen entre ellas, y los requisitos temporales que se deben cumplir.
- **Requisitos temporales (*Timing requirement*):** Especificación de los requisitos temporales que deben ser satisifechos por la aplicación. Pueden ser relativos a la finalización de una tarea de la respuesta con referencia al instante en que se generó el evento que la inició (requisito temporal global), de la duración de la ejecución de una tarea de la respuesta (requisito temporal local), a la variabilidad de los tiempos de respuesta o ejecución (jitter) o a las tasas de cumplimiento o incumplimiento de los plazos.

Términos relativos al análisis y diseño de tiempo real.

- **Análisis de planificabilidad (*Schedulability Analysis*):** Evaluación de los tiempos de respuesta de peor caso que se van a obtener en la ejecución de la aplicación de acuerdo con el modelo de tiempo real que sirve de base. En el caso de que éstos sean inferiores a los establecidos en los requisitos temporales, se puede garantizar que la aplicación va a ser planificable durante su ejecución.
- **Asignación optima de prioridades (*Optimized Priority Assigment*):** Permite evaluar a partir del modelo de tiempo real los valores de los parámetros de planificación que pueden ser asignados a los threads, canales de comunicación y elementos de sincronización para alcanzar la planificabilidad de la aplicación.
- **Cálculo de holguras (*Slack Calculus*):** Permite evaluar a partir del modelo de tiempo real la holgura con la que se satisfacen los requisitos temporales, o lo que falta para que se satisfagan. La holgura puede ser determinada para los recursos, para los elementos lógicos (transacciones y tareas) o para el sistema como un conjunto. La holgura se define como el tanto por ciento en el que se puede incrementar el tiempo de ejecución del elemento, manteniendo la aplicación planificable.

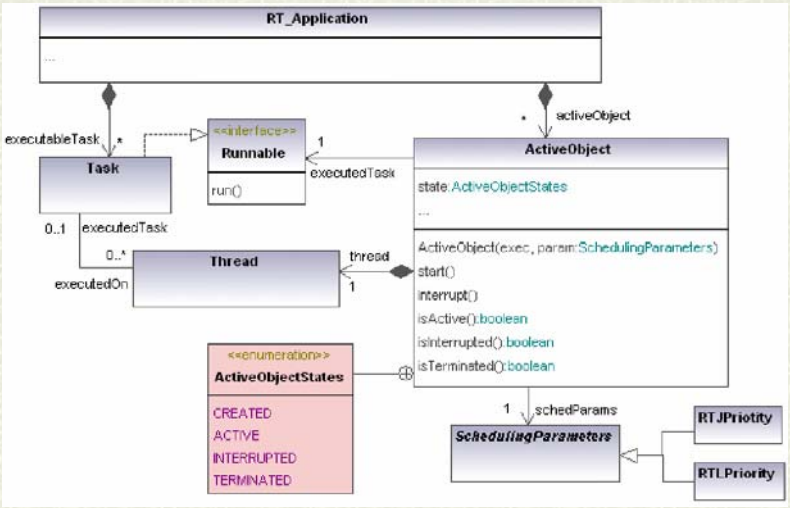
Patrones de tiempo real

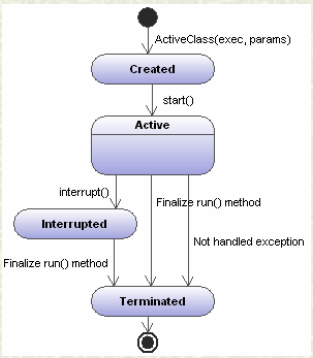
- **Patrones de control de mecanismos de ejecución concurrente**: patrones para la creación, asignación y liberación de trabajos, control de estado y finalización de *threads* y manejadores.
  - Patrón *ActiveObjet*
  - Patrón ExecutionTimeSever
  - Patrón ThreadPoolExecutor
- **Patrones de respuesta a eventos**: conjunto de mecanismos para la gestión de la atención y respuesta a eventos.
- **Patrones de sincronización**: mecanismos básicos a través de los que diferentes tareas que se ejecutan concurrentemente sincronizan sus estados y/o intercambian información de forma segura.
  - Patrón *ProtectedObject*
  - Patrón *Syncronizer*
- **Patrones de comunicación entre particiones**: Representan mecanismos de sincronización e intercambio de datos entre threads que se encuentran instanciados en diferentes nudos o particiones.
  - Patrón *Socket*
- **Patrones de tareas de tiempo real**: describen la estructura de diferentes tareas que se ejecutan con un patrón de disparo predefinido y con unos requisitos temporales determinados.
  - Patrón *PeriodicTask*
  - Patrón *SporadicTask*
  - Patrón *AperiodicTask*

Patrón “ActiveObject”

- **Propósito**: Cada objeto activo ejecuta concurrentemente una tarea que se le asigna, haciendo uso de un *thread* propio que se implementa con los recursos que le proporciona la plataforma de ejecución.
  - Las aplicaciones de tiempo real se diseñan como un conjunto de objetos activos que compiten por el acceso al procesador y a los recursos compartidos que requieren acceso en régimen de exclusión mutua.
  - El patrón formaliza la gestión de su ciclo de vida y define su prioridad para acceder al procesador y a los recursos protegidos.
- **Motivación y aplicabilidad**:
  - Cada objeto activo va a gestionar la ejecución de una tarea diferentes,por los que se puede planificar independientemente la ejecución de cada una de ellas.
  - Hace segura la ejecución concurrente de tareas, gestionando la suspensión o finalización de cada una de ellas, sólo cuando el estado de acceso a los mutexes sea el adecuado para que no afecten a otras tareas que siguen accediendo a ellos.
- Ejemplos de uso:
  - El muestreo periódico de una señal se puede construir con un un objeto activo, para que se ejecute en el instante previsto, con independencia de otras tareas que se estén ejecutando.
  - En una base de datos de tiempo real se consigue que a una actualización se le pueda limitar la latencia en el acceso a una tarea programando cada acceso en un objeto activo diferentes.

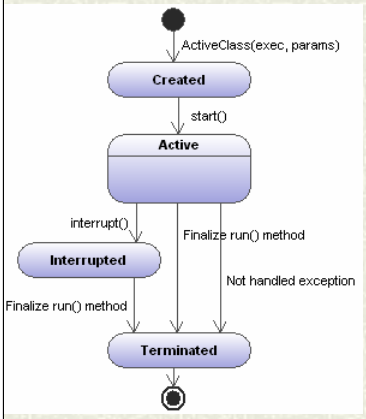
Patrón “ActiveObject”: Estructura

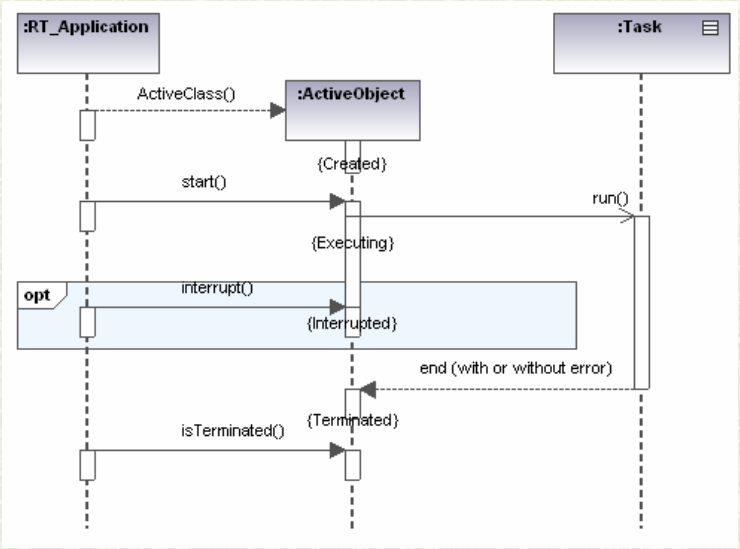




Santander, 2012 Métodos, procesos y entornos para sistemas de tiempo real J.M. Drake 11

Patrón “ActiveObject”: Ciclo de vida





Patrón ActiveObjet: Modelo de tempo real.

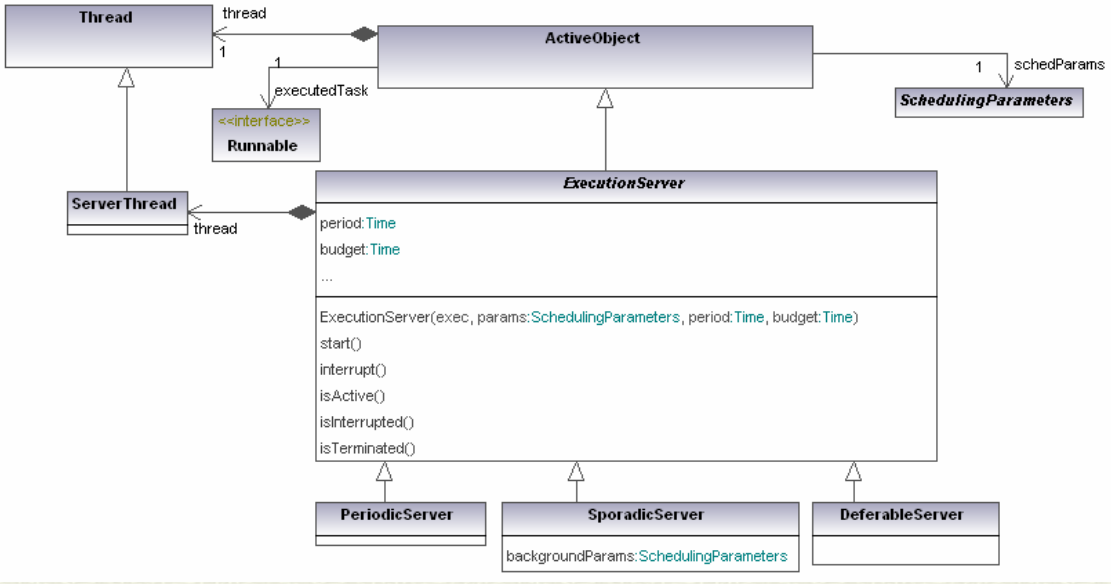
```
<!-- Modelo de la plataforma de ejecución -->
<mast_mdl:Regular_Processor Name="theProcessor"/>
<mast_mdl:Primary_Scheduler Name="theScheduler" Host=" theProcessor">
        <mast_mdl:Fixed_Priority_Policy Max_Priority="58" Min_Priority="11"/>
</mast_mdl:Primary_Scheduler>
<!-- Modelo del thread como elemento que se puede ser planificado  -->
<mast_mdl:Thread Name="theActiveObject" Scheduler=" theScheduler">
        <mast_mdl:Fixed_Priority_Params  Priority="20"/>
</mast_mdl:Thread>
<!-- Modelo de la tarea como una operación simple -->
<mast_mdl:Simple_Operation Name="theTask"
                        Worst_Case_Execution_Time="22E-3"
                        Avg_Case_Execution_Time="12.3E-3"
                        Best_Case_Execution_Time="1.42E-3"/>

<!-- Ejemplo de la transascción que describe la actividad del objeto activo -->
<mast_mdl:Regular_End_To_End_Flow Name="ActiveObjectActivity">
        <!-- El objeto activo ejecuta periodicamente la tarea-->
        <mast_mdl:Periodic_Event Name="Trigger" Period="1.33E-3"/>
        <mast_mdl:Internal_Event Name="End">
                <!-- Requisito temporal: deadline al final del periodo de activavión -->
                <mast_mdl:Hard_Global_Deadline Referenced_Event="Trigger" Deadline="1.33E-3"/>
        </mast_mdl:Internal_Event>
        <!-- Establece la actividad que se ejecuta -->
        <mast_mdl:Step Input_Event="Trigger" Output_Event="EndEvent"
                        Step_Schedulable_Resource="theActiveObject"
                        Step_Operation=" theTask"/>
</mast_mdl:Regular_End_To_End_Flow>
```

Patrón “ExecutionTimeSever”

- **Propósito:** Es un tipo de especializado de *ActiveObject*, que tiene una capacidad de procesamiento (*budget*) preestablecido.
  - Si la ejecución de la tarea consume el *budget*, la prioridad del *thread* se reduce a un nivel que no afecta a otros *threads* con requisitos de tiempo real.
  - El *budget* disponible para la ejecución se restituye de forma periódica.
- **Motivación y aplicaciones**:
  - Aplicaciones con tareas aperiódicas, en las que se controla la influencia que dichas tareas pueden tener en la planificación del sistema.
  - Aplicaciones que se instalan en plataformas abiertas en las que se conoce su carga total de trabajo, y por ello no se les puede aplicar las técnicas clásicas de diseño y análisis de tiempo real.

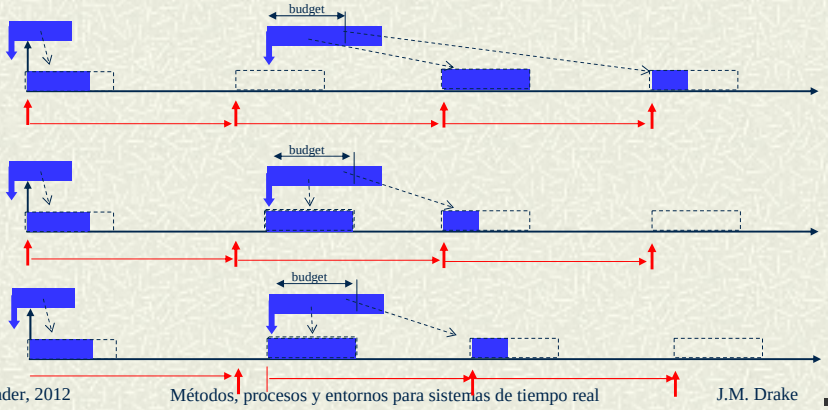
Patrón “ExecutionTimeSever” :Estructura

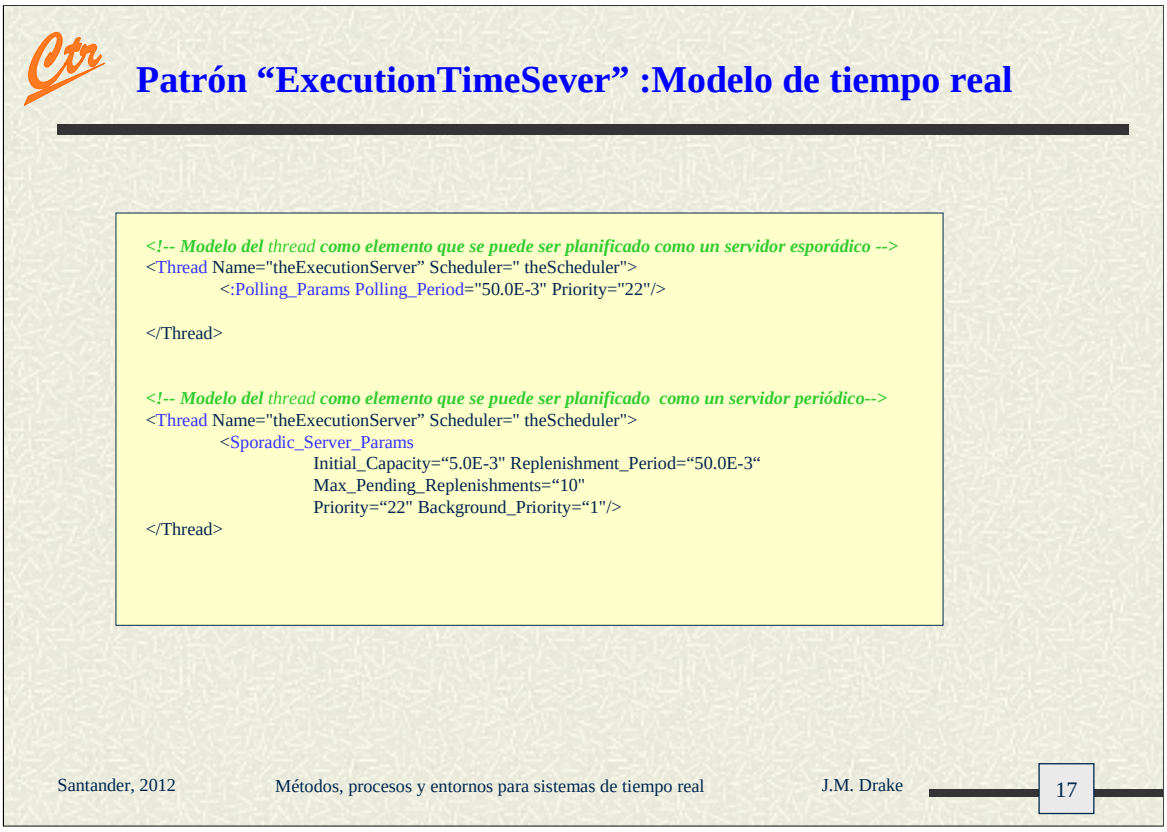


ServerThread

- Un servidor es una estructura con:
  - Un thread
  - Una estructura de datos
  - Dos timer: uno de proceso que mide el budget y otro absoluto que mide el periodo de restitución.

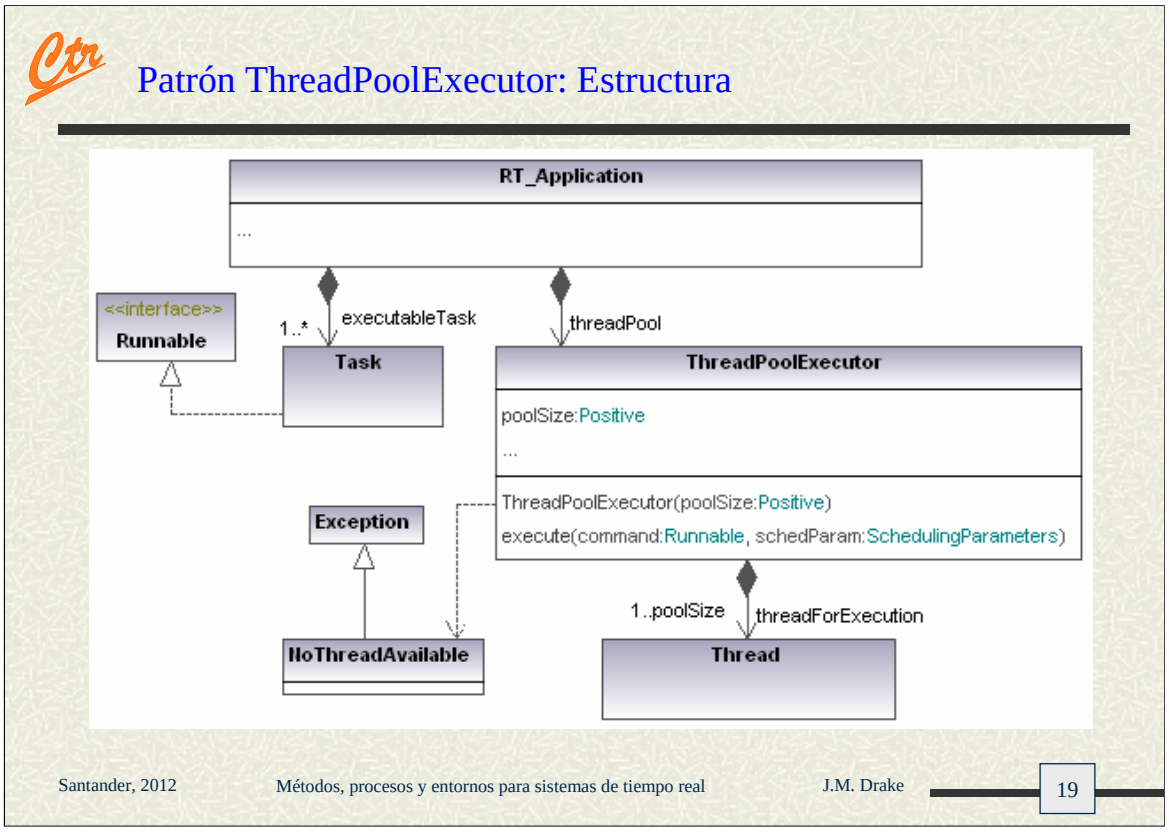
  que permite ejecutar una tarea con capacidad de procesamiento controlado.
- De acuerdo con la estrategia de reemplazo:

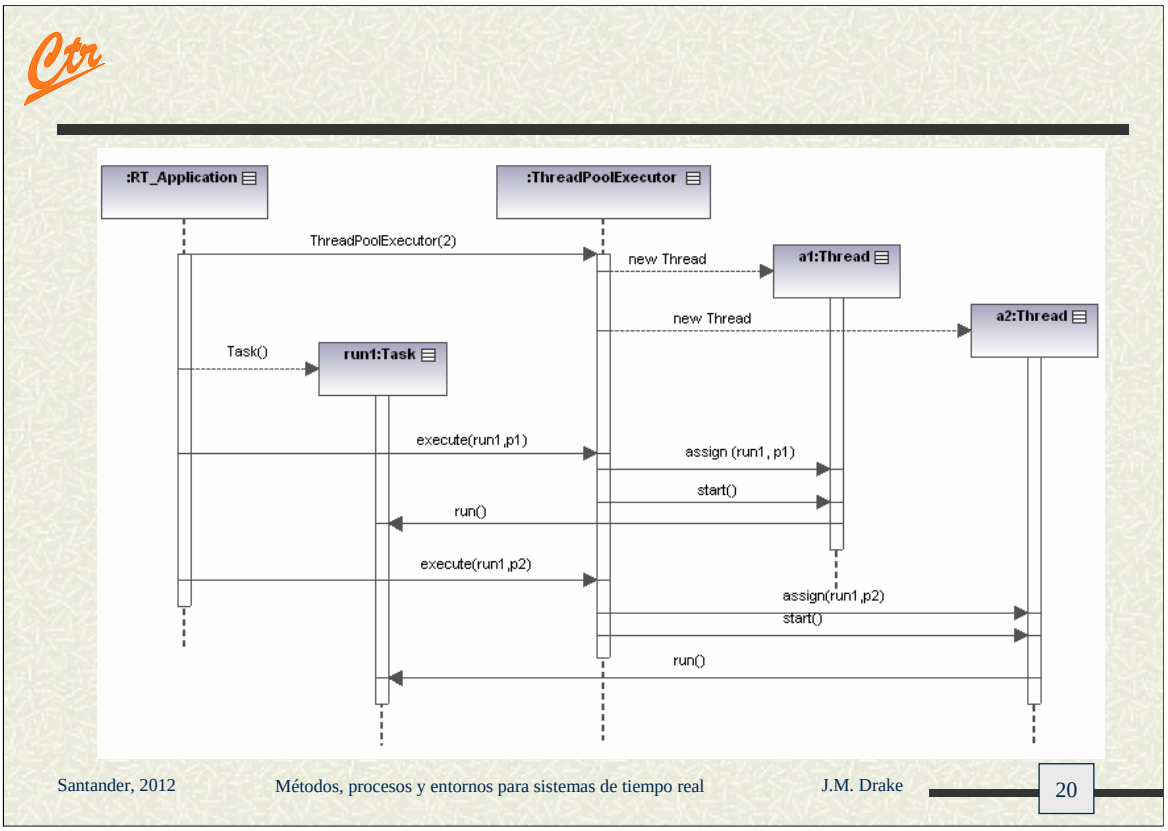



Patrón “ExecutionTimeSever” :Modelo de tiempo real
<!-- Modelo del thread como elemento que se puede ser planificado como un servidor esporádico -->
<Thread Name="theExecutionServer” Scheduler=" theScheduler">
<:Polling_Params Polling_Period="50.0E-3" Priority="22"/>
</Thread>
<!-- Modelo del thread como elemento que se puede ser planificado como un servidor periódico-->
<Thread Name="theExecutionServer” Scheduler=" theScheduler">
<Sporadic_Server_Params
Initial_Capacity=“5.0E-3" Replenishment_Period=“50.0E-3“
Max_Pending_Replenishments=“10"
Priority=“22" Background_Priority=“1"/>
</Thread>
Santander, 2012
Métodos, procesos y entornos para sistemas de tiempo real
J.M. Drake
17

Patrón ThreadPoolExecutor.

- **Propósito:** gestionar un número limitado de mecanismos de ejecución concurrente (*thread*) que son creados en la fase de inicialización del sistema y a los que se puede asignar dinámicamente la ejecución de diferentes tareas durante la fase de ejecución de tiempo real. En la fase de finalización del sistema se gestiona su terminación y destrucción. Las tareas que se puede asignar deben estar implementada en objetos que implementar la interfaz Runnable. El patrón proporciona medios para configurar los parámetros de planificación asociados a la ejecución de cada tarea de manera independiente.
- **Motivación**:El patrón se utiliza en sistemas en los que existen tareas concurrentes que se invocan de forma dinámica. Por ejemplo en una base de datos de tiempo real, cada tarea corresponde a los requerimientos que recibe, y estos se deben ejecutar con un *thread* propio de prioridad adecuada.
- Ventajas:
  - Eliminan las sobrecargas por la creación y destrucción dinámica de *threads*.
  - Se limita la cantidad de recursos que se utilizan en el sistema.
  - Se simpifica el mantenimiento de programas concurrentes a lo largo de su ciclo de vida, posibilitando el cambio de las políticas de ejecución sin cambiar el código de la propia ejecución.

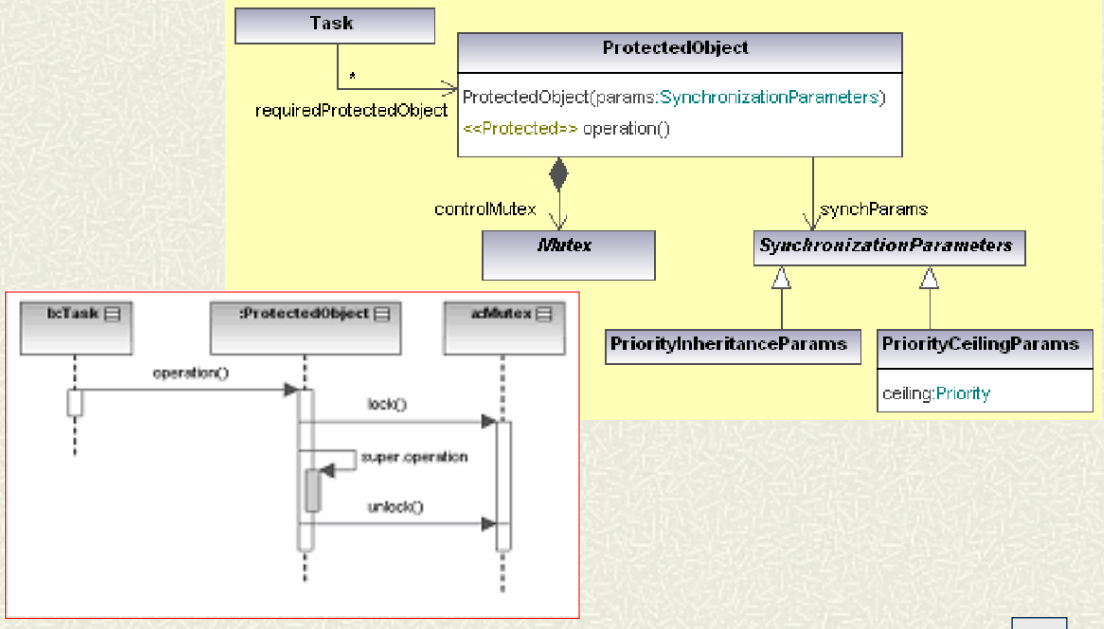

Patrón ThreadPoolExecutor: Estructura
RT_Application
...
<<interface>>
Runnable
1..*
executableTask
threadPool
Task
ThreadPoolExecutor
poolSize:Positive
...
ThreadPoolExecutor(poolSize:Positive)
execute(command:Runnable, schedParam:SchedulingParameters)
Exception
1..poolSize
threadForExecution
NoThreadAvailable
Thread
Santander, 2012
Métodos, procesos y entornos para sistemas de tiempo real
J.M. Drake
19


:RT_Application
:ThreadPoolExecutor
ThreadPoolExecutor(2)
new Thread
a1:Thread
new Thread
a2:Thread
Task()
run1:Task
execute(run1,p1)
assign (run1, p1)
start()
run()
execute(run1,p2)
assign(run1,p2)
start()
run()
Santander, 2012
Métodos, procesos y entornos para sistemas de tiempo real
J.M. Drake
20

Patrón ProtectedObject

- **Propósito**: Garantizar el acceso en régimen de exclusión mutua. al estado y a los atributos de un objeto por parte de un conjunto de tareas que acceden a él de forma concurrente,
- **Motivación**: Se introduce un único mecanismo de sincronización por objeto de la clase, y es el propio compilador el que introduce automáticamente la toma y liberación del mutex del objeto en cada método de su interfaz. .
  - Con su uso se evita el acceso directo desde el código de los semáforos y mutexes que son elementos de muy bajo nivel de abstracción. Esto hace la programación mucho mas segura.

Patrón ProtectedObject:Estructura



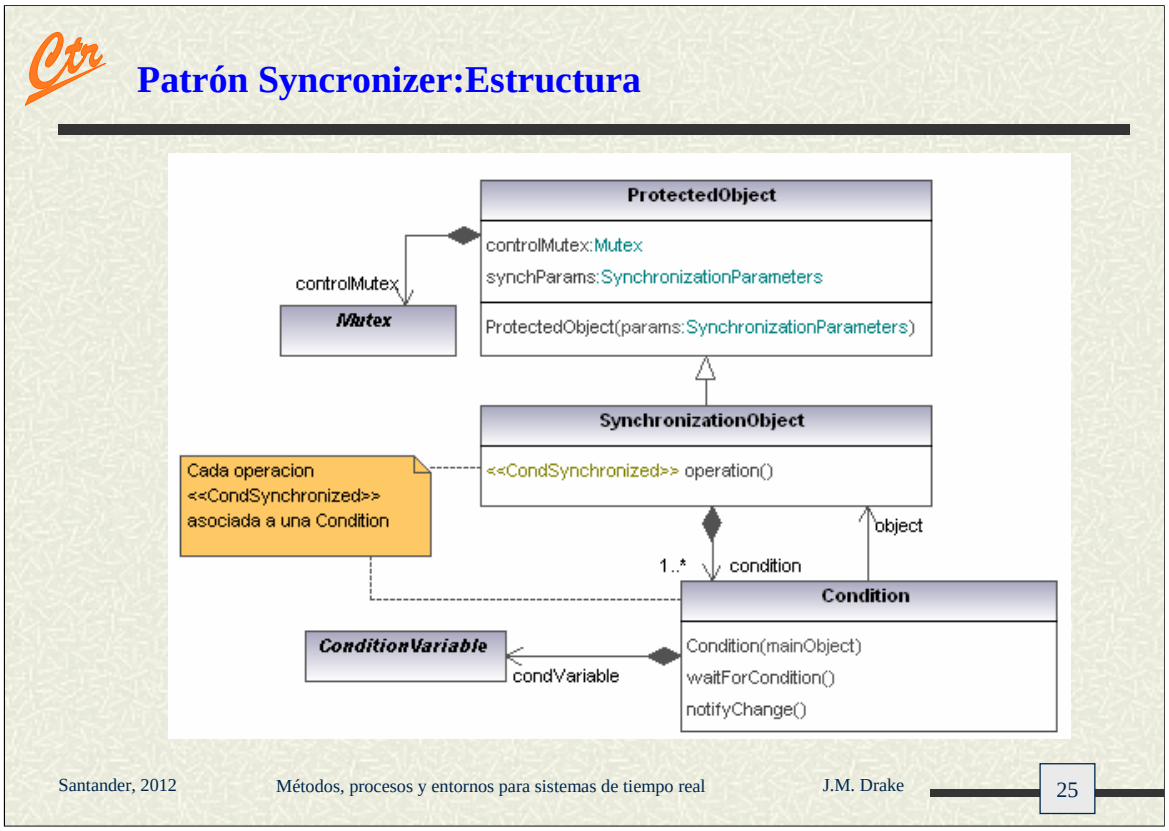
Patrón ProtectedObject:Modelo de tiempo real

```
<!-- Modelo del mutex asociado al objeto protegido  -->
<Inmediate_Ceiling _Resource Name="theMutex" Ceiling="58"/>

<!-- Modelo de las operaciones del objeto operación simples -->
<Simple_Operation Name="operation" Worst_Case_Execution_Time="22E-3"
                  Average_Case_Execution_Time="12.3E-3"
Best_Case_Execution_Time="1.42E-3">
```

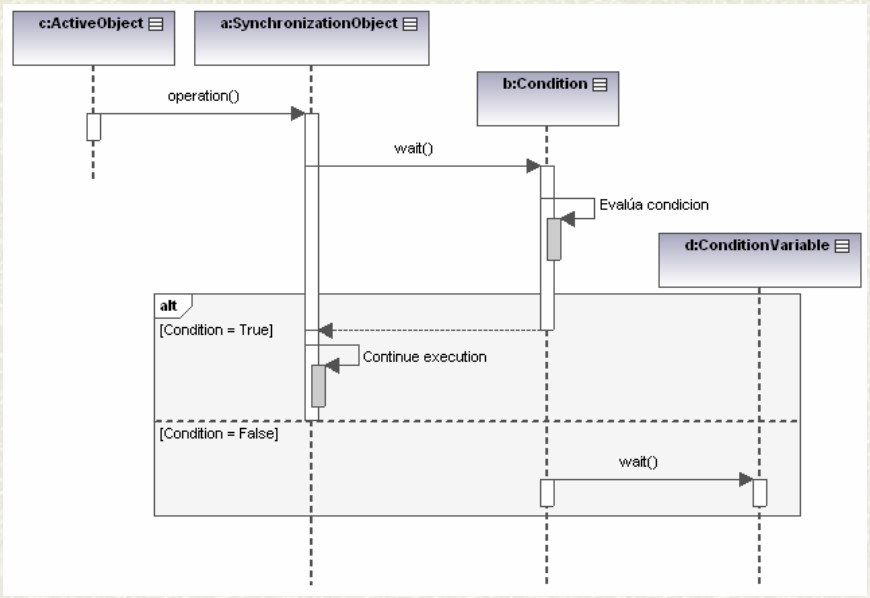
Patrón Syncronizer

- **Propósito:** Un objeto *synchronizer* incluye un mecanismo de sincronización en el que los threads que invocan sus métodos pueden quedar suspendidos a la espera de que el objeto alcance un determinado estado interno.
- **Motivación**: El patrón ofrece un mecanismo que permite la sincronización de los flujos de ejecución de diferentes tareas, esto es, una tarea se suspende hasta que otra tarea alcanza un determinado estado o hasta que se alcanza cierto estado en el sistema. La primera tarea se suspende en espera de que la segunda le comunique cuando se ha alcanzado el estado deseado. Cuando dicho estado se alcanza, la tarea suspendida reinicia su ejecución.



Patrón Syncronizer:Estructura

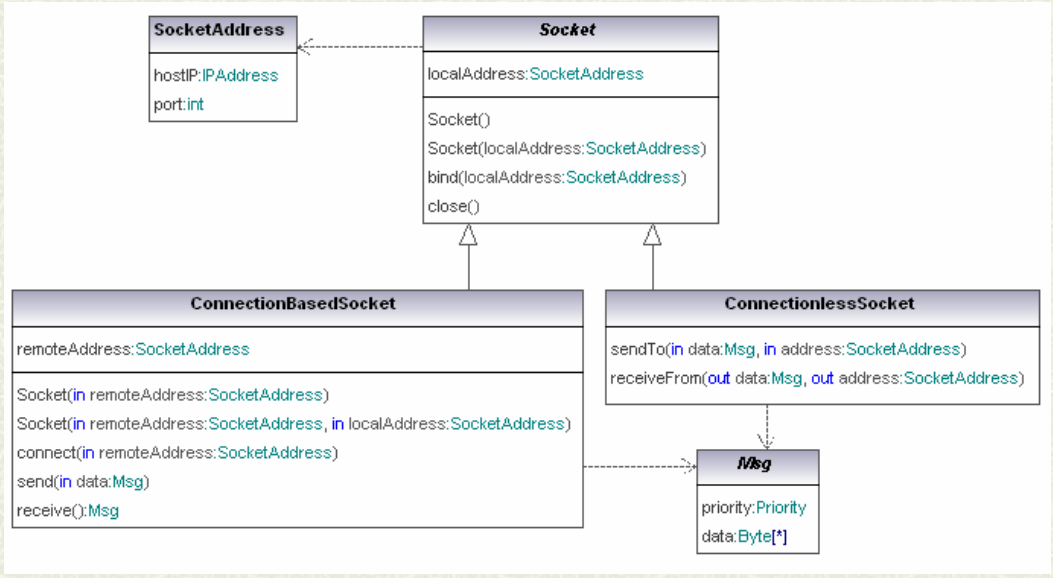
Patrón Syncronizer:Estructura



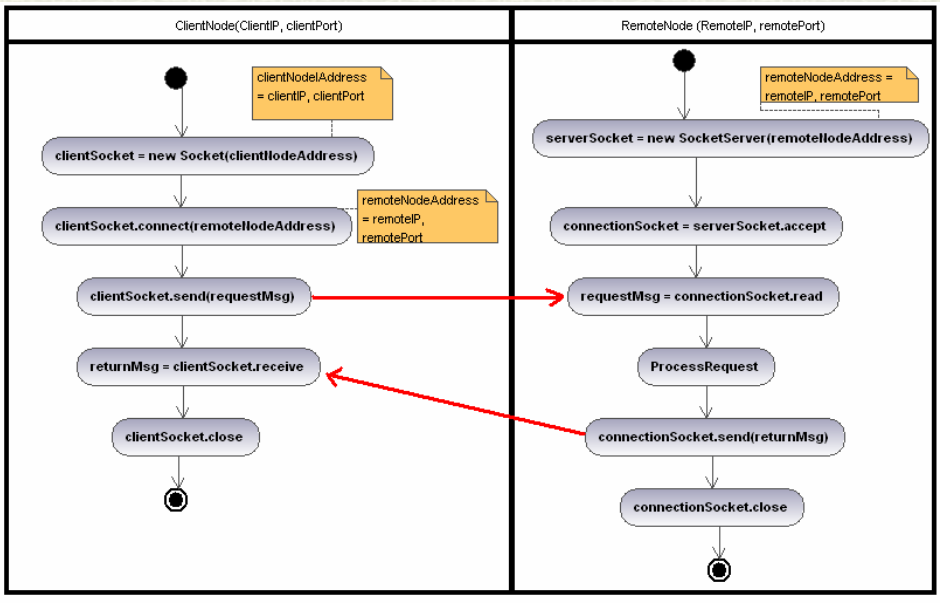
Patrón Socket

- **Propósito**: conseguir un mecanismo de sincronización y comunicación entre *threads* que se encuentran en distintos nodos. Cada *socket* representa un extremo de un canal de comunicación entre dos nodos, que puede ser utilizado por las tareas para enviar datos o sincronizarse con otras tareas de nodos remotos.
- **Motivación**: En sistemas distribuidos, es necesario contar con algún tipo de mecanismo que permita implementar la sincronización de espera cuando los threads que se comunican se encuentran en diferentes nodos o espacios de memoria. Con el uso de *sockets* podemos conseguir que un *thread* en el nodo cliente se quede a la espera de que se realice algún tipo de actividad en el nodo remoto, o viceversa.
  - Los *sockets* proporcionan una abstracción de los protocolos de transporte que se utilizan para implementar la comunicación entre los procesadores, facilitando así la labor de los programadores, que no tienen por qué conocer detalles acerca del modo de gestionar los envíos de datos a través de la red.

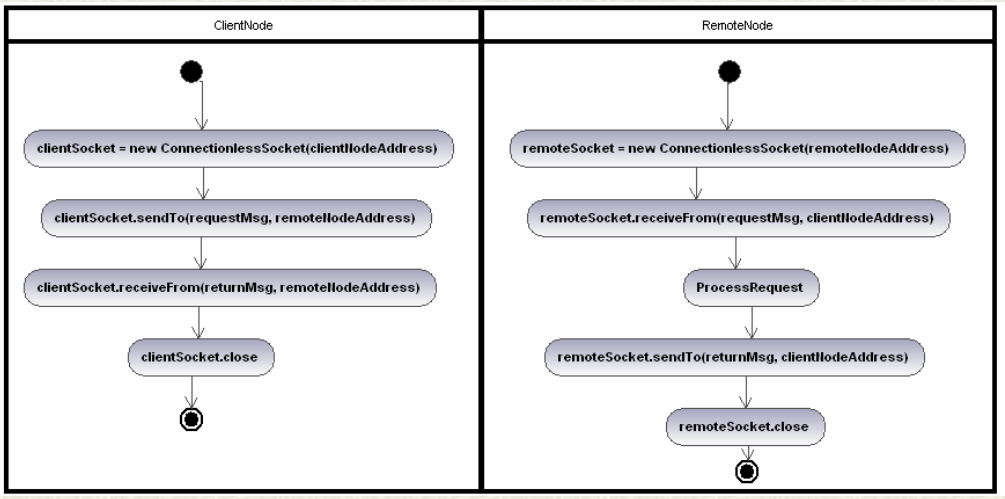
Patrón Socket:Estructura



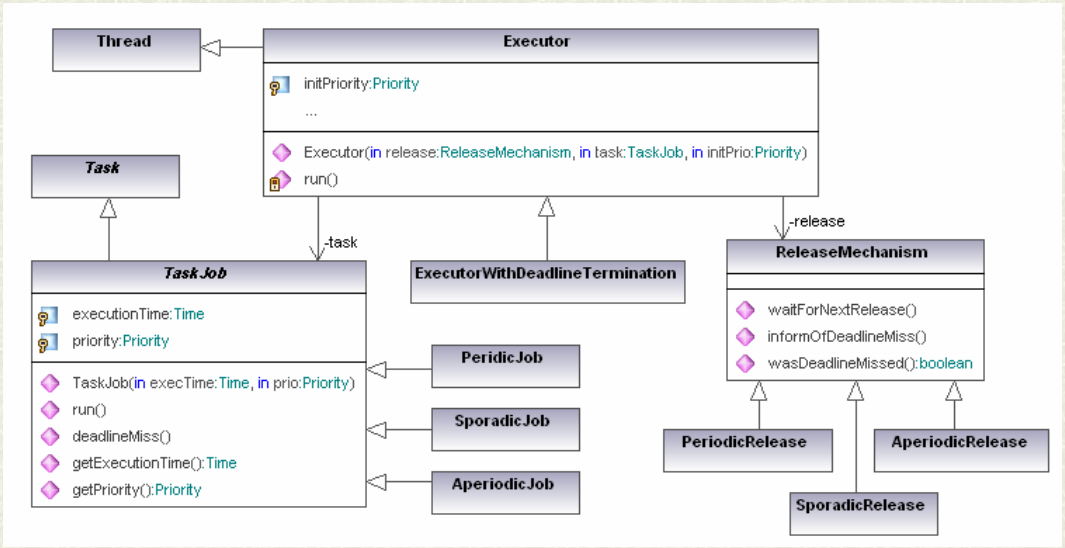
Sockets: Interacciones en un socket orientado a la conexión



Sockets: Interacciones en un socket no orientado a la conexión



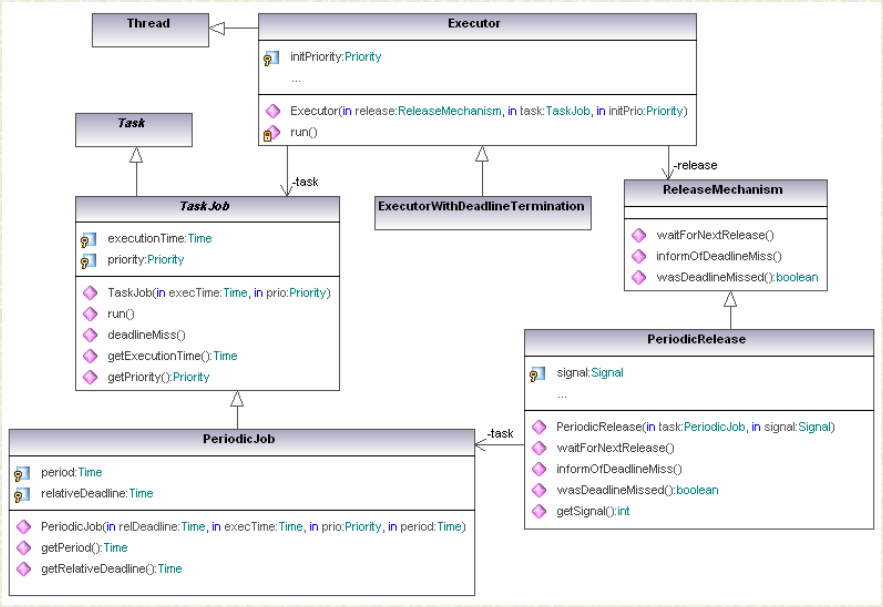
Gestión global de tareas: Elementos comunes

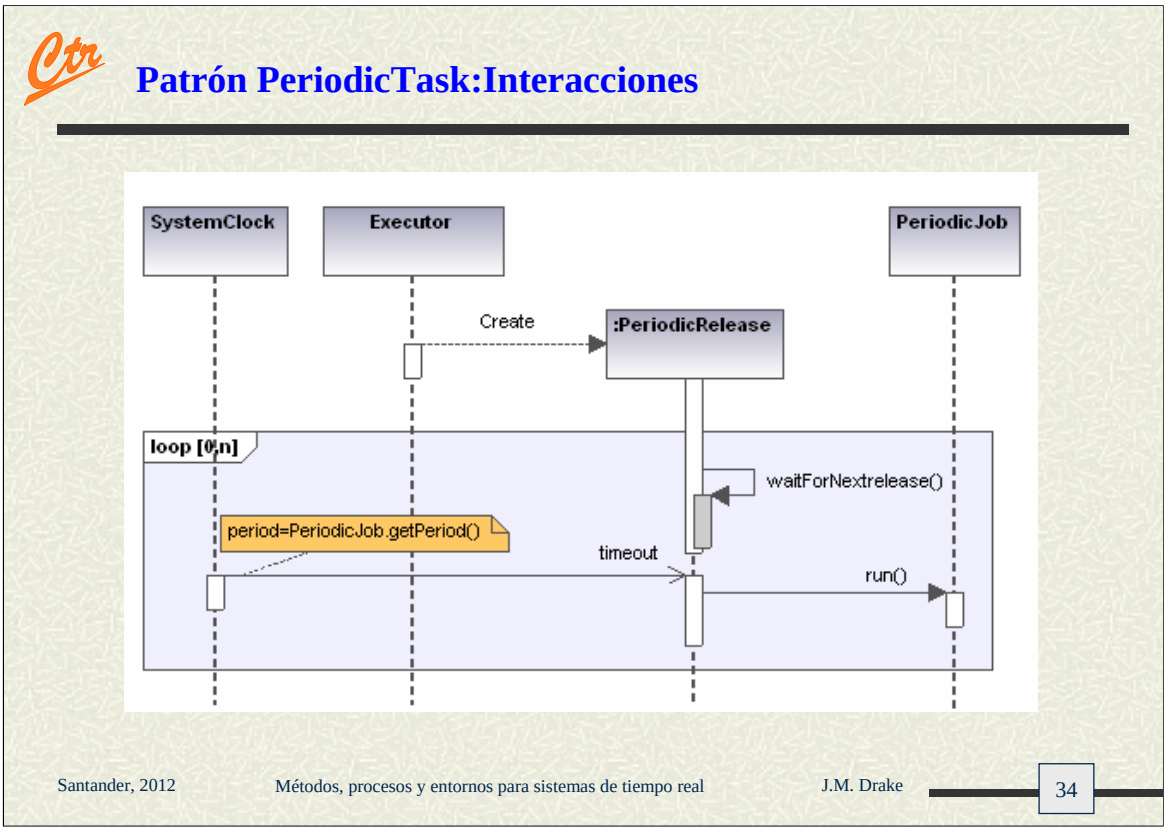


Patrón PeriodicTask

- **Propósito**: ejecuta periódicamente un trabajo con requisitos temporales simples o múltiples. En nuestro caso consideraremos el requisito temporal simple consistente en un plazo máximo desde que se inicia el periodo hasta que finaliza la ejecución del trabajo. El patrón proporciona mecanismos para detectar incumplimientos de este plazo.
- **Motivación y aplicabilidad**: prácticamente todas las aplicaciones de tiempo real tienen bucles de control o de monitorización que es necesario repetir de manera periódica. Es habitual que estas tareas tengan un plazo de finalización para cada una de sus activaciones, que se cuenta desde el inicio del periodo. En muchas ocasiones el plazo coincide con el final del periodo, lo que indica que cada instancia de la tarea debe finalizar antes de que comience la siguiente. En otras ocasiones el plazo es menor al periodo, lo que indica que la actividad a realizar es muy urgente, o requiere una finalización con una variabilidad (*jitter*) pequeña. También hay casos en los que el plazo puede ser mayor al periodo, aunque son menos habituales.
- **Aplicaciones típicas**: Los sistemas de control o de monitorización de señales.

Patrón PeriodicTask:Estructura



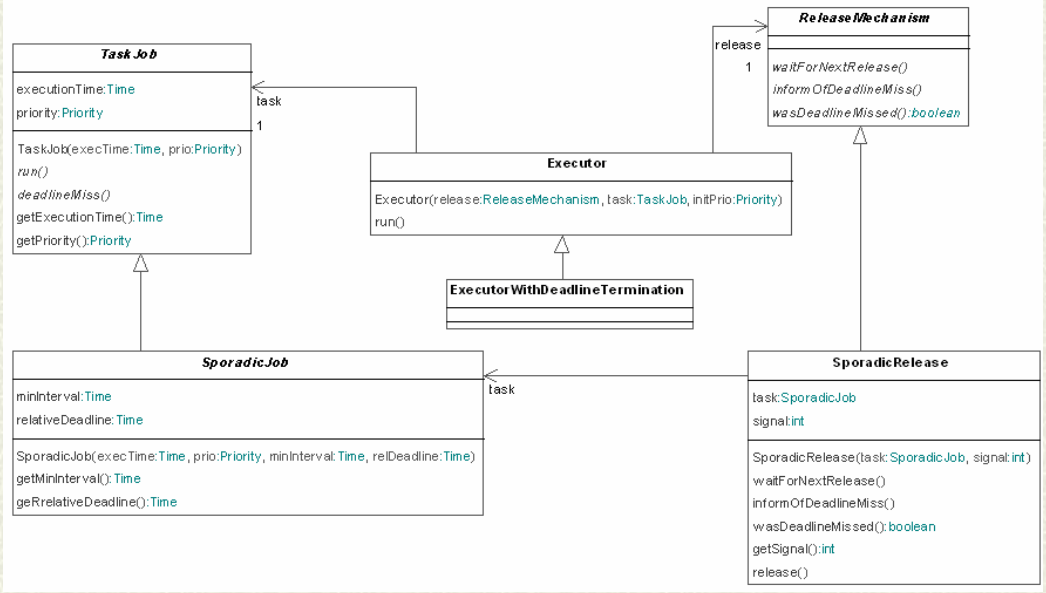


Patrón PeriodicTask:Interacciones

Patrón SporadicTask

- **Propósito**: ejecuta un trabajo con un patrón esporádico, en el que el periodo medio y el mínimo está especificado. Los trabajos pueden tener requisitos temporales simples o múltiples.
- **Motivación**: ejecuta repetidamente un trabajo en respuesta a la llegada de un evento externo, cuyo intervalo mínimo entre llegadas está especificado. El trabajo puede tener un plazo de finalización que se inicia con la llegada del evento y termina con la finalización de la ejecución del trabajo. El patrón proporciona mecanismos para detectar incumplimientos de este plazo. Así mismo proporciona los mecanismos para proteger al sistema de llegadas de eventos demasiado próximas, garantizando así la ejecución con una separación suficiente y, por tanto, con un gasto máximo de CPU igual al de procesar un evento a cada intervalo igual al tiempo mínimo entre llegadas. Se suele aplicar en subsistemas de control o de monitorización de señales.

Patrón SporadicTask:Estructura

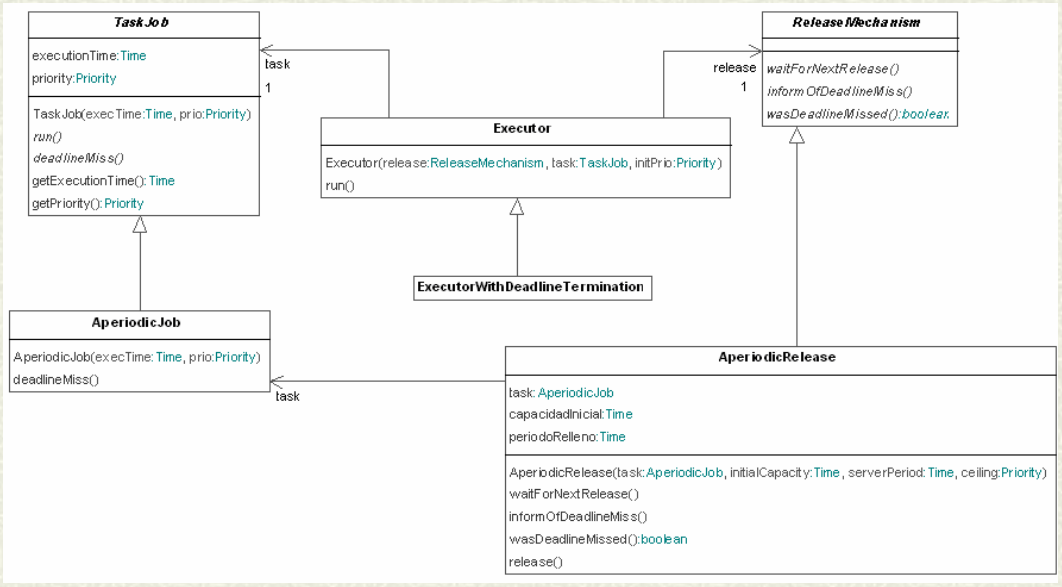


Patrón AperiodicTask

- **Propósito**: ejecuta un trabajo que se solicita a intervalos irregulares y sin requisitos temporales. Las tareas son ejecutadas bajo un servidor de tiempo de ejecución limitado, de modo que su impacto sobre tareas de prioridad inferior se pueda limitar.

  Este patrón constituye la manera más básica de ejecutar una tarea aperiódica que ejecuta una instancia cada vez que llega un evento del exterior, pero limitando la anchura de banda dedicada a la tarea aperiódica, de modo que otras tareas de prioridad inferior puedan ejecutar cumpliendo sus requisitos temporales.

- **Motivación**: prácticamente todas las aplicaciones de tiempo real tienen necesidad de responder a estímulos repetitivos que llegan del exterior, quizás a través de una interrupción hardware o de un mensaje que llega por una red, siendo los intervalos entre las llegadas de estos estímulos irregulares. Estas tareas no pueden tener un plazo de finalización para sí mismas, dada la irregularidad de sus ritmos de activación, pero es necesario limitar la cantidad de recursos o anchura de banda que consumen, para garantizar que otras tareas de prioridad inferior tienen suficientes recursos para ejecutarse cumpliendo sus requisitos temporales.
- **Aplicaciones típicas**: Subsistemas de control o de monitorización de señales en los que hay respuestas a eventos externos, por ejemplo provenientes de un operador o de sucesos de temporización irregular.


TaskJob
executionTime:Time
priority:Priority
TaskJob(execTime:Time, prio:Priority)
run()
deadlineMiss()
getExecutionTime():Time
getPriority():Priority
task
1
ReleaseMechanism
waitForNextRelease()
informOfDeadlineMiss()
wasDeadlineMissed():boolean
release
1
Executor
Executor(release:ReleaseMechanism, task:TaskJob, initPrio:Priority)
run()
ExecutorWithDeadlineTermination
AperiodicJob
AperiodicJob(execTime:Time, prio:Priority)
deadlineMiss()
task
AperiodicRelease
task:AperiodicJob
capacidadInicial:Time
periodoRelleno:Time
AperiodicRelease(task:AperiodicJob, initialCapacity:Time, serverPeriod:Time, ceiling:Priority)
waitForNextRelease()
informOfDeadlineMiss()
wasDeadlineMissed():boolean
release()