Programa Oficial de Postgrado en Ciencias, Tecnología y Computación
Máster en Computación
Facultad de Ciencias - Universidad de Cantabria

Servicio de gestión de tiempos para sistemas distribuidos sobre el middleware ICE

Ángela del Barrio Fernández
delbarrioa@unican.es

Director:
José María Drake Moyano.
Grupo de Computadores y Tiempo Real
Departamento de Electrónica y Computadores.

Santander, octubre de 2008
Curso 2007/2008

Agradecimientos

En primer lugar quiero expresar mi más sincero agradecimiento a mi director de proyecto el Dr. José María Drake Moyano, catedrático del Departamento de Electrónica y Computadores de la Universidad de Cantabria, por su confianza, amabilidad, dedicación, minuciosidad y su disposición en todo momento para resolver cualquier duda surgida durante el desarrollo del proyecto.

Gracias a mis compañeros del Grupo de Computadores y Tiempo Real, por el buen ambiente de trabajo y sentimiento de grupo del que disfruto cada dia y con quienes casi a diario aprendo cosas nuevas.

Por último desearía expresar mi más profundo agradecimiento a mi familia por su apoyo y dedicación.

Índice de contenido

Índice de tablas

1.Ámbito y objetivos del trabajo

1.1 Aleatoriedad del tiempo en los sistemas distribuidos

En los sistemas de tiempo real se necesita disponer de un reloj global con el que caracterizar temporalmente los eventos que se reciben del entorno y las acciones que se realizan sobre él [1]. Así mismo, en los sistemas concurrentes, en los que las acciones que se ejecutan en ellas no tienen definido un orden estricto de ejecución, se puede necesitar un reloj global si se necesita comparar sus acciones por su precedencia temporal. Así, por ejemplo, en una red eléctrica que esta siendo monitorizada, cuando se produce un fallo en uno de sus nodos, suele provocar fallos en cadena en otros muchos otros nodos. Si el sistema debe identificar el origen de los fallos, debe ser capaz de ordenar los eventos con resolución de milisegundos para identificar cual de ellos fue el primero que se produjo.

En un sistema monoprocesador es sencillo implementar un reloj global y monótono, ya que cuando un proceso quiere obtener el tiempo actual, realiza una llamada al núcleo del sistema, que le devuelve el dato leyendo el único reloj existente en él. Si un proceso A pregunta el tiempo y un momento después un proceso B también lo hace, el valor obtenido por B será mayor o igual al obtenido por A, pero en ningún caso, en condiciones normales de funcionamiento será menor. En un sistema distribuido constituido por varios procesadores, es mucho más difícil disponer de relojes que proporcionen un tiempo global y no ambiguo.

Los relojes de los computadores se basan en dispositivos hardware de tipo temporizador (*timers)*. Un timer de un ordenador habitualmente consiste en un cristal de cuarzo que oscila de forma precisa. Asociado al mismo existen dos elementos: un contador (*counter)* y un registro (*holding)*. Cada una de las oscilaciones del cristal decrementa el contador en una unidad. Cuando la cuenta se hace cero, se genera una interrupción y el contador se vuelve a cargar a partir del registro *holding*. De esta forma es posible programar un temporizador que genere interrupciones a una frecuencia determinada. Cada una de estas interrupciones se llama *(tick) del reloj*.

Cuando se arranca por primera vez el sistema, generalmente se pregunta al usuario la hora y la fecha. Este dato se convierte en un número de ticks referenciados a una fecha inicial (*epoch*) y se almacena en la memoria.

Si hay un solo ordenador que utiliza un único reloj, no tiene mucha importancia la exactitud del mismo, ya que todos los eventos se van a regir por el mismo reloj e internamente el sistema será consistente, figura 1.1.

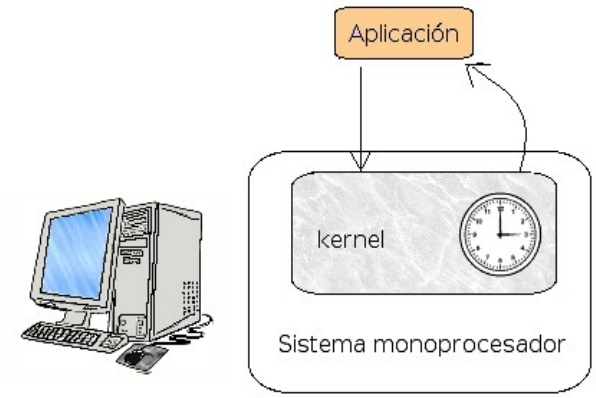

Figura 1.1: Sistema monoprocesador

Cuando el sistema es distribuido y está compuesto por varios computadores, cada uno con su propio reloj, la situación cambia. A pesar de que la frecuencia a la que oscila el cristal de cuarzo sea estable, es imposible garantizar que los cristales de diferentes ordenadores corran exactamente a la misma frecuencia. En la practica, cuando un sistema está formado por *n* ordenadores, los *n* cristales correrán con tasas

ligeramente diferentes causando que los relojes software se desincronicen gradualmente, figura 1.2. Esta diferencia se denomina sesgo del reloj (*clock skew)*. Como consecuencia del sesgo, eventos caracterizados temporalmente por diferentes relojes, no pueden ser comparados entre sí. En estos casos es necesario que los relojes se encuentren sincronizados entre si, esto es, que exista un proceso continuado de comparación y ajuste que garantice que la frecuencia media de todos los relojes sea exactamente la misma [2].

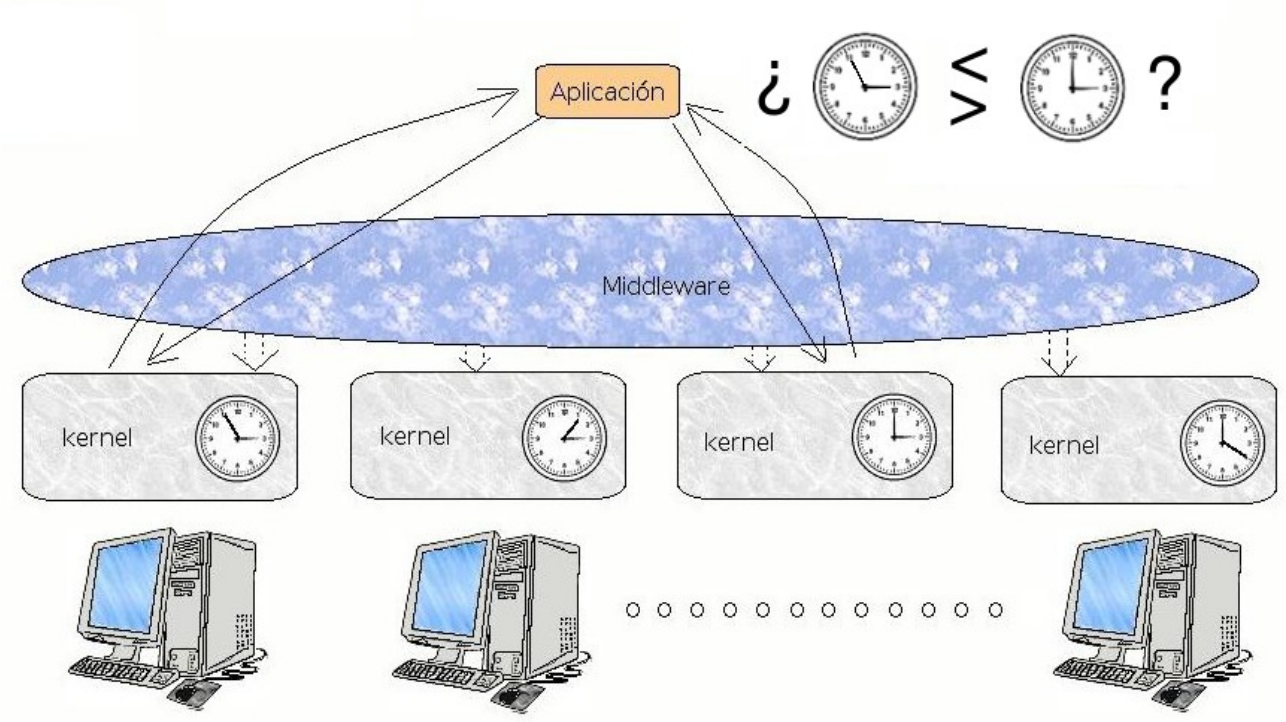

Figura 1.2: Plataforma distribuida

1.2 Formatos y gestión del tiempo en los sistemas informáticos

Para proporcionar el tiempo de forma precisa, el *National Institute of Standards and Technology* (NIST) rige una estación de radio de onda corta llamada WWV. La difusión *broadcast* de WWV emite un pulso corto al inicio de cada segundo UTC[1], La precisión de WWV es aproximadamente de ± 1 msg, pero debido a fluctuaciones atmosféricas aleatorias en la práctica, la precisión no es mayor de ±10msg. Existen varios satélites que también ofrecen el servicio UTC con altas precisiones. La utilización de cualquiera de estos sistemas, ya sea radio o satélite requiere un conocimiento preciso de la posición relativa del transmisor y del receptor para compensar el retraso de propagación de la señal.

Si una máquina tiene un receptor WWV, el objetivo consiste en conseguir que las demás máquinas se sincronicen con esta. Si ninguna máquina tiene un receptor WWV, cada máquina computa su propio tiempo, y el objetivo es mantener todas las máquinas con un tiempo lo más próximo posible las unas de las otras. Existen múltiples algoritmos para conseguir este propósito (Cristian, Drummond y Babaoglu, Kopetz y Oschsenreiter) [3]. Todos estos algoritmos se basan en el mismo modelo de sistema subyacente, donde se asume que un timer provoca una interrupción *H* veces por segundo. Cuando el tiempo se acaba, el manejador de la interrupción añade 1 al reloj software que lleva la cuenta del número de ticks o interrupciones desde un momento determinado del pasado, *C*. De manera más específica, cuando un tiempo UTC es *t*, el valor del reloj de la máquina local *p*, es *Cp(t),* figura *1.3*. En un sistema cuyo reloj funcionase de manera perfecta *Cp(t)=t* para todo *p* y *t*. En otras palabras, *dC/dt =1*.

1 Universal Coordinated Time. Escala de tiempo atómica derivada de *International Atomic Time* (TAI)

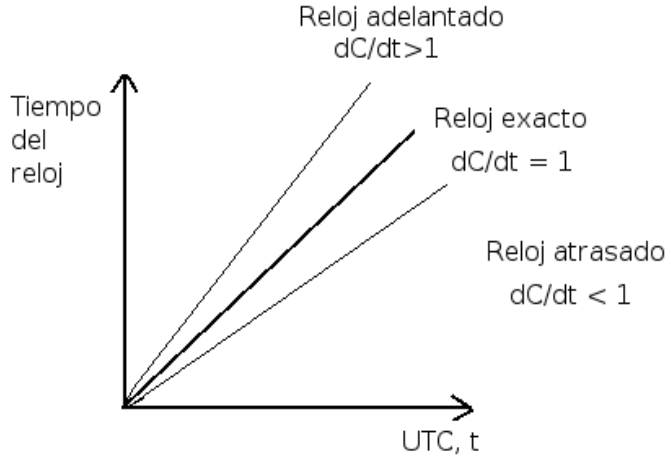

Figura 1.3: Drift de relojes

Los timers reales no interrumpen exactamente *H* veces por segundo al sistema. De forma teórica, un timer con *H= 60* debería generar 216.000 interrupciones cada hora. En la práctica, el error relativo que se obtiene con chips de tiempo modernos es del orden de 10E-5, lo que significa que una máquina en concreto podría oscilar en el rango de 215.998 a 216.003 ticks por hora. De forma más precisa, existe una constante ρ de forma que

$$1-\rho \leq \frac{dC}{dt} \leq 1+\rho$$

Fórmula 1.1

se puede asegurar que el timer está funcionando dentro de su especificación. La constante ρ viene especificada por el fabricante y se conoce con el nombre de ***maximum drift rate*** (tasa máxima de dispersión).

Si dos relojes derivan de la hora UTC en sentidos opuestos, en un momento Δt después de que hayan sido sincronizados, como mucho podrán distar 2ρΔt.

1.3 Plataforma distribuida de referencia

La plataforma en que se despliega y ejecuta la aplicación es distribuida y tiene procesadores distribuidos por toda el área de trabajo del sistema.

El middleware utilizado ha sido ICE[2] sobre la plataforma Linux Ubuntu 6.06, pero funciona en entornos heterogéneos y con aplicaciones desarrolladas en diferentes lenguajes de programación. El servidor de tiempos se ha desarrollado utilizando el lenguaje de programación C/C++.

En el sistema operativo se ha aplicado un parche de alta resolución de reloj [4] [5] . En la figura 1.4 se muestra la plataforma distribuida de referencia que se ha utilizado para implementar el servicio de tiempos [6]. Los ordenadores unidos mediante una línea discontinua representan sistemas susceptibles de incorporarse a dicho servicio.

2 Internet Communications Engine

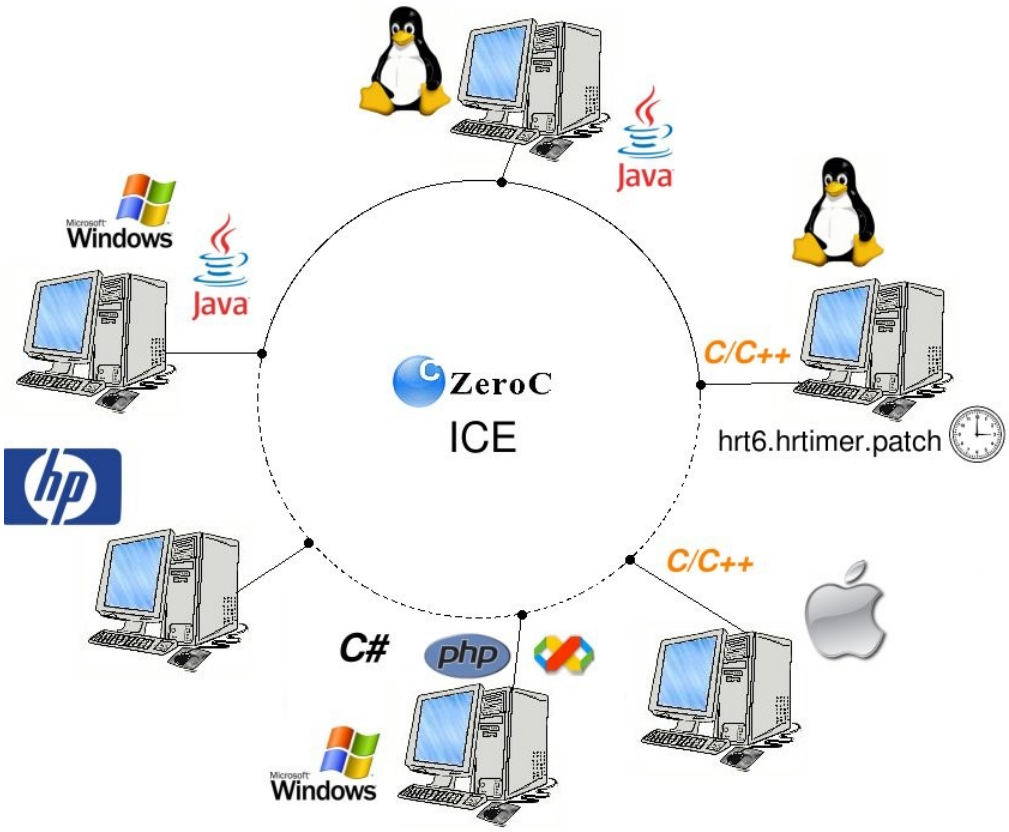

Figura 1.4: Plataforma distribuida de referencia

1.4 El middleware ICE

ICE (Internet Communication Engine) es un *middleware* orientado a objetos [7] . ICE proporciona herramientas, APIs, y soporte de bibliotecas para construir aplicaciones cliente-servidor orientadas a objetos. Una aplicación ICE se puede usar en entornos heterogéneos: los clientes y los servidores pueden escribirse en diferentes lenguajes de programación, pueden ejecutarse en distintos sistemas operativos y en distintas arquitecturas y pueden comunicarse empleando diferentes tecnologías de red. Además, el código fuente de estas aplicaciones puede portarse de manera independiente al entorno de desarrollo. Las principales características de Ice son las siguientes:

- Proporciona un *middleware* apto para adaptarse a sistemas heterogéneos
- Provee un conjunto de características que soportan el desarrollo de aplicaciones distribuidas reales en un amplio rango de dominios.
- Tiene licencia GPL
- Existe una versión para sistemas embebidos Ice-e
- La organización ZeroC garantiza el funcionamiento de Ice en las siguientes plataformas [8] :

Ice para C++

Plataformas	Compiladores
Windows XP SP2 (x86)	Visual Studio 2005 SP1 Visual C++ 2005 Express Edition Visual Studio 2008 Visual C++ 2008 Express Edition Visual C++ 6.0 SP5 and STLport 4.6.2 o superior CodeGear C++Builder 2007
Windows Vista (x86 y x64)	Visual Studio 2005 SP1 Visual C++ 2005 Express Edition (x86 solamente) Visual Studio 2008 Visual C++ 2008 Express Edition (x86 solamente) CodeGear C++Builder 2007 (x86 solamente)
Windows Server 2003 Standard (x86 y x64)	Visual Studio 2005 SP1 Visual C++ 2005 Express Edition (x86 solamente) Visual Studio 2008 Visual C++ 2008 Express Edition (x86 solamente)
Windows Server 2008 Standard (x86 y x64)	Visual Studio 2005 SP1 Visual C++ 2005 Express Edition (x86 solamente) Visual Studio 2008 Visual C++ 2008 Express Edition (x86 solamente)
Red Hat Enterprise Linux 4.6 (i386 and x86_64)	GCC 3.4.6
Red Hat Enterprise Linux 5.1 (i386 and x86_64)	GCC 4.1.2
SuSE Linux Enterprise Server 10 SP1 (i586 y x86_64)	GCC 4.1.2
MacOS X Leopard (Intel x86)	GCC 4.0.1
Solaris 10 (SPARC y x86/x64)	Sun Studio 12 (CC 5.9), 32/64 bit
HP-UX 11.11 (PA-RISC)†	HP aCC 3.56, 32/64 bit

Ice para Java

Plataformas	Compiladores
Todas las plataformas soportadas excepto HP-UX	JDK 1.5.0 or 1.6.0

Ice para .NET

Plataformas	Compiladores
Windows XP SP2 (x86)	Visual Studio 2005 (.NET 2.0) Visual Studio 2008 (.NET 3.5)
Windows Vista (x86 and x64)	Visual Studio 2005 (.NET 2.0) Visual Studio 2008 (.NET 3.5)
Windows Server 2003 Standard (x86 y x64)	Visual Studio 2005 (.NET 2.0) Visual Studio 2008 (.NET 3.5)
Windows Server 2008 Standard (x86 y x64)	Visual Studio 2005 (.NET 2.0) Visual Studio 2008 (.NET 3.5)
SuSE Linux Enterprise Server 10 SP1 (i586 y x86_64)	Mono 1.2.2

Ice para PHP

Plataformas	Compiladores
Windows XP SP2 (x86)	PHP 5.2.6
Red Hat Enterprise Linux 4.6 (i386 and x86_64)	PHP 5.1.6
Red Hat Enterprise Linux 5.1 (i386 and x86_64)	PHP 5.1.6
SuSE Linux Enterprise Server 10 SP1 (i586 y x86_64)	PHP 5.1.2

Ice para Python

Plataformas	Compiladores
Todas las plataformas Ice para C++ excepto HP-UX	Python 2.3, 2.4 or 2.5

Ice para Ruby

Plataformas	Compiladores
Windows XP SP2 (x86)	Ruby 1.8.6
Red Hat Enterprise Linux 4.6 (i386 and x86_64)	Ruby 1.8.1
Red Hat Enterprise Linux 5.1 (i386 and x86_64)	Ruby 1.8.5

Tabla 1.1: Plataformas soportadas por Ice 3.3

1.5 Objetivos y organización del trabajo

El presente trabajo ha sido el diseño e implementación de un servicio de tiempos (*Time Service*) que implemente fielmente la especificación *Enhanced Time Service* establecida por OMG [9] [10], adaptada para plataformas distribuidas y heterogéneas que operen sobre el *middleware Ice*.

Los aspectos que resuelve este servicio son:

- Gestiona la sincronización de todos los relojes de la plataforma de acuerdo con una estructura de dos niveles: Un nudo de la plataforma actúa como servidor de tiempo de todos lo demás nudos, y él se sincroniza con una servidor externo de bajo stratum.
- En caso de fallo del servidor de tiempo raíz, lo sustituye por otro de acuerdo con la información de configuración disponible en la plataforma.
- Permite obtener el tiempo UTC actual, convenientemente cualificado para que pueda ser comparado con otros instantes de tiempos obtenidos también del servicio.
- Permite comparar de forma segura diferentes instantes de tiempos obtenidos en los diferentes nudos de la plataforma.
- Permite generar relojes de misión, en los que se fija el instante de tiempo que constituye su referencia, se puede controlar su tasa de cuenta de tiempo, así como controlar su marcha.
- Proporciona un servicio de ejecución de tareas periódicas y retrasadas, al que un cliente de la plataforma puede encargar que ejecute repetidamente una función de callback cada cierto tiempo, o por una sola vez en un instante absoluto o relativo a la invocación.

Para obtener esta funcionalidad, se ha desarrollado un servicio distribuido con servicios locales en muchos nudos de la plataforma para minimizar las latencias, y que opera de acuerdo a los siguientes criterios:

- El reloj con el que se mide el tiempo es el reloj del procesador en el que se ejecuta el servico de tiempos local que se invoca.
- Todos los relojes de los diferentes nudos de la plataforma distribuida son sincronizados por el *TimeService* utilizando NTP. Cada Servicio de tiempo local supervisa el estado de sincronización del procesador local y mantiene la información estadística necesaria para cualificar los tiempos que se obtienen de él.
- Los tiempos y los intervalos que se obtienen del servicio son estructuras de datos con definición binaria bien definida y acorde con el estándar de OMG. La interpretación de estas estructuras se

realizan a través de clases envolventes (*Wrapper*) que son creadas localmente por cada cliente.

- Todos los accesos a los servicios de *TimeService* (la gestión del servicio, el acceso a los relojes, y la invocaciones de las operaciones periódicas y retrasadas se realizan utilizando el middleware Ice.

2.Estándar OMG *Enhancement View of Time Specification*

2.1 Estándares OMG de gestión de tiempos en sistemas distribuidos.

El servicio *TimeService* que se especifica en este documento ofrece la funcionalidad definida por la organización OMG [11] en su documento "*Time Service Specification*" [9] OMG formal/02-05-01, y la funcionalidad más avanzada de *Clock Service* definida en el documento "*Enhacement View of Time Specification*" [10] OMG formal/04-10-04.

La organización OMG, propone en estos documentos, un servicio, denominado *TimeService* [10] que permita al usuario obtener el tiempo actual, junto con la estimación del error asociado a él. Además del tiempo actual, el servicio también proporciona medios para:

- Establecer de forma fiable el orden en el que los eventos se han generado.
- Calcular y comparar intervalos de tiempo definidos por dos eventos que lo delimitan.
- Generar relojes especiales de forma que el cliente que los utiliza pueda definir el instante de referencia con respecto al que miden el tiempo, la escala de tiempos con la que miden y el control de su marcha.
- La generación de eventos temporizados basados en el reloj.

2.2 Representación del tiempo y caracterización de los relojes

Representación del tiempo

En los sistemas informáticos existen muchos formatos diferentes para representar el tiempo [10] tabla 2.1. Por ejemplo, en *X/Open DCE Time Servic*e se definen tres representaciones binarias del tiempo absoluto, mientras que en el UNIX SVID se define otra representación del tiempo diferente.

	Linux Time [12]	***Java Time [13]***	***UTC Time [14]***	***Windows Time [15]***		***Gregorian Time***
				32bits	***DOS***	
epoch	01-01-1970	01-01-1970	01-01-1958	01-01-1601	1980-01-01	15-10-1582
resolution	10 msg	1ms		100 nsg		
tamaño de palabra	64 bits	64 bits	64 bits	64 bits		

Tabla 2.1: Representación del tiempo en diferentes sistemas

A fin de garantizar la coherencia y compatibilidad de los diferentes servicios de tiempo que se ofrecen, tanto en el módulo *TimeService* como en el *ClockService* se utiliza un único formato de representación del tiempo que se denomina *TimeT* y en la que se formulan las lecturas de cualquier reloj definido en la especificación. Las características de esta representación son:

Unidad de tiempo: 100 nanosegundos (10^{-7} segundos)
Tiempo base: 15 October 1582 00:00:00.
Rango aproximado: AD[3] 30,000

La correspondiente representación binaria para los tiempos relativos tiene características similares a las de la absoluta:

Unidad de tiempo: 100 nanosegundos (10^{-7} segundos)
Rango aproximado: +/- 30,000 años

3 Anno Domini: Es una designación para numerar los años en calendarios Julianos y Gregorianos

Características de los relojes

Los relojes tienen un conjunto de características con las que se pueden decidir si son útiles o no para una aplicación particular. Las características que se definen para los relojes en esta especificación son:

- **Resolución** *(resolution)*: Representa la granularidad de las medidas del reloj. También representa el intervalo de tiempo durante el cual la lectura del reloj no cambia. La resolución es habitualmente la inversa de la frecuencia del oscilador que gobierna al reloj.
- **Precisión** *(precision)*: Representa el número de bits proporcionados por la lectura del reloj y su escala. Tanto la precisión como la resolución son muy importantes para asegurar la unicidad del tiempo así como para poder establecer una comparación entre dos tiempos.
- **Estabilidad** *(stability)*: Representa la capacidad de un reloj para garantizar medidas de intervalos de tiempo que sean consistentes con el tiempo físico que ha transcurrido, o lo que es lo mismo, para marcar el paso del tiempo con una frecuencia constante. La estabilidad se mide mediante un conjunto de derivadas de la frecuencia del reloj, que describen su variación frente al envejecimiento del sistema físico, o frente al cambio de factores ambientales (por ejemplo temperatura).

Estas características son inherentes a cualquier reloj y su medida debe realizarse con referencia a una fuente de tiempo aceptada como estándar. Para ciertos sistemas, la caracterización del reloj se obtiene a través de medidas estáticas previas, o de la especificación del fabricante. Los relojes que son caracterizados de esta forma se denominan relojes no coordinados (*uncoordinated).*

Cuando en el sistema está presente más de un reloj, se define un conjunto de características que son relevantes para establecer la comparación de los tiempos que miden los diferentes relojes:

- **Offset**: Representa la diferencia entre las lecturas de dos relojes diferentes de un mismo instante de tiempo. Para gestionar relojes que soporten tiempo local y tiempo de sesión, el *offset* es subdividido en dos componentes: el offset deliberado (*deliberate offset)* y el offset aleatorio (*unsynchronized offset)*.
- **Sesgo** (*skew*): Representa la velocidad de cambio (primera derivada) del offset entre dos relojes (en un determinado instante de tiempo). También equivale a la diferencia entre las frecuencias de los dos relojes. Para poder caracterizar los relojes a fin de ajustar su frecuencias y sincronizar los errores, este parámetro se descompone en las componentes: sesgo deliberado (*deliberate skew*) y sesgo accidental (*accidental skew*). Cuando se utilizan relojes que admiten control (parar, continuar, etc.) es muy importante introducir una marca que indique si el reloj ha sido parado o inicializado dentro del intervalo de medida.
- **Deriva** *(drift)*: Representa la tasa de cambio del sesgo (segunda derivada del offset) entre dos relojes. Se debe definir una indicación o marca especial, en el caso de que el sesgo deliberado cambie durante un intervalo de medida.

Se puede caracterizar la exactitud (*accuracy*) de un reloj si sus medidas se pueden comparar con otro reloj aceptado como estándar, o sincronizado con él.

Existe un conjunto de protocolos destinados a ajustar los relojes físicos, de forma que constituyan relojes lógicos que estén sincronizados con otros relojes que se consideran maestros. En particular, el protocolo NTP permite sincronizar los relojes con servidores de tiempo primarios externos organizados en niveles (*stratum*).

Se definen características adicionales para los relojes coordinados. Estas características suelen ser específicas del protocolo de sincronización. En esta especificación, se definen las siguientes características para los relojes coordinados:

- **Escala de tiempo de coordinación** *(coordination time scale)*: La escala de tiempo con la que se coordina ya sea directamente o indirectamente. Habitualmente es UTC, pero también se utilizan otros elementos de Tiempo Universal y tiempo local (como por ejemplo, offset por zona de tiempo y hora del día).

- **Nivel de stratum de coordinación** *(coordination strata)*: Indica el nivel de coordinación con la fuente de tiempo de referencia última (habitualmente una fuente de tiempo externa).
- **Fuente de coordinación** *(coordination source)*: Es la fuente con la que se coordina el reloj.

La Especificación Temporal de OMG, incluye un conjunto de estructuras de datos para codificar todas estas características de un reloj, las cuales se pueden obtener del catalogo de relojes a través de la interfaz *ClockCatalog*.

2.3 Relojes de referencia y sincronización de relojes

La especificación de OMG incluye interfaces para sincronizar un reloj a un reloj master [10]. Un reloj master es aquel que proporciona valores de tiempo garantizados en exactitud a los requerimientos que establece la aplicación, bien porque la naturaleza de la fuente de tiempo hardware que le sirve de base lo cumple, o bien porque está sincronizada con otro reloj master (*external clock synchronization)* que tiene las características requeridas.

La sincronización de un reloj con un reloj master, consta de dos pasos:

1. Determinar la diferencia entre los relojes. A veces, este paso puede consistir en una lectura simple del reloj master, aunque lo habitual, es que sea un proceso de ajuste continuo el que lea periódicamente el reloj master, para conseguir sincronizar los relojes de forma gradual con unos errores acotados.
2. Aplicar la corrección obtenida a partir de la información estadística del servicio a las lecturas que solicitan las aplicaciones cliente.

El proceso de sincronización necesita ser realizado de forma semi-automática por un thread demonio, puesto que suele ser de larga duración. Sin embargo, la especificación proporciona ciertas opciones de control explícito que pueden ser gestionadas por el entorno de ejecución. Hay que hacer notar dos cosas:

1. La habilidad para realizar la sincronización descrita en el paso 1, está fuera de la especificación.
2. No se formulan interfaces específicas de interoperatividad, lo único que se requiere es la capacidad del reloj para realizar lecturas de tiempo sobre el reloj master remoto.

La especificación establece la relación entre los requisitos de sincronización y los requisitos que caracterizan las diferencias entre ambos relojes. En particular, las derivadas del offset entre los dos relojes solo estarán disponibles si los relojes están coordinados a través de un proceso de sincronización activa.

Las interfaces que soportan la sincronización de un reloj con un master son tres:

La interfaz *SynchronizeBase* añade una operación primitiva a la interfaz *Clock*, que permite determinar el offset entre dos relojes y el error que conlleva su determinación. Requiere que el reloj sea capaz de medir el intervalo de tiempo que transcurre en la lectura del reloj remoto (presumiblemente el master). La duración de este intervalo determina la exactitud con el cual el reloj se sincroniza al Master. La interfaz se introduce como un módulo para las aplicaciones que implementan algoritmos especializados de sincronización.

Se definen dos interfaces adicionales para la sincronización: la interfaz *Synchronizable* ofrece una operación tipo constructor para crear instancias de objetos que implementan la interfaz *SynchronizedClock*. La operación *newSlave* activa la convergencia del reloj esclavo para que converja el maestro. Estos relojes convergen suavemente entre sí, y los parámetros de la operación establecen el error al que debe converger la diferencia para considerar finalizado el ajuste, así como los periodos con los que debe llevarse a cabo las lecturas del master.

La interfaz *SynchronizedClock* soporta actualizaciones periódicas de la información de sincronización. También proporciona medios para que la sincronización sea controlada a través de requerimientos explícitos de forma que se pueda recuperar la sincronización, cuando esta se ha perdido.

2.4 Arquitectura del servicio

Los elementos software (tipos, constantes e interfaces) que constituyen el *TimeService* se organizan por paquetes. En la figura 2.1, se muestran los paquetes definidos y la interdependencias entre ellos.

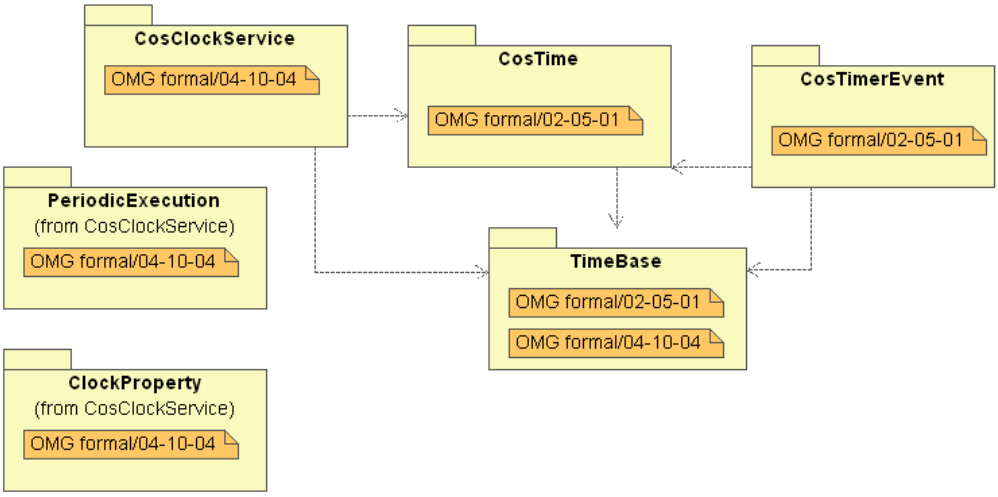

Figura 2.1: Arquitectura software del servicio TimeService [10]

El *Clock Service* reutiliza los tipos de datos definidos en el paquete *TimeBase*. El paquete *TimeBase* se define de forma separada a fin de que los otros servicios puedan hacer uso de los tipos que declara, sin la necesidad de utilizar interfaces del resto de paquetes. Las definiciones del paquete *TimeBase* de las especificaciones OMG formal/02-05-01 y OMG formal/04-10-04 son las mismas.

Las restantes definiciones de interfaces están contenidas en el paquete *CosClockService*, y en los paquetes agregados *ClockProperty* y *PeriodicExecution*.

El modelo de objetos definido en el paquete CosClock*Service* soporta múltiples fuentes de tiempo. La fuente de medida de tiempo base es la interfaz *Clock,* Figura 2.2. Ésta ofrece también una operación para acceder a las propiedades del reloj.

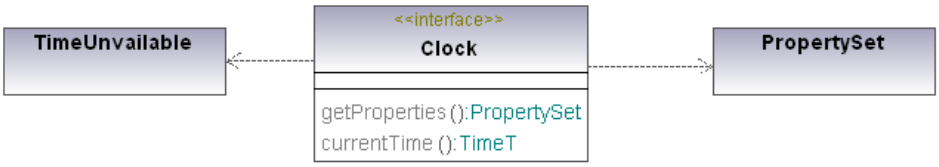

Figura 2.2: Esquema de la interfaz Clock

Además de la interfaz *Clock,* el paquete ***CosClockService*** incluye también las interfaces:

- Interfaz *TimeService*: Soporta la lectura del tiempo instantáneo UTC y de los intervalos de tiempo *TimeSpan* que son los nuevos tipos que sustituyen a los objetos encapsulados UTO[4] y TIO[5] generados en el servicio *TimeService* de la especificación OMG formal/02-05-01.
- Interfaz *SynchronizeBase*: Interfaz que ofrecen los servicios de tiempo para facilitar la conversión entre diferentes tipos de representación del tiempo.

4 UTO: Universal Time Object
5 TIO: Time Interval Object

- Interfaz *Synchronizable*: Incorpora una funcionalidad estandarizada que sirve de base para incorporar algoritmos propios especializados, como la creación de un SynchronizedClock.
- Interfaz S*ychroniz*edClock: Corresponde a un reloj capaz de sincronizarse con un determinado reloj maestro con unos errores de acotación especificados.
- Interfaz *ControlledClock*: Corresponde a un reloj que ofrece operaciones para parar, inicializar y reanudar la cuenta.
- Interfaz *PeriodicExecution*: Corresponde a objetos que gestionan la ejecución periódica y aplazada de operaciones definidas como métodos de objetos de aplicación. Esta interfaz retorna objetos que implementan la interfaz *Controller* y que son utilizados por la aplicación para gestionar el ciclo de vida de la ejecución lanzada.

El paquete *ClockProperty* está agregado como parte del paquete CosClockService, e incluye la definición de tipos, constantes y enumerados que describen las propiedades de los relojes.

Las propiedades de un reloj pueden resultar de un proceso de medida o pueden ser establecidas en tiempo de configuración como parámetro del fabricante. Las propiedades de un reloj son catalogadas como propiedades, y así son adecuadas para ser registradas en los servicios de nombres.

El esquema de la figura 2.3 corresponde con lo definido en la especificación OMG, y se conserva para mantener la semántica. Sin embargo en la implementación realizada todos las *properties* se describen mediante un string alfanumérico que representa su valor.

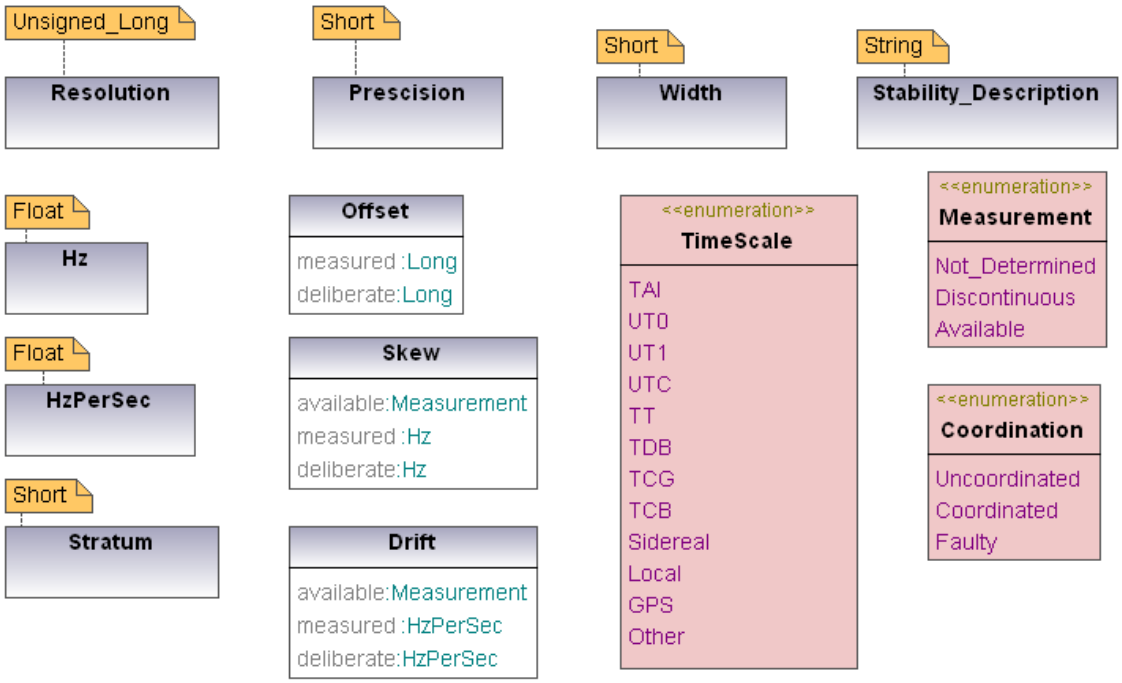

Figura 2.3: Propiedades soportadas por un reloj

2.5 Relojes UTC y tipos de datos para la representación del tiempo

El módulo *TimeBase* incluye todos los tipos de datos, enumerados y constantes que se usan en el servicio. Se definen como una estructura independiente para que los restantes módulos y servicios puedan hacer uso común de ellos sin necesitar para su acceso interfaces específicas, figura 2.4. Los tipos definidos en este paquete son válidos para los servicios definidos en las especificaciones OMG formal/02-05-01 [9] y OMG formal/04-10-04 [10].

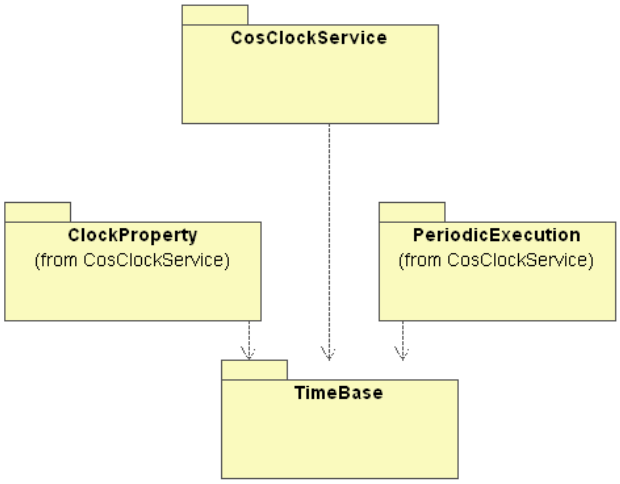

Figura 2.4: Estructura de paquetes que describen el servicio

El paquete *CosClockService* incluye las definiciones tipos e interfaces definidos para la gestión de los diferentes tipos de relojes que se definen en la especificación OMG formal/04-10-04.

El paquete ClockProperty es un paquete agregado en el paquete CosClockService que incluye los tipos, constantes, excepciones y enumerados que se utilizan para definir las características de un reloj. PeriodicExecution es un paquete agregado en el paquete CosClockService que incluye los tipos, constantes, excepciones y enumerados que se definen para la gestión de la ejecución periódica y aplazada de una actividad.

El paquete TimeBase, define los tipos de datos, enumerados y constantes que se muestran en la figura 2.5.

Tipo ***TimeT*** *=>* Representa un solo valor temporal, de 64 bits de tamaño, que mantiene el número en unidades de 100 ns que han transcurrido desde la base de tiempos. Para el tiempo absoluto, la base es el 15 de octubre de 1582 00:00 del Calendario Gregoriano. Todos los tiempos absolutos deben ser calculados utilizando fechas del Calendario Gregoriano.

Tipo **InaccuracyT** => Representa el valor de inexactitud en el tiempo, en unidades de 100 nanosegundos. Según la definición de la inexactitud en el campo de X/Open DCE *Time Service*, 48 bits son suficientes para mantener este valor.

Tipo ***TdfT*** => Es de 16 bits de tamaño y tipo short y mantiene el factor tiempo de desplazamiento en forma de minutos de desplazamiento con respecto al meridiano de Greenwich. Los desplazamientos hacia el Este del meridiano son positivos, mientras que hacia el Oeste son negativos.

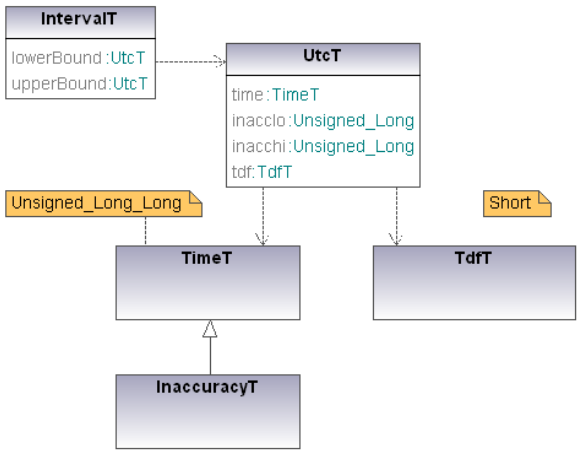

Figura 2.5: Estructura de los tipos de datos definidos en la especificación

Tipo ***UtcT*** => Define la estructura del valor de tiempo que se utiliza de manera universal en este servicio. El valor básico del tiempo es de tipo *TimeT* y se guarda en el campo *time*. El hecho de que una estructura *UtcT* almacene un tiempo relativo (es decir, una duración), o un tiempo absoluto viene determinado por el contexto; no existe ningún *flag* en el objeto que guarde la información de estado (hay que tener en cuenta que, si una estructura *UtcT* se utiliza para mantener un espacio de tiempo, su *tdf*, se pondrá a cero).

Los campos iacclo y inacchi juntos, guardan una estimación de la inexactitud de 48-bits. Estos dos campos juntos guardan un valor del tipo *InaccuracyT* almacenado en 48 bits. El campo *tdf* guarda la información de la zona temporal. Las implementaciones deben colocar el factor de desplazamiento de tiempo para la zona de tiempo local en este campo, siempre y cuando creen un UTO que exprese el tiempo absoluto.

El campo time de un *UtcT* se utiliza para expresar el tiempo absoluto guardando una estructura UTC, independientemente de la zona horaria. Por ejemplo, para expresar el tiempo 3:00 pm en Alemania (que se encuentra una hora al Este de la Zona de Tiempo Universal), el campo *time*, se pondrá a las 2:00 pm de la fecha determinada, y el campo *tdf* debe ser ajustado a 60. Esto significa que, para cualquier valor 'utc' perteneciente a un UtcT, la hora se puede calcular como:

utc.time + utc.tdf * 600.000.000

Fórmula 2.1

Nótese que es posible producir valores correctos de *UtcT* poniendo siempre el campo *tdf* a cero y fijando el campo time a la hora *UTC*, sin embargo, se recomienda que las implementaciones incluyan información del horario local de para el valor *UtcT* que se producen.

Tipo ***IntervalT*** => Este tipo guarda un intervalo de tiempo representado como dos valores **TimeT** correspondientes a los límites inferior y superior del intervalo. Una estructura **IntervalT** que contenga un límite inferior mayor que el límite superior no es válida. Para que el intervalo tenga sentido, la base de tiempos utilizada en el intervalo superior e inferior debe ser la misma, y el tiempo base no debe ser cubierto por el intervalo.

La interfaz UTC define las operaciones que pueden ejecutarse sobre un objeto que representa un tiempo. La interfaz UTC proporciona varias operaciones para el tiempo. Estas incluyen los siguientes grupos de operaciones:

- La construcción de un objeto que ofrece una interfaz UTC a partir de sus elementos (su UtcT) y la extracción de elementos de un UTC (como atributos de solo lectura).
- La comparación de tiempos.
- La conversión de un tiempo relativo a uno absoluto y la conversión a un intervalo.

La interfaz TimeSpan representa un intervalo de tiempo y define las operaciones relativas a los mismos. Permite:

- La construcción de un objeto que ofrezca la interfaz TimeSpan a partir de sus elementos y la extracción de elementos de un TimeSpan.
- Establecer comparaciones entre dos intervalos así como el tipo de solapamiento que se produce entre ellos.
- Construir un UTC donde la inexactitud sea igual al TimeSpan, y el valor temporal sea el punto medio del intervalo.

La interfaz UtcTimeService define las operaciones que son implementadas por un reloj que proporciona la obtención del tiempo actual.

- *Operación* ***universalTime****():UtcT raises(TimeUnaivalable)* => La operación universal_time devuelve el valor actual del tiempo y realiza una estima de la incertidumbre con la que se obtiene. Eleva la excepción TimeUnavailable para indicar un fallo del servidor de tiempos subyacente. El tiempo retornado en el UtcT por esta operación no tiene garantías de ser seguro o de confianza. Si existe algún tiempo disponible, esta operación lo devuelve.
- Operación ***secureUniversalTime****():UtcT raises(TimeUnavailable)* => La operación secure_-universal_time devuelve el tiempo actual en un UtcT solo si este tiempo tiene garantías de haber sido obtenido de forma segura. Para garantizar esto, el Time Service subyacente debe atenerse a los criterios que deben seguirse para tiempos seguros, presentados en el Apéndice B, de la especificación de OMG: "*Guías de Implementación*". Si existe alguna incertidumbre sobre el cumplimiento de estos criterios, entonces esta operación debe devolver una excepción del tipo TimeUnavailable. Por lo tanto, siempre se puede confiar en el tiempo obtenido mediante esta operación.
- Operación ***absolute_time****(in withOffset:UtcT): UtcT raises (TimeUnavailable)* => La operación absolute_time devuelve un nuevo UtcT que contiene el tiempo absoluto correspondiente al offset del actual, mediante el parámetro with_offset. Eleva la excepción DATA_CONVERSION si el intento de obtener el tiempo absoluto produce un overflow.

2.6 Relojes controlados o de misión

Ciertos relojes pueden ser parados, reanudados, inicializados o controlados de cualquier modo. Un ejemplo típico de este tipo de reloj es el cronómetro que controla el tiempo de juego en un partido de baloncesto. Para la gestión de los relojes de este tipo se ha definido la interfaz *ControlledClock*. Esta interfaz proporciona a los clientes las operaciones *start*, *stop*, *set*, o las que facilitan la modificación de la frecuencia del reloj.

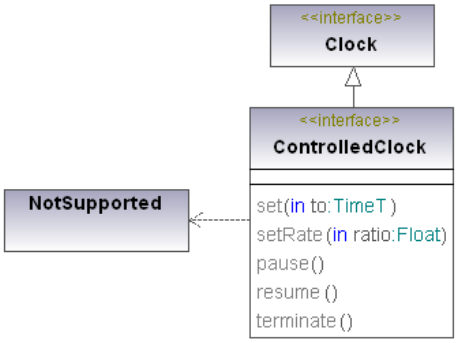

Figura 2.6: Interfaz ControlledClock

Los relojes controlados pueden tomar como tiempo de referencia instantes pasados o futuros, figura 2.7 y empezar a contar el tiempo tomando estos como instante inicial. Además pueden ir hacia delante o hacia

atrás en función de que el parámetro *ratio_factor* sea positivo o negativo o correr más rápido o más despacio en función de si éste es mayor o menor que la unidad.

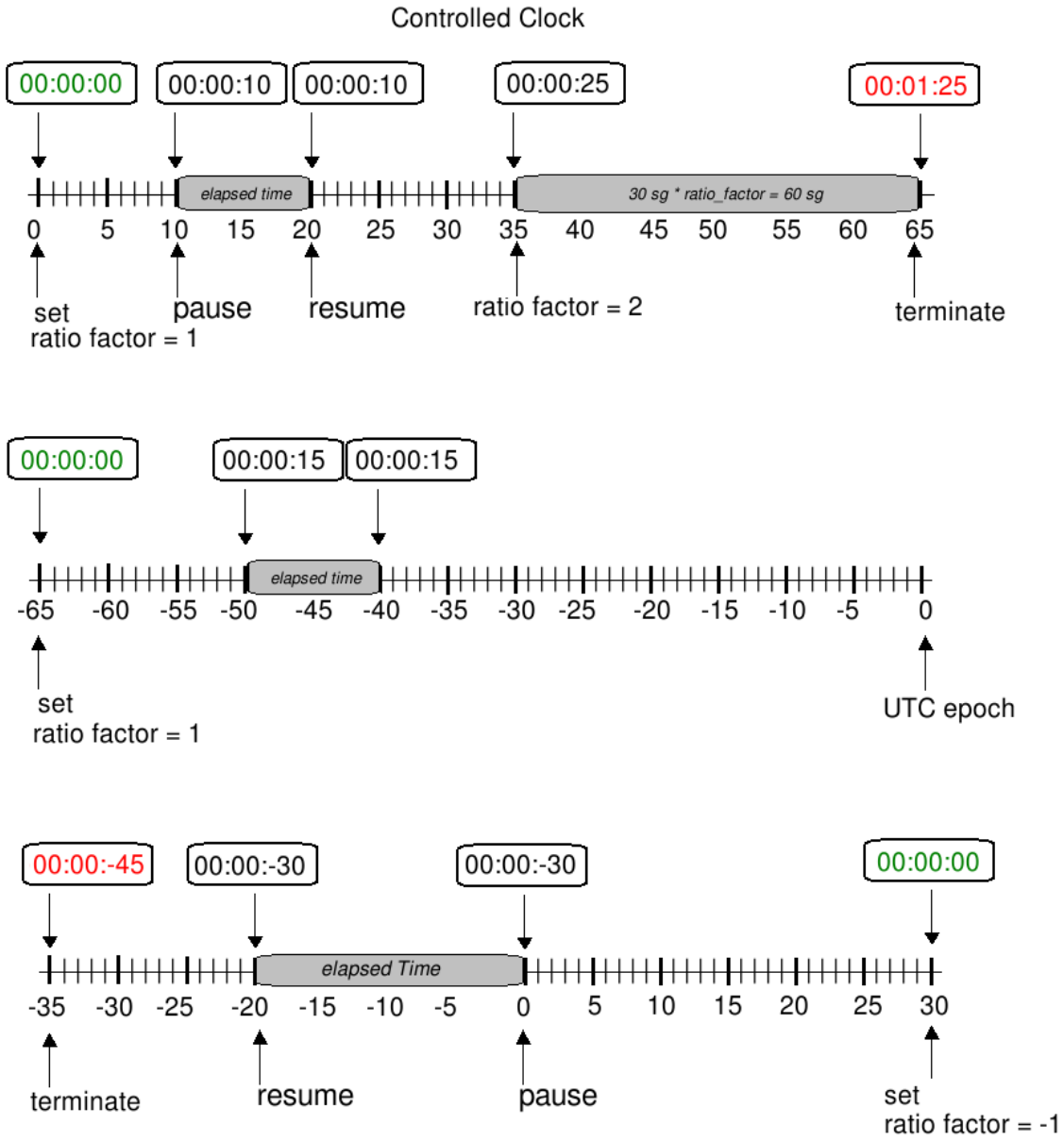

Figura 2.7: Funcionamiento de relojes controlados

2.7 Ejecuciones periódicas y aplazadas

En sistemas en los que hay requisitos de tiempo real, es habitual definir tareas que se ejecutan periódicamente. Estas ejecuciones periódicas suelen implementarse utilizando los recursos que proporcionan el sistema operativo o los threads de los lenguajes de programación. Sin embargo, estos métodos se basan en el reloj del sistema, y no sirven si se necesita utilizar un reloj especial concreto o remoto. Con este fin, se define la interfaz *PeriodicExecution.* A través de ella, se puede obtener una instancia de un objeto que ofrece la interfaz *Executor*, en la que invocando su método *execute* se ordena la ejecución de una tarea periódica. En el lanzamiento de la ejecución de una tarea periódica se obtiene la referencia a un objeto *Controller,* que constituye el mecanismo con el que se controla la ejecución en marcha. La invocación de una actividad periódica permite pasar un único parámetro de datos, el periodo de ejecución, y el offset que define su instante de inicio. Otras operaciones permiten ordenar la suspensión, reanudación y finalización de las actividades en ejecución.

La especificación no proporciona medios para gestionar las excepciones que se generan durante la ejecución de estas tareas periódicas o aplazadas.

La especificación de OMG [10], no define ninguna interfaz específica para ordenar la ejecución aplazada de actividades. No obstante, este tipo de ejecuciones pueden realizarse a través de la invocación a la interfaz periódica, y especificar la cuenta de ejecución a 1.

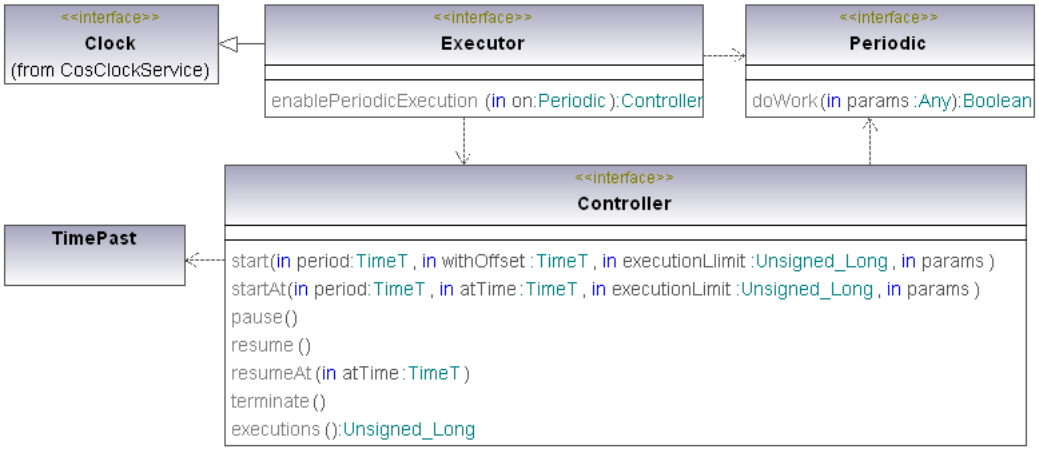

Figura 2.8: Interfaces relativas a la ejecución periódica y aplazada de tareas.

3.Sincronización de relojes y protocolo NTP

3.1 Protocolo NTP : The Network Time Protocol

NTP es un protocolo destinado a sincronizar redes de computadores a través de una red de comunicación [16] [17] . NTP está diseñado para permitir la sincronización de relojes de forma que se ajusten lo máximo posible al Tiempo Universal Coordinado (UTC: Universal Time Coordinated), incluso en presencia de fuentes de tiempo erróneas, o cuando las fuentes de tiempo no se encuentren disponibles de forma momentánea.

Actualmente, la versión formalizada de NTP es la 3 y está descrita en el RFC 1305 [17] . La versión 4 supone una revisión significativa del estándar NTP, y es la versión de desarrollo actual, pero aún no ha sido formalizada en un RFC. La versión 4 simple (SNTPv4) está descrita en el RFC 2030 [18] .

3.2 Redes y niveles.

NTP Utiliza el protocolo UDP en el puerto 123 para comunicarse entre clientes y servidores [6]. La utilización de UDP, evita la utilización de ancho de banda de la red, en el caso de que un servidor de tiempo con muchos clientes deje de funcionar en un determinado momento.

Las redes NTP funcionan de modo jerárquico, en donde un número pequeño de servidores proporciona el tiempo a un gran número de clientes. Los clientes de cada nivel, o *stratum*, son a su vez servidores potenciales de más clientes en un número de *stratum* mayor. La numeración de los *stratums* va aumentando desde el servidor primario (*stratum* 1) hasta los niveles más bajos de la jerarquía, a los que corresponde el nivel de *stratum* más alto. Los clientes, pueden obtener información de varios servidores y determinar automáticamente la mejor fuente de tiempo, figura 3.1.

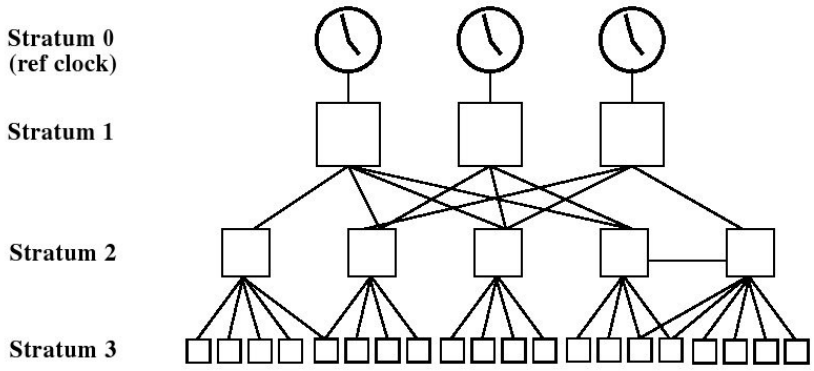

Figura 3.1: Modelo jerárquico de servidores en NTP [16]

Los relojes de referencia corresponden al nivel *stratum* 0. A su vez, los servidores conectados directamente a un reloj de referencia se denominan servidores *stratum* 1. Un cliente del servicio de tiempo nunca se puede conectar a un reloj *stratum* 0 directamente, siempre deberá sincronizarse a través de un *stratum* 1 o superior.

No obstante, un servidor, a pesar de ser *stratum* 1, puede incurrir en tiempos imprecisos si se configura para utilizar como reloj de referencia su propio reloj interno. Por esto, se suelen utilizar diversas fuentes de tiempo para establecer la sincronización de un sistema.

La exactitud de NTP, por lo tanto, siempre está limitada por la fuente de tiempo. Un cliente, sincronizado con un reloj inexacto, será a su vez inexacto. Esta dependencia se verifica a lo largo de toda la cadena de relojes de referencia. Si el reloj de referencia está mal calibrado, todos los clientes del mismo se verán afectados.

3.2.1 Tipos de clientes y servidores

La relación entre clientes y servidores NTP se puede configurar para operar de diferentes maneras. Por

ejemplo, una máquina puede ser cliente de otra con un número de *stratum* menor, a su vez, del mismo nivel o *peer* de otra con el mismo nivel de *stratum* y ser servidor de otras máquinas que tengan un nivel de *stratum* mayor que ella.

- Servidor: El servidor NTP es el encargado de proporcionar el tiempo a los clientes. El cliente lanza una petición al servidor y éste devuelve una respuesta con su correspondiente sello temporal, así como con información a cerca de su inexactitud y número de stratum.
- Cliente: El cliente NTP es quien obtiene las respuestas temporales de los servidores y utiliza esta información para calibrar su propio reloj. El cliente determina la diferencia entre su reloj y el del servidor y ajusta su tiempo para que coincida con el de éste. El error máximo se determina en base al tiempo de ida y vuelta del paquete temporal que el cliente espera recibir.
- Peer: Un peer en el contexto de NTP es un miembro de un grupo de servidores muy unidos. En un grupo de dos *peers*, en cualquier momento, el *peer* más preciso actúa como servidor, mientras que el resto actúan como clientes.
- Servidor broadcast/multicast: Los servidores broadcast y multicast envían actualizaciones temporales periódicas a la dirección de broadcast y multicast respectivamente. Este tipo de servidores resulta útil para disminuir el tráfico de red en sistemas con muchos clientes NTP.
- Cliente broadcast/multicast: Un cliente broadcast o multicast, permanece a la espera de un paquete NTP de la dirección de broadcast o multicast. Cuando se recibe el primer paquete, intenta cualificar el retraso hasta el servidor y así poder cuantificar mejor el tiempo correcto de las emisiones posteriores. Este esquema de funcionamiento, viene acompañado además de un tipo de intercambios breves donde el cliente y el servidor actúan como cliente y servidor NTP ordinarios (sin broadcast). Una vez que estos intercambios se llevan a cabo, el cliente tiene una aproximación del retardo de la red y procede a estimar el tiempo basándose solamente en los paquetes broadcast.

3.2.2 Precisión y Resolución

El protocolo NTP tarda varios minutos o incluso horas en ajustar el sistema al último grado de precisión.

El grado de sincronización de un servidor depende principalmente de la latencia de la red. La precisión de NTP depende por lo tanto del entorno de red. Como NTP utiliza paquetes UDP, la congestión de tráfico de red puede evitar temporalmente la sincronización, sin embargo, el cliente puede seguir ajustándose en base al histórico de la deriva (o *drift*) que mantiene.

En una red LAN en buenas condiciones sin *routers* ni otros elementos que introduzcan retardo en la red, lo normal es obtener una precisión de hasta pocos milisegundos. Cualquier elemento que añada latencia, tales como *hubs*, *switches* , *routers*, elevado tráfico de red, reducirán la precisión.

En una red WAN la precisión se encuentra típicamente en el rango de 10-100ms. En el caso de internet, la precisión de la exactitud no es predecible, por lo que se debe prestar mucha atención cuando se configura un cliente para utilizar servidores NTP públicos.

Si en un sistema se requieren precisiones mayores que las mencionadas, se pueden emplear diversas opciones para conseguirlo:

- Conectarse directamente a un reloj de referencia: La conexión directa a un reloj de referencia, implica una precisión limitada únicamente por el reloj de referencia y las latencias hardware y software involucradas en la conexión con el mismo.
- Pulsos Por Segundo (PPS): Utilizar receptores de radio PPS. Este sistema consigue precisiones del orden de decenas de microsegundos.
- Precisión del Tiempo del Kernel: Existen sistemas que poseen resoluciones del orden de microsegundos o incluso mayores. NTP, utiliza la precisión temporal de los kernels de la forma definida en el RFC 1589.

La resolución máxima de una marca temporal de NTP es de aproximadamente 200 picosegundos (el tiempo que tarda un pulso eléctrico en recorrer 2 cm de cable de cobre), por lo tanto la precisión de NTP va a estar más limitada por latencias del hardware que por el propio protocolo NTP.

3.3 Algoritmos de sincronización de reloj

3.3.1 Estimación del offset y del error

sg (0,005 +/- 0,03/2).

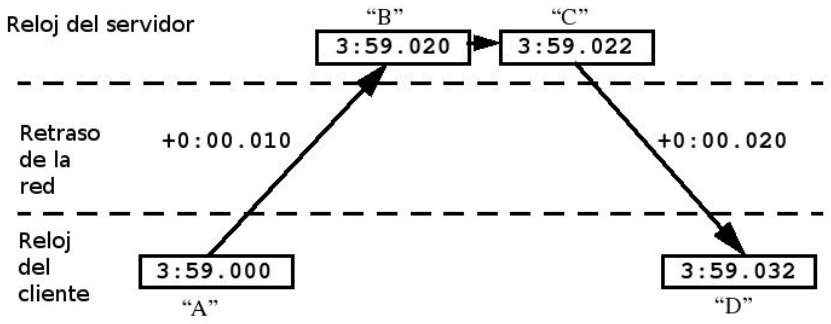

Figura 3.3: Muestra de intercambio entre un cliente y un servidor [16]

NTP no conoce la disparidad real entre los relojes, pero asume (para configuraciones de un solo servidor) que el ajuste correcto del cliente es exactamente el punto medio del intervalo. En el ejemplo anterior, el reloj se ajustaría 0,005 segundos. En realidad, para que los relojes se sincronicen realmente, el ajuste debería haber sido 0,01 segundos. Cuando NTP utiliza múltiples relojes, el procedimiento es mucho más complejo. NTP elige el mejor reloj con el que sincronizarse en base al stratum, la latencia de la red y la precisión solicitada. Si un reloj se selecciona como preferido, será éste el que se elija, a menos que su tiempo esté lo suficientemente alejado de los otros servidores NTP disponibles como para que se considere que está funcionando mal.

Cuando se sincronizan múltiples relojes NTP, los límites de error determinan el valor de ajuste del reloj. Un entorno con una combinación de múltiples relojes puede resultar confuso y llevar a localización de problemas. La siguiente sección describe el funcionamiento de la combinación de relojes.

3.3.2 Algoritmos de filtrado e intersección

Una vez se recibe un paquete, éste entra en una cola de paquetes de la misma fuente. La cola contiene 8 posiciones y cuando un nuevo paquete llega, el paquete más viejo es descartado. NTP utiliza un algoritmo de filtrado para reducir los efectos de pequeños errores en la precisión. El algoritmo de filtrado proporciona como salida valores que representan la mejor predicción para el *offset* actual y el máximo error de un reloj determinado. El funcionamiento detallado del algoritmo de filtrado está descrito en el RFC 1305 [17].

La existencia de la cola y su modo de funcionamiento, afectan al comportamiento del protocolo y a su velocidad de sincronización con nuevos servidores. Cada vez que se desea sincronizarse con una fuente nueva, es necesario esperar hasta recibir de ésta al menos 5 paquetes válidos.

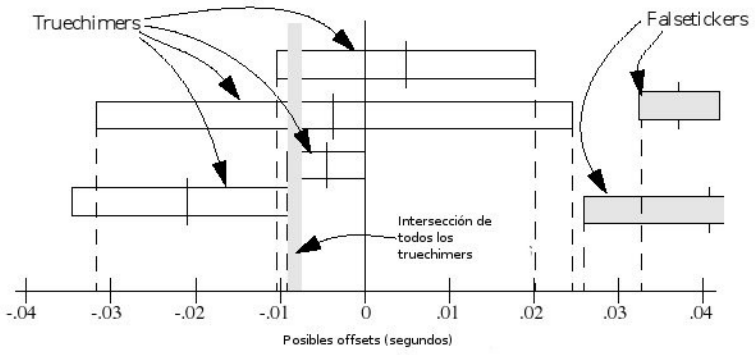

Figura 3.4: Algoritmo de Intersección [16]

Para filtrar, NTP utiliza los intervalos cuyos errores se solapan en un intervalo (*truechimers*), los relojes cuyos valores quedan fuera del intervalo (*falsetickers*) se descartan. En la mayoría de los casos, los *falsetickers*, son servidores que están utilizando su propio reloj interno como reloj de referencia. La variación del reloj es la causa de que poco a poco el servidor se convierta en un *falseticker*. La figura 3.4 muestra un diagrama del funcionamiento del algoritmo de intersección.

3.3.3 Algoritmo de clustering

El algoritmo de clustering ordena todos los *truechimers* basándose en una ponderación de diversos factores para determinar la calidad del servidor (*stratum*, dispersión del reloj maestro, retraso al reloj maestro, retraso con el cliente y variación del error). El campo *stratum* siempre es el que tiene mayor preponderancia en la selección. La lista se recorta entonces de forma que sólo queden en ella los 10 relojes mejores.

Se selecciona entonces el mejor de los relojes en base al análisis histórico, a la dispersión y al error asociado a cada reloj. Si la fuente actual de sincronización permanece activa, se utiliza automáticamente, si no, se selecciona el mejor reloj de la lista como nueva fuente de sincronización.

3.3.4 Algoritmo de combinación de relojes

A pesar de que solamente se selecciona un reloj como fuente de sincronización, la corrección en el reloj del cliente se ve influenciada por todos los servidores activos seleccionados en el algoritmo de clustering. El *offset* de cada uno de ellos se pondera en función de su calidad (determinada por los factores descritos en el algoritmo de clustering). Se calcula la media de los *offset*s ponderados y ésta es la que se utiliza para ajustar el reloj del cliente.

3.4 Servicios NTP en LINUX

La distribución NTP está integrada por varios componentes. Entre ellos cabe destacar el demonio NTP, *ntpd*, que constituye el método usual de utilización de NTP. El modo de funcionamiento de este demonio, se establece en su fichero de configuración, donde se puede habilitar el acceso, los servidores de tiempo, la monitorización, la autenticación, la gestión remota y la aproximación temporal. En el momento que se desee cambiar la configuración de NTP, se deberá parar y reiniciar el funcionamiento del demonio.

El programa *ntpdate*, permite al cliente sincronizarse con un servidor NTP sin tener que mantener un demonio de forma permanente. Solo permite una transacción NTP y se puede utilizar para efectuar una sincronización inicial rápida antes de llamar al demonio. La carga de procesador del demonio *ntpd* es pequeña, por lo que no se ganará demasiada capacidad deshabilitándolo. Además *ntpdate*, al contrario que *ntpd*, no puede limitar la deriva del reloj basándose en un histórico de datos. *Ntpd* realiza un ajuste continuo, de forma que los cambios en el tiempo sean suaves y no tenga necesidad de realizar saltos temporales.

El programa *ntpq* permite realizar el seguimiento del estado del demonio *ntpd* tanto en una máquina local como remota. Es útil para un administrador ya que permite comprobar la configuración de un *host* remoto.

El programa *ntpdc* también permite realizar el seguimiento del estado del demonio *ntpd* en una máquina local o remota. Sin embargo, también permite realizar cambios de configuración de forma instantánea.

El commando *ntptrace*, proporciona información sobre el nombre del cliente, su número de *stratum*, su *offset* con respecto al *host*, la distancia de sincronización, y el identificador del reloj de referencia asociado al servidor (si es que existe).

Si se desea se puede hacer que NTP guarde información estadística del proceso de sincronización. Existen tres tipos de ficheros de información: *peerstats*, *loopstats* y *clockstats.* Para la mayor parte de los sistemas, el fichero más interesante es el *peerstats*. El fichero *clockstats*, sólo es útil en máquinas que estén conectadas directamente a un reloj de referencia y el fichero *loopstats* en general solo se utiliza si se requiere tiempo de precisión.

En el fichero *peerstats* se puede encontrar información de diverso tipo; como el tiempo en que ha llegado el

último paquete del servidor, la dirección IP del servidor que manda la información, el estado del sistema, el *offset* estimado con respecto al servidor (que representa cuánto de lejos se encuentra el reloj del cliente del servidor), el retraso de ida y vuelta de la red a ese host y la dispersión (que representa el máximo error posible en el *offset* estimado).

Dia	*Segundo*	*Dirección IP*	*Estado*	*Offset*	*Round trip delay*	*Dispersión*	*Skew (varianza)*
54760	27632.703	147.156.1.1	9414	0.050735679	0.023454185	0.005007214	0.009456763
54760	27754.724	91.189.94.4	9414	0.042006228	0.045157121	0.004124764	0.009854153
54760	27759.702	147.156.1.1	9414	0.050735679	0.023454185	0.005125865	0.011573720
54760	27881.724	91.189.94.4	9414	0.042006228	0.045157121	0.005393728	0.010542374
54760	27889.702	147.156.1.1	9414	0.050735679	0.023454185	0.005555123	0.014575232
54760	28011.724	91.189.94.4	9414	0.042006228	0.045157121	0.007147908	0.012601400

Tabla 3.1: Ejemplo del fichero peerstats generado por ntpd para una configuración concreta

- Campos 1 a 3:

El primer campo indica el día del Calendario Juliano para la entrada del log. Se puede determinar el dia a partir del 1 de enero de 2001 restando 51911 a este número.

El segundo campo es un stamp de tiempo que indica los segundos del día que han transcurrido. El tercer campo muestra la dirección IP del host al que se refiere.

- Campo 4:

Este campo representa el estado del host de la dirección IP mencionada en el campo 3. Este campo es el que contiene la información más importante para el administrador del sistema, ya que se puede determinar la configuración y estado para todos los servidores NTP de un cliente determinado. Este campo está constituido por cuatro bits hexadecimales. Su significado desglosado se puede ver en las figuras 3.5, 3.6 y 3.7 . Los dos primeros especifican la configuración del servidor:

Formato del código Status	significado
1xxx	El servidor ha enviado petición de sincronización peer con una máquina local pero no está configurado localmente
7xxx	El servidor es peer, es decir, no está configurado localmente pero se puede alcanzar y utiliza una autenticación correcta
8xxx	El servidor está configurado, pero o bien no es alcanzable o bien no está autenticado
9xxx	El servidor está configurado y es alcanzable
Cxxx	El servidor está configurado para usar autenticación pero no es alcanzable
Dxxx	El servidor está configurado para usar autentización y es alzanzable, pero no está utilizando una key fiable
Fxxx	El servidor está autenticado y es alcanzable
x0xx	El cliente rechaza al servidor por no pasar los sanity checks

Figura 3.5: códigos de estado en peerstats

Formato del código Status	significado
x1xx	El servidor supera los sanity checks, pero no está lo bastante cerca de otros servidores para superar el algoritmo de intersección
x2xx	El servidor supera los checks de corrección (algoritmo de intersección)
x3xx	El servidor supera los checks de selección (no es descartado porque existan demasiados (más de 10) servidores mejores que él)
x4xx	El servidor supera los algoritmos de clustering sin ser descartado a causa de su elevada dispersión
x5xx	El servidor podría ser una fuente de sincronización, pero está demasiado alejado, lo que significa que todos los relojes tampoco son fiables o también están demasiado alejados
x6xx	El servidor es la fuente de sincronización actual

Figura 3.6: códigos de estado en peerstats

Los dígitos tercero y cuarto indican eventos que suceden en el cliente. El tercer dígito indica el número de eventos que han sucedido desde el último error de NTP. Deja de incrementarse al llegar a 15 (F en hexadecimal). Para un servidor que funcione perfectamente, este valor debería ser xx1x a menos que se haya invocado a ntpq después de que el servidor haya comenzado, en cuyo caso debería valer 0.

Código de evento	Descripción del evento
xxx0	Evento no especificado (o bien no ha sucedido ningún evento, o bien ha sucedido algún tipo de error especial)
xxx1	Error de IP ocurrido al alcanzar el servidor
xxx2	Incapaz de autenticar el servidor que solía ser fiable (Indica que las keys han cambiado o que alguien se está haciendo pasar por el servidor)
xxx3	Un servidor alcanzable se vuelve inalcanzable
xxx4	Un servidor inalcanzable se vuelve alcanzable
xxx5	El reloj del servidor presenta un error

Figura 3.7: códigos de evento en peerstats

- Los campos 5 – 7

El quinto campo del fichero peerstats muestra el offset estimado para un host en particular. Este valor se expresa en segundos. El sexto campo representa el retraso de ida y vuelta hasta ese servidor, también en segundos. El séptimo y último campo muestra la dispersión en segundos. La dispersión representa el error máximo posible del offset estimado. En otras palabras, si un servidor mantiene un tiempo correcto, la diferencia temporal entre los clientes y el tiempo correcto debe enmarcarse entre el offset menos la dispersión y el offset más la dispersión.

- El campo 8:

El octavo campo, skew, representa la diferencia entre las frecuencias de los dos relojes.

4.Implementación del reloj UtcTimeService sobre ICE

4.1 Estrategia de implementación del servicio de tiempo

La implementación del servicio TimeService que se propone, va destinada a una plataforma distribuida y heterogénea constituida por nudos con Sistema Operativo Windows y nudos con sistema operativo LINUX, que ejecutan aplicaciones desarrolladas en los lenguajes C/C++ y Java.

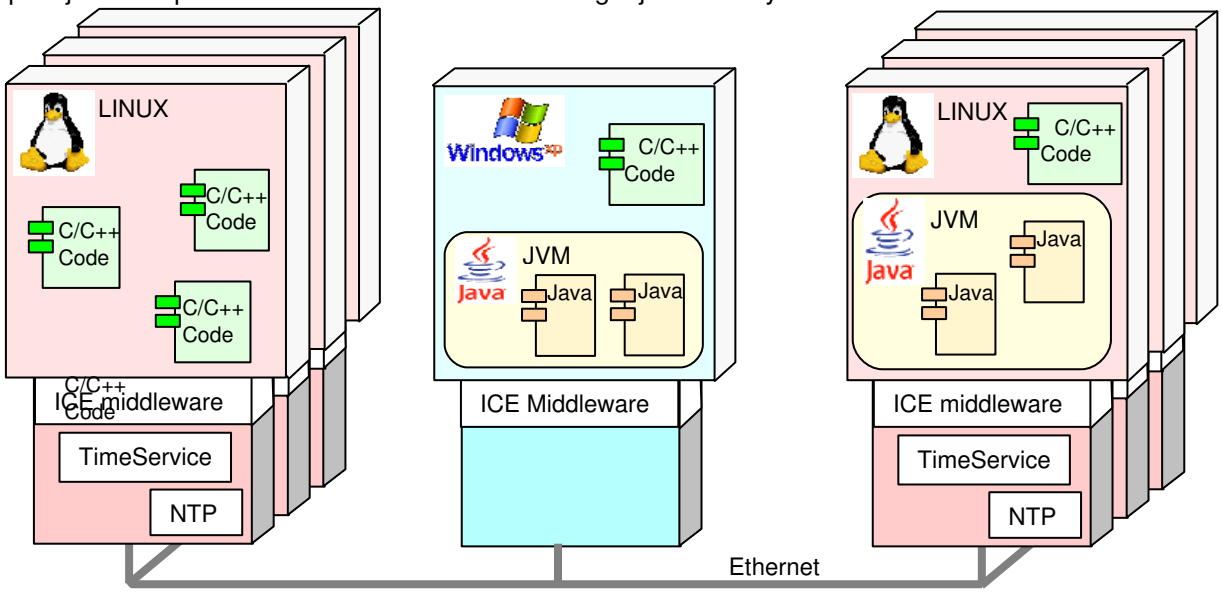

Figura 4.1: Plataforma de referencia

Los objetivos y la funcionalidad del TimeService son los establecidos en el estándar "*Enhacement View of Time* Specification" OMG formal/04-10-04.

Todos los nudos de la plataforma están conectados entre sí por una red Ethernet, y en todos los nudos existe el middlware ICE como base de comunicación entre las aplicaciones y los servicios.
Algunos de los nudos tienen acceso a servidores de tiempo externos oficiales:

- Vía GPS para nudos situados en zonas abiertas.
- Vía Radio (WWV en Estados Unidos, HBG en Suiza, MSF en Inglaterra o TDF en Francia).
- Vía Internet a través de los servidores NTP Servers

En el diseño del TimeService se ha priorizado la precisión en la determinación de los tiempos relativos medidos en los diferentes nudos de la plataforma frente a la precisión de los tiempos respectos del tiempo oficial.

La implementación del TimeService que se propone se basa en los siguientes criterios:
El sistema TimeService es un sistema distribuido con un agente local en cada nudo. Los TimeService ofrecen la interfaz pública ICE TimeServiceManaging, a través de la cual se configuran y coordinan como un único sistema distribuido a nivel de la plataforma.

- El reloj de sistema de cada nudo es la base del TimeService local. El TimeService local gestiona la sincronización del reloj del sistema con los relojes de los otros nudos, y mantiene la información estadística necesaria para comparar los tiempos obtenidos en los diferentes nudos de la plataforma.
- En los casos de que el nudo disponga de un mecanismo de sincronización propio, el TimeService lo utiliza. En este caso, el TimeService es responsable de su configuración, control y sincronización, y obtiene de él la información estadística con la que cualifica los tiempos que se

miden en él. Este es el caso de nudos LINUX, en los que se utiliza la aplicación de sincronización estándar basada en el protocolo NTP.

- En los casos de que el nudo no disponga de mecanismos de sincronización comercial, el TimeService asume el mecanismo de sincronización y realiza la evaluación estadística que permite cualificar los tiempos medidos en él. Esta opción no está implementada en esta versión.
- Al TimeService de cada nudo se accede a través de un único punto público que implementa la interfaz ClockCatalog, cuya dirección está bien definida y a la que se puede acceder tanto localmente como remotamente. Esta interfaz proporciona el acceso a los relojes disponibles en el nudo, así como a las características de cada uno de ellos.
- La medida de un tiempo UTC en un nudo se realiza a través del único reloj que ofrece el TimeService que implementa la interfaz UtcTimeService. Los tiempos UtcT que se obtienen de él son el resultado de una lectura del tiempo basada en el reloj del sistema y su cualificación en base a la información estadística que el TimeService tiene sobre la sincronización del reloj respecto de otros relojes de la plataforma. A este reloj se accede a través del ClockCatalog utilizando la clave "LocalClock".
- En cada TimeService local se pueden crear relojes especiales basados en el reloj del sistema del nudo. La creación o acceso a estos relojes se realiza a través del ClockCatalog del TimeService local. En la versión actual se han implementado dos tipos de relojes especiales:

  Clave: "ControlledClock" => Por cada invocación de la operación *getEntry()* con esta clave, crea un nuevo reloj que implementa la interfaz ControlledClock, del que retorna su acceso remoto.

  Clave: "Executor" => Una invocación a la operación *getEntry()* con esta clave, retorna el acceso remoto al único objeto que implementa la interfaz *Executor,* a través del que se puede gestionar invocaciones periódicas basadas en el reloj de sistema local.

La implementación que se propone corresponde a la arquitectura que se describe en el siguiente diagrama de clases:

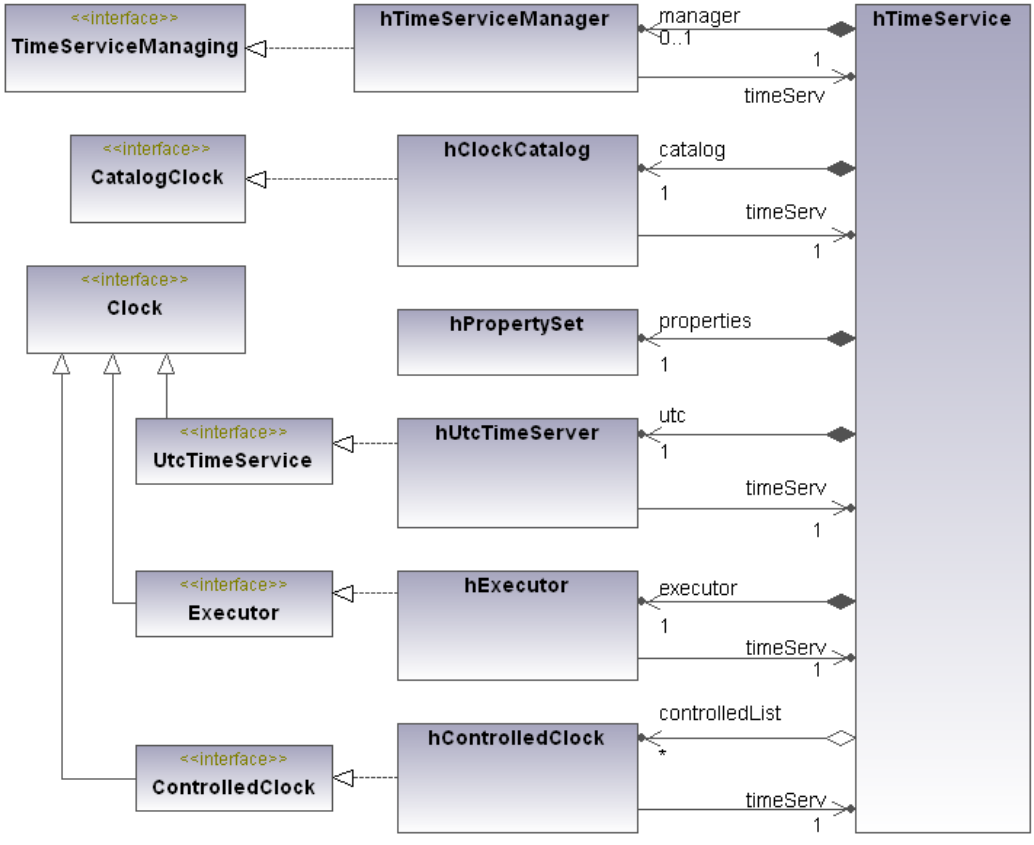

Figura 4.2: Arquitectura del TimeService.

4.2 hTimeService: Gestión y configuración de la sincronización del reloj local

La clase hTimeService describe el servidor TimeService que se instala en un nudo LINUX. La clase es instanciada por el procedimiento main() que se ejecuta para la instalación del servicio. Contiene todos los recursos que se necesitan disponer para implementar los servicios definidos en esta especificación.

Se basa en la instalación y mantenimiento de un servicio NTP que mantiene el reloj del sistema sincronizado con los restantes servicios de la red, y en la utilización del propio reloj del sistema como base de la funcionalidad del servidor TimeService.

Las funciones que implementa hTimeService son:

1. Lee e interpreta el fichero TimeServiceConfig que establece la jerarquía de dependencia NTP del nudo respecto de los otros nudos de la plataforma distribuida. Por defecto este fichero es "hTimeServiceConfig.xml". Pero opcionalmente puede ser otro cuya ruta se pasa como parámetro del constructor.
2. Es el contenedor de todos los elementos que constituyen el TimeService. Cuando se construye una instancia de la clase hTimeService, recursivamente se construyen todos los elementos que constituyen el TimeService.
3. Es directamente responsable de la gestión y control del reloj del sistema en el que se basa el TimeService. En particular, es responsable de la sincronización del reloj del sistema. Bajo su control se gestiona el software NTP que mantiene el reloj sincronizado con el servidor que corresponda.
4. Periódicamente supervisa la información estadística que genera el software NTP y que representa el estado de sincronización del nudo con respecto al que en ese momento está establecido como servidor. En función de esta información, evalúa las características de los relojes que dependen del estado de sincronización del reloj.
5. Detecta cuando el servidor de NTP falla, y en tal caso, selecciona el nuevo servidor NTP que debe ser utilizado. Para ello sigue la secuencia de selección establecida como parámetro de configuración.
6. Implementa e instancia un servidor *hClockCatalog*, que constituye el único punto de acceso al servicio *TimeService*. El servidor hClockCatalog implementa la interfaz ICE *CatalogClock* que se registra de forma bien definida a fin de poder ser accedida por cualquier cliente.
7. Instancia un único servicio *UTCTimeService*, que constituye el medio a través del que cualquier cliente puede obtener tiempos absolutos, y definir intervalos temporales. Ofrece la interfaz ICE *UtcTimeService*, a la que se puede acceder a través del servicio CatalogClock, utilizando la clave "LocalClock".
8. Bajo requerimiento de un cliente, implementa e instancia la estructura de datos que se necesita para implementar un *ControlledClock*, y retorna el acceso a la interfaz ICE *ControlledClock*, a través de la que se controla. La instanciación de un nuevo *ControlledClock* se realiza a través del servicio *CatalogClock* utilizando la clave "ControlledClock".
9. Implementa un servicio de ejecución de tareas periódicas. A la interfaz *Executor* de este servicio, se accede a través del servicio CatalogClock con la clave "Executor".

En el diagrama de clases de la figura 4.3 se describen los elementos más relevantes de la clase hTimeService.

Operaciones de la clase hTimeService:

- *hTimeService()*:void => Constructor del servicio. Es invocada por el método main() que instancia el servicio. Debe:
    1. Leer el fichero de configuración, y establecer los valores de los atributos básicos y establecer las propiedades del servicio que son dependientes del fichero de configuración.
    2. Instanciar todos los elementos que constituyen el servicio, y establecer entre ellos los enlaces que se necesitan para operar.
    3. Activar los procesos y activar y dar alta en el middleware de los servicios que son públicos.

  instanceICEServers(): void => Construcción, activación y gestión de los servidores ICE: El servicio requiere del middleware ICE la construcción y declaración del conjunto de servidores ICE que ofrece

permanentemente el servicio:

1. Servidor de gestión que implementa la interfaz *TimeServiceManaging*. Este servidor se declara con el identificador *managingIdentifier*, y en la dirección *managingPort*. Ambos son pasados como parámetros de configuración, son públicos y conocidos por cualquier cliente que haga uso del servicio.
2. Servidor de acceso a los relojes que implementa la interfaz *ClockCatalog*. Este servidor se declara con el identificador *catalogIdentifier* y en la dirección *catalogPort*. Ambos son pasados como parámetros de configuración, son públicos y conocidos por cualquier cliente que haga uso del servicio.
3. Servidor de acceso al reloj UTC que implementa la interfaz *UtcTimeService* . El servidor es único, y los clientes deben solicitar siempre el acceso (*proxy*) a través del servicio *ClockCatalog*, con la contraseña "LocalClock". El identificador y la dirección del servicio son privados para los clientes, no obstante, a fin de organizar la configuración global del sistema, en el fichero de configuración se proporcionan unos valores que pueden ser utilizados de manera opcional.

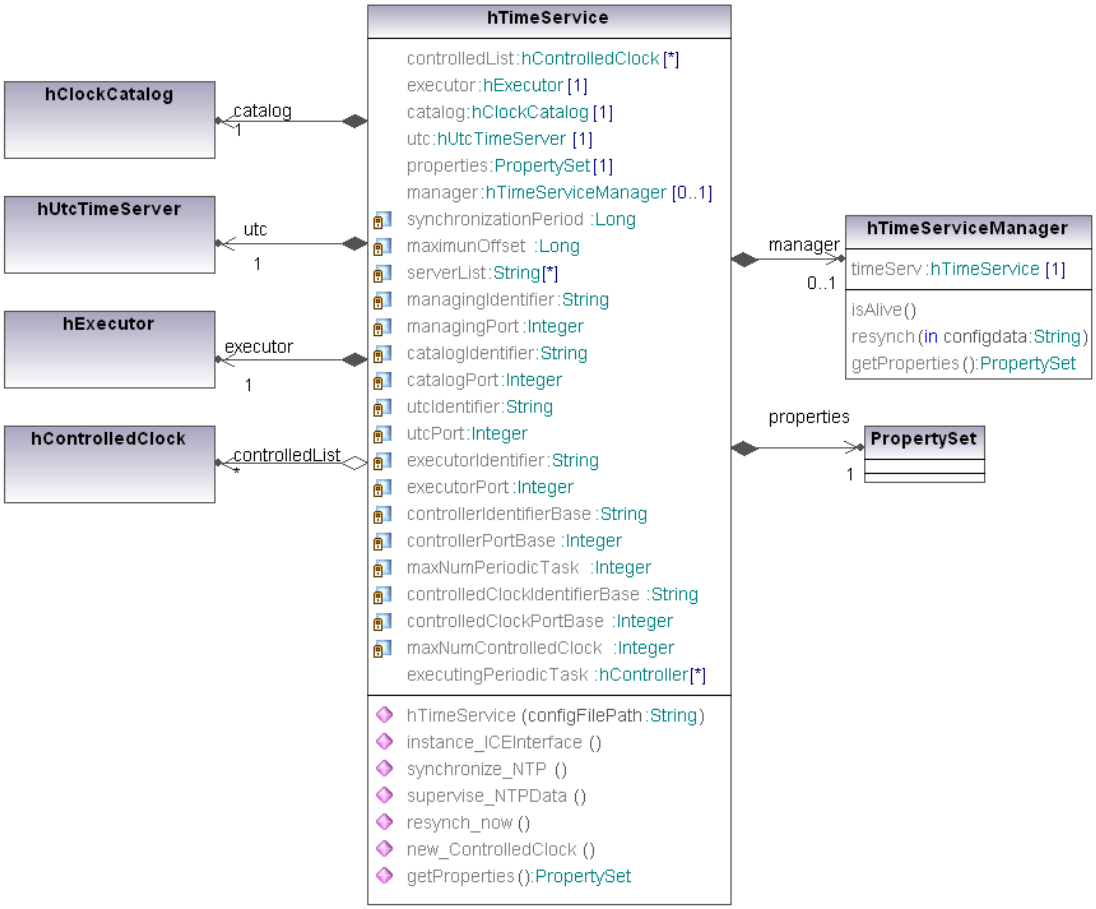

Figura 4.3: Clase hTimeService

4. Servidor de acceso al ejecutor de tareas periódicas que implementa la interfaz *Executor*. El servidor es único, y los clientes deben requerir siempre el acceso (*Proxy*) a través del servicio *ClockCatalog*, con la contraseña "Executor". El identificador y la dirección del servicio son privados para los clientes, al igual que en el caso anterior, a fin de organizar la configuración global del sistema, en el fichero de configuración se proporcionan unos valores opcionales. Por cada ejecución de una tarea periódica, el servicio de ejecución de tareas periódicas retorna el proxy a una interfaz del tipo *Controller.*

- *synchronized_NTP()*:void => Gestión de la sincronización NTP. Proceso por el que se activa la sincronización del procesador local con el servidor que se haya programado en el fichero de configuración. Este proceso implica:

  1. Construir el fichero de configuración NTP a partir de los datos de configuración del servicio.
  2. Sincronización gruesa inicial del procesador con el servidor NTP.
  3. Activación del proceso de sincronización fina y continua del reloj a través del NTP.

  Cada nudo recibe como parámetro de configuración, una lista de servidores NTP:

  - Si el nudo que se configura es el primero de esa lista, se configura como servidor.
  - Si el nudo está en la lista de servidores, se configura como cliente de todos aquellos que le preceden en la lista.
  - Si el nudo no está en la lista, se configura como cliente de todos los nudos que se encuentran en la lista.

- *supervise_NTPData()*:void => Supervisión de la sincronización NTP: Se encarga de la construcción del proceso de monitorización periódica de los resultados de la sincronización NTP, de la ejecución de las operaciones necesarias para actualizar los valores de las propiedades que dependan de la sincronización, y en el caso de que se presenten problemas de vivacidad, de reconfigurar el sistema.
- *resynch_Now()*:void => Reinicia la sincronización del reloj con un nuevo servidor NTP. Se invoca cuando en la supervisión periódica de los datos de NTP encuentra que el servidor no transfiere datos.
- *new_ControlledClock()*:void => Función para la creación y gestión bajo demanda de los relojes de misión.
- *getProperties():PropertySet* => Retorna el conjunto de propiedades definidas en el TimeService.

4.3 hClockCatalog: Gestión y acceso a los relojes

Constituye el punto de acceso de los clientes al TimeService. Implementa la interfaz *ClockCatalog* definida en el estándar, y que es ofertada como una interfaz ICE que permite a los clientes invocar funciones para acceder a los diferentes tipos de relojes del servicio a través de claves. El acceso de esta interfaz está bien definido por configuración, por lo que los clientes pueden construir u obtener el Proxy de acceso.

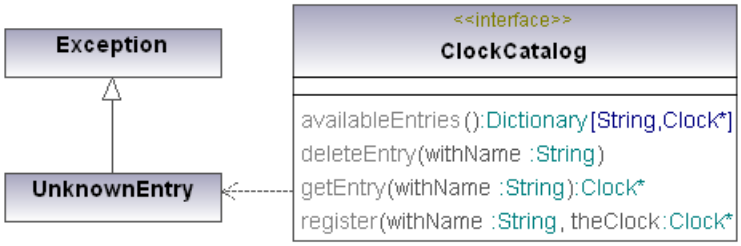

Figura 4.4: Interfaz ClockCatalog.

Interfaz **ClockCatalog**: La interfaz ClockCatalog permite a las aplicaciones descubrir y seleccionar un reloj para su uso. Tiene la intención de ser una alternativa ligera a la utilización del Trading Service (por ejemplo en sistemas embebidos).

Operaciones de ClockCatalog:

Operación ***getEntry****(withName:String): Clock* raises (UnknownEntry)* => Recupera el Proxy de acceso a un *Clock* a partir del nombre que tiene asignado o con el que se registró. Eleva la excepción *UnknownEntry* si el parámetro withName no corresponde a un reloj previamente registrado.

Operación ***availableEntries****():Dictionary<String, Clock*> =>* Recupera el catálogo completo de forma que el cliente pueda seleccionar un reloj basándose en sus propiedades conocidas.

Operación ***register****(withName:String, entry: Clock*)=>* Registra un nuevo reloj en el catálogo.

Operación ***deleteEntry(withName: String)*** => Elimina la entrada que se registró con el nombre que se pasa como argumento. No tiene efecto si el argumento corresponde a una clave predefinida ("LocalClock", "Controlled" y "Ejecutor"). Eleva la excepción *UnknownEntry* si el parámetro withName no corresponde a un reloj previamente registrado.

La excepción ***UnknownEntry*** *:* Indica que el catálogo no contiene entradas con el nombre especificado.

La clase concreta hClockCatalog implementa la interfaz ClockCatalog, y por tanto ofrece las operaciones declaradas en ella.

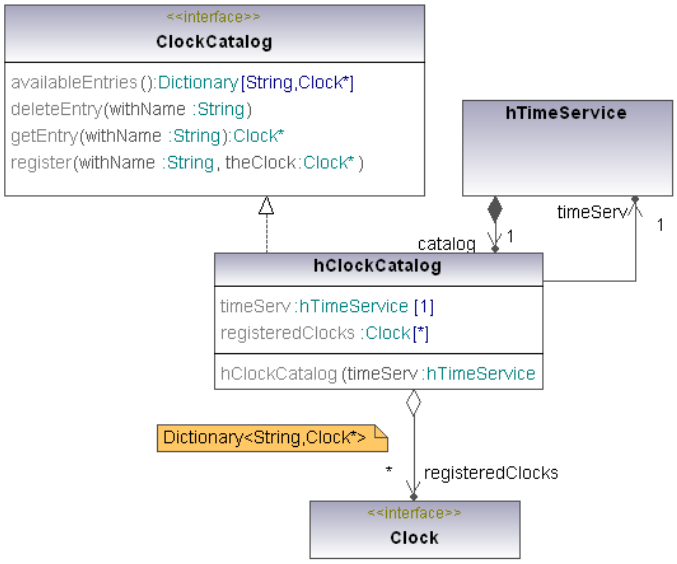

Figura 4.5: Estructura de la clase hClockCatalog

En la implementación de la clase hClockCatalog existen dos atributos:

Atributo ***timeServ: hTimeService*** => Referencia a la instancia del servicio en el que está definido. Lo utiliza para implementar su funcionalidad.

Atributo ***registeredclock: Dictionary<String, Clock*>*** => Constituye una estructura tipo diccionario, en el que se almacenan los proxys de los *Clock* que se registran, indexados por su nombre que es un *String*.

El constructor de la clase es:

Constructor **hClockCatalog(timeServ: hTimeService)** => Construye una instancia de hClockCatalog que se instancia en el hTimeService cuya referencia se pasa como argumento.

4.4 hUtcTimeService: Medida y cualificación del tiempo

El elemento hUtcTimeService constituye el reloj base a través del cual los clientes leen de forma fiable el tiempo actual. Implementa la interfaz definida por el estándar UtcTimeService.

Cada TimeService implementa un único hUtcTimeService que puede ser accedido por cualquier cliente que requiera datar sus eventos. Los clientes obtienen el acceso al servicio UtcTimeService a través de ClockCatalog utilizando la clave "LocalClock".

Las operaciones que ofrece el servicio UtcTimeService son:

Heredadas de la interfaz Clock:

currentTime*() : TimeT* => Devuelve el tiempo actual medido por el reloj.

getProperties*(): PropertySet* => Retorna las propiedades declaradas del reloj.

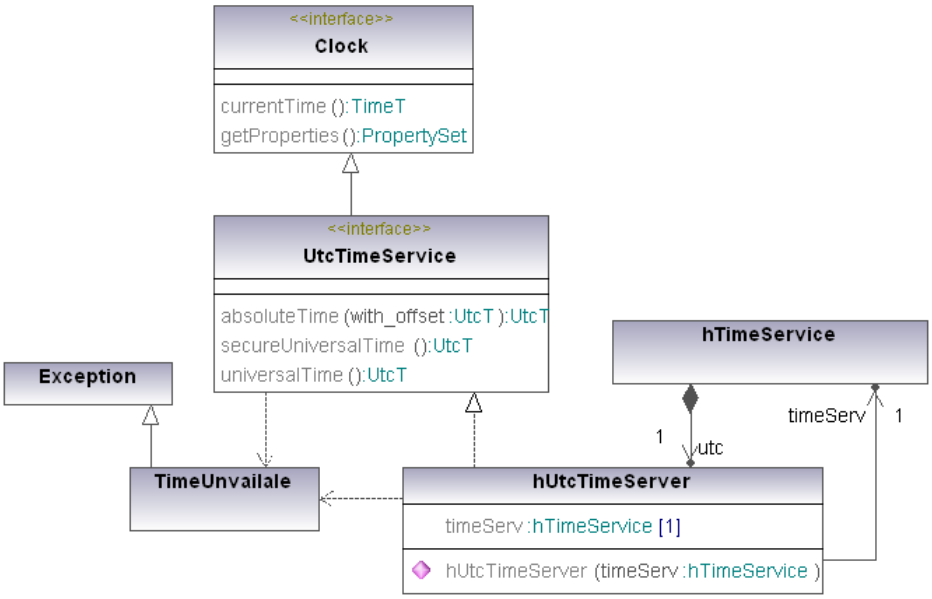

Figura 4.6: Servicio hUtcTimeService

Heredadas de la interfaz UtcTimeServer:

universalTime*():UtcT raises(TimeUnaivalable)* => Devuelve el valor actual del tiempo y realiza una estima de la incertidumbre con al que se obtiene. Eleva la excepción TimeUnavailable para indicar un fallo del TimeService. El tiempo retornado en el UtcT por esta operación no tiene garantías de ser seguro o de confianza. Si existe algún tiempo disponible, esta operación lo devuelve.

secureUniversalTime*():UtcT raises(TimeUnavailable)* => Devuelve el tiempo actual en un UtcT solo si este tiempo tiene garantías de haber sido obtenido de forma segura. Para garantizar esto, el TimeService comprueba:

- Si el servidor NTP con el que se sincroniza el reloj del time Service está activo y ha actualizado el tiempo dentro del tiempo establecido por los parámetros de configuración.
- El offset que actualmente se obtiene respecto del Server NTP es inferior al máximo definido en los parámetros de configuración *maximunOffset*.

Si existe alguna incertidumbre sobre el cumplimiento de estos criterios, entonces esta operación debe devolver una excepción del tipo TimeUnavailable. Por lo tanto, siempre se puede confiar en el tiempo obtenido mediante esta operación.

absolute_time*(in withOffset:UtcT): UtcT raises (TimeUnavailable)* => La operación *absolute_time()* devuelve un nuevo UtcT que contiene el tiempo absoluto correspondiente al offset del actual, mediante el parámetro with_offset. Eleva la excepción DATA_CONVERSION si el intento de obtener el tiempo absoluto produce un rebase de valor.

4.5 hUTC y hTimeSpan: Gestión de instantes de tiempo e intervalos temporales

La clase hUTC es una clase envolvente de un tipo UtcT definido en el paquete TimeBase y que representa un instante de tiempo (time) representado mediante una estructura binaria de 128 bits (16 bytes). La

estructura contiene:

- Tiempo (*time*) formulado como un entero de 64 bits que representa el tiempo transcurrido en unidades de 100 ns, y tomando como referencia en inicio del calendario gregoriano 15-10-1582/00:00:00.
- Inexactitud (*inaccuracy*) formulado como un entero sin signo de 48 bits, que representa el tiempo de incertidumbre formulado también en unidades de 100 ns.
- Zona de desplazamiento horario (*tdf*) formulado como un entero con signo de 16 bits, que representa el desplazamiento medido en minutos del origen horaria de la zona en que se mide el tiempo respecto del meridiano de Greenwich.

Existen dos interpretaciones del significado UtcT:

1. Representación de un instante de tiempo UTC que incluye la inexactitud con la que fue medida, a fin de poder ser comparado de forma fiable y segura con otro instante de tiempo.
2. Representación de un intervalo de tiempo UTC delimitado por los instantes [*time-inaccuracy/2, time+inaccuracy/2*].

La clase hUTC implementa la interfaz UTC, la cual proporciona funciones para una gestión sencilla de la información del instante temporal definido en el dato UtcT que la clase hUTC envuelve.

En el siguiente diagrama de clases se muestran las operaciones definidas en la interfaz UTC,

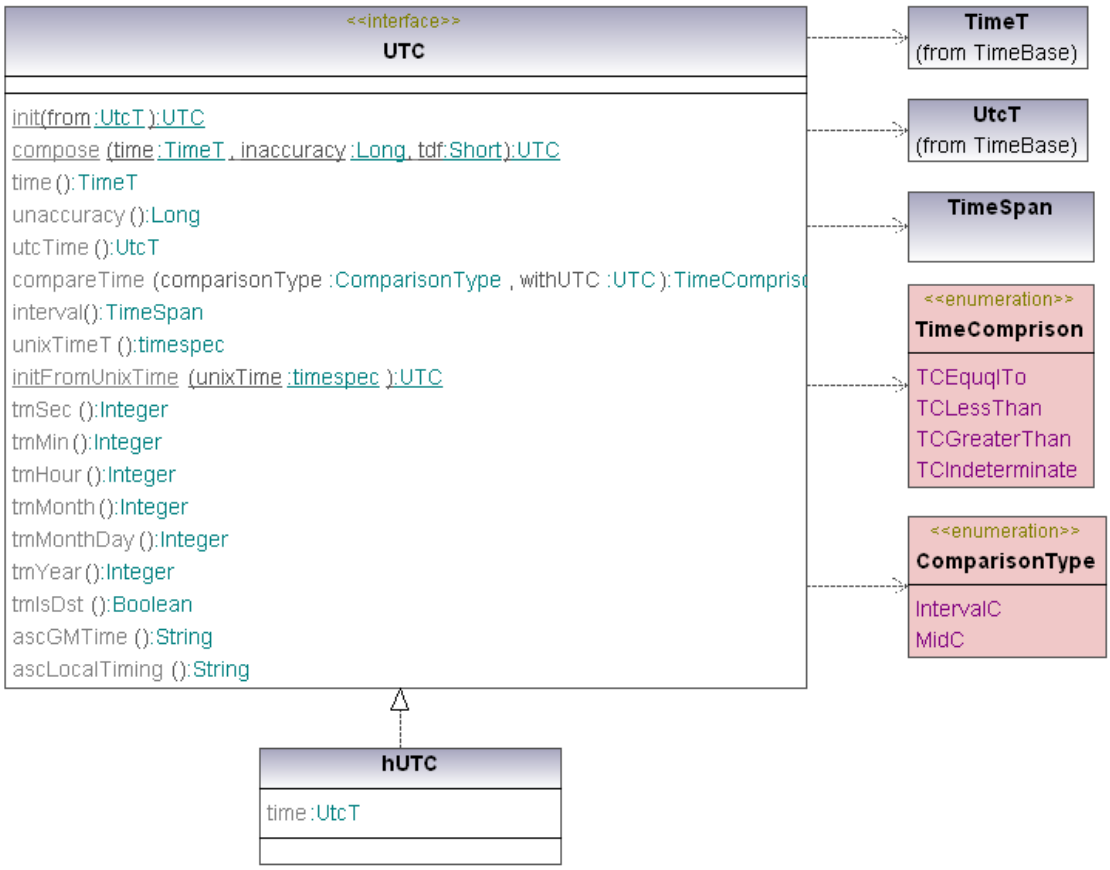

Figura 4.7: Diagrama de clases de la Interfaz UTC

Clase hUTC: Clase concreta envolvente de un instante de tiempo cualificado UtcT. Implementa todas las operaciones definidas en la interfaz UTC, que tienen como objeto interpretar el instante de tiempo cualificado.

Atributos:

time: UtcT => Instante de tiempo que es interpretado por las operaciones de la clase. Se establece en el

constructor y es de sólo lectura.

Operaciones heredadas de la interfaz UTC que implementa:

init(from:UtcT):UTC => Constructor estático que genera un objeto UTC que interpreta el instante de tiempo o el intervalo de tiempo definido por la estructura binaria *from.*

compose(time:TimeT, inaccuracy:Long, tdf:Short): UTC => Constructor estático que genera una instancia UTC a partir de sus campos que se pasan como parámetros.

spans():TimeSpan => Genera una instancia TimeSpan que describe el UTC como un intervalo temporal.

time():TimeT => Retorna el instante de tiempo que corresponde a la instancia UTCT a la que aplica.

unaccuracy():Long => Retorna la inexactitud que corresponde a la instancia UTC a la que se aplica.

utcTime():UtcT => Retorna la estructura binaria que envuelve la instancia UTC a la que se aplica.

compareTime(comparisonType:ComparisonType, withUTC:UTC):
TimeComprison => Compara el instante de tiempo que corresponde a la instancia a la que se aplica con el instante de tiempo que corresponde a la instancia que se pasa como argumento *withUTC*. En la comparación se tiene en consideración la inexactitud con el que se ha medido cada tiempo El parámetro *compatisonType* es enumerado y establece la forma de comparación:

IntervaIC : Realiza la comparación considerando la inexactitud, y puede devolver uno de los cuatro valores.

MidC : Realiza la comparación no usando la inexactitud, y en consecuencia no puede devolver el valor TCIndeterminate.

El resultado de la comparación se describe mediante los valores del tipo enumerado TimeComparison que puede tomar los valores:

TCEqualTo: Ambos tiempos son taxativamente iguales.

TCLessThan: El tiempo de la instancia es con seguridad menor que la del parámetro *withUTC*.

TCGreaterThan: El tiempo de la instancia es con seguridad mayor (posterior) que el del parámetro *withUTC*.

TCIndeterminate: No se puede asegurar si el tiempo de la instancia es menor, mayor o igual que el del parámetro *withUTC*.

interval():TimeSpan => Retorna una instancia TimeSpan que describe el UTC como un intervalo temporal.

unixTimeT():timespec => Retorna el instante de la instancia a la que se aplica como un tiempo compatible con UNIX. Esto es una estructura que contiene dos campos ULong:

sec: número de segundos.

nsec: número de nanosegundos

El instante de referencias de tiempos es el 01-01-1970/00:00:00

initFromUnixTime(unixTime:timespec): UTC => Construye una instancia UTC a partir de un tiempo formulado en formato UNIX.

tmSec():Integer => Segundo del horario que corresponde al instante de la instancia UTC a la que se aplica.

tmMin():Integer=> Minuto del horario que corresponde al instante de la instancia UTC a la que se aplica.

tmHour():Integer=> Hora del horario que corresponde al instante de la instancia UTC a la que se aplica.

tmMonthDay():Integer=> Día del mes del calendario que corresponde al instante de la instancia UTC a la que se aplica.

tmMonth():Integer=> Mes del calendario que corresponde al instante de la instancia UTC a la que se aplica.

tmYear():Integer=> Año del calendario que corresponde al instante de la instancia UTC a la que se aplica.

tmIsDst():Boolean=> Retorna True.

ascLocalTiming():String => Retorna el string que describe el instante descrito por la instancia a la que se aplica, tomando como referencia el origen de hora de la zona de tiempo.

ascGMTime():String=> Retorna el string que describe el instante descrito por la instancia a la que se aplica, tomando como referencia el origen de hora de Greenwich.

La clase TimeSpan envuelve un dato del tipo IntervalT definido en el paquete TimeBase, que representa un intervalo de tiempo delimitado por dos instantes de tiempo y que esta representado por una estructura binaria de 128 bits (16 bytes). La estructura contiene:

- upperBounf:TimeT => formulado como un entero de 64 bits que representa el instante final del intervalo de tiempo.
- lowerBound:TimeT => formulado como un entero de 64 bits que representa el instante inicial del intervalo de tiempo.

En el siguiente diagrama de clases se muestran los elementos de la clase TimeSpan,

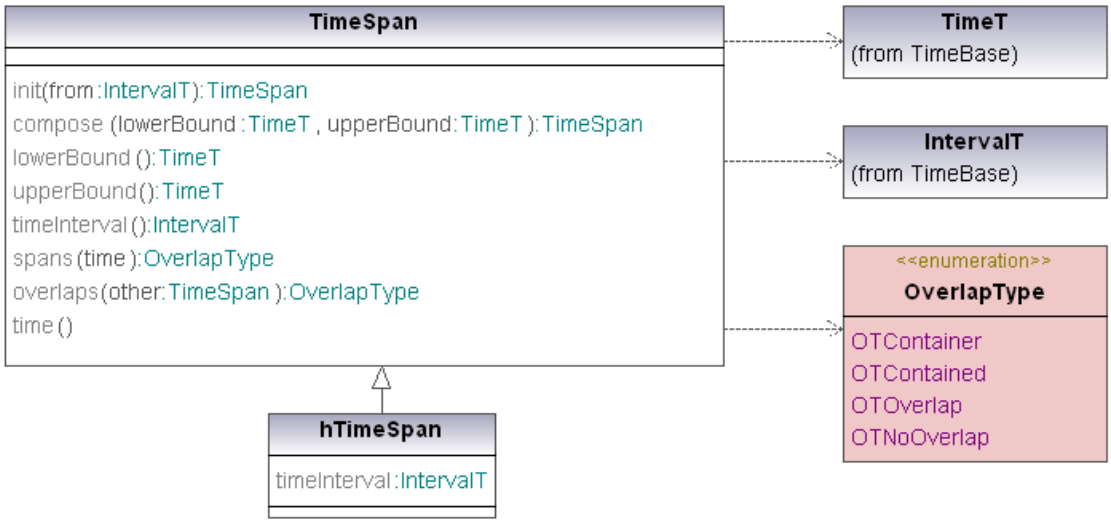

Figura 4.8: Diagrama de clases de la interfaz TimeSpan

Atributos de la clase hTimeSpan:

interval:IntervalT => Estructura binaria que envuelve la clase TimeSpan. Representa un intervalo de tiempo.

Operaciones de la clase hTimeSpan heredadas de la interfaz TimeSpan:

init(from:IntervalT):TimeSpan => Constructor estático que genera una instancia TimeSpan que envuelve la estructura binaria que se pasa como parámetro.

compose(lowerBound:TimeT, upperBound:TimeT):TimeSpan => Constructor estático que genera una instancia TimeSpan cuyos instantes inicial y final se pasa como parámetros.

lowerBound():TimeT => Retorna el instante que constituye el instante inicial del intervalo.

upperBound():TimeT => Retorna el instante que constituye el instante final del intervalo.

timeInterval(): IntervalT => Retorna la estructura binaria que envuelve.

spans(time:UTC):OverlapType => Esta operación devuelve un valor del tipo OverlapType dependiendo de cómo se representan el intervalo en el objeto y el intervalo formulado como el UTC time.

El valor OverlapType que retorna puede tomar los cuatro valores: OTContainer, OT Container, OTOverlap y OTNoOverlap. Su significado se describe en la siguiente gráfica:

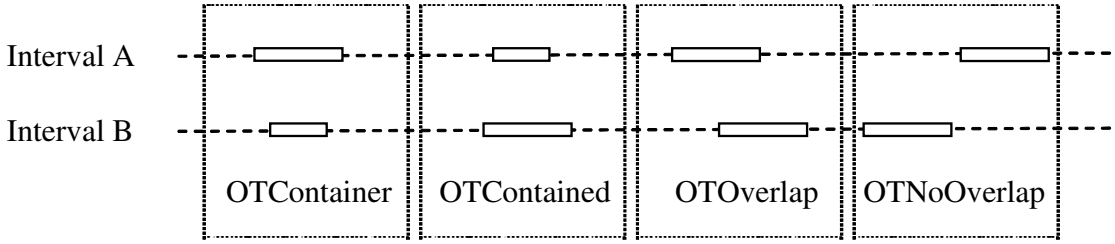

Figura 4.9: Tipos de intervalos

overlaps(other:TimeSpan): OverlapType =>=> Esta operación devuelve un valor del tipo OverlapType dependiendo de cómo solapan el intervalo del objeto y el intervalo del parámetro *overlap*.

time():UTC => Retorna formulado como una instancia UTC, el intervalo de tiempo que corresponde al intervalo que se pasa como parámetro.

4.6 Ejemplo de acceso local y remoto al UtcTimeService

El objetivo de esta sección es mostrar un conjunto de ejemplos típicos de utilización del TimeService sobre una plataforma distribuida. El objetivo no es hacer una catalogación exhaustiva, sino plantear unos ejemplos sencillos que muestre el sencillez del uso del servicio.

Datación de eventos singulares

En este ejemplo, un cliente utiliza el TimeService para datar eventos que esporádicamente necesita registrar, con el objeto de que en el futuro puedan compararse la precedencia temporal con otros eventos que se han registrado en otros nudos de la plataforma distribuida.

En el siguiente diagrama de secuencia se muestra los sucesivos pasos que debe ejecutar el cliente:

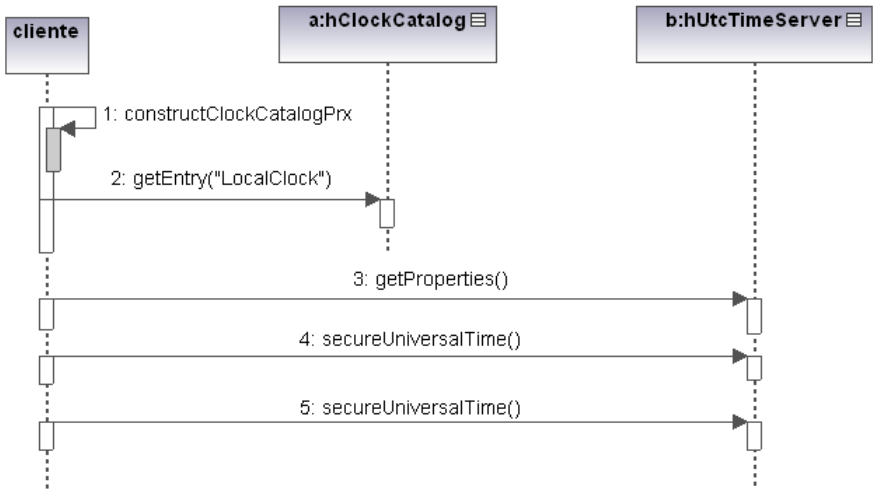

Figura 4.10: Datación de eventos singulares

1. El cliente accede a un TimeService de la plataforma. El acceso se realiza accediendo a la interfaz *ClockCatalog*, que esta bien definida una vez que se conozca el nudo en que se encuentra instalado el TimeService al que se accede. Lo óptimo es que el cliente acceda al TimeService local, si este existe. Para acceder a la interfaz *ClockCatalog* construye u obtiene el Proxy de esta interfaz.

2. El cliente invoca la operación *getEntry()* con la clave “LocalClock” para obtener el Proxy de la interfaz *UtcTimeService*, que ofrece el servicio para datar eventos.
3. Si es necesario, el cliente invoca la operación *getProperties()* para evaluar si las características del TimeService son las que necesita para la funcionalidad que tiene asignada.

4.5.. Una vez que tiene el acceso a la interfaz *UtcTimeService* el cliente puede datar los eventos invocando las operaciones *secureUniversalTime()* o *universalTime()*. Ambos retornan una estructura binaria *UtcT* que contiene tanto el tiempo de datación como las características de inexactitud que son necesaria para comparar el tiempo obtenido con otros tiempos obtenidos en otros nodos del sistema distribuido. La diferencia entre ambos métodos es que la operación *secureUniversalTime()* lanza una excepción si el nivel de sincronización del TimeService no satisface los niveles de exactitud que se requieren.

Este tipo de datación, incorpora en cada dato de tiempo una estructura UtcT de 16 bytes o 128 bits. El análisis posterior los datos de tiempo binario se realiza utilizando la clase envolvente UTC.

Datación de secuencias de eventos

En este ejemplo el cliente necesita datar una larga secuencia de eventos, y quiere optimizar la cantidad de información que utiliza para almacenar los datos de tiempo de esos eventos. Para ello, va utilizar como dato de tiempo una estructura binaria *TimeT* que requiere sólo 8bytes (64 bits) (frente a los 16 bytes que requería la estructura *UtcT* del ejemplo anterior).

Este ahorro de información implica no incluir la información de inexactitud en cada dato de tiempo. Sin embargo, esta falta puede ser suplida asociando al primer tiempo datado o uno de cada cierto tiempo la información *UtcT* completa. Con esto, no se pierde capacidad de comparación bajo las dos situaciones siguientes:

- Los tiempos de la secuencia difieren entre ellos menos del doble del tiempo de sincronización. En este caso el tiempo de inexactitud y tiempo de zona permanece constante para todo ellas, y las características asociada a la primera es extrapolable a las demás.
- Los tiempos de la secuencia de eventos sólo se van a comparar con eventos del mismo TimeService. En este caso, la monotonicidad del reloj en que se basa el TimeService garantiza la comparación de los tiempos con independencia de las características de inexactitud.

En el siguiente diagrama de secuencia se muestra los pasos que debe seguir el cliente:

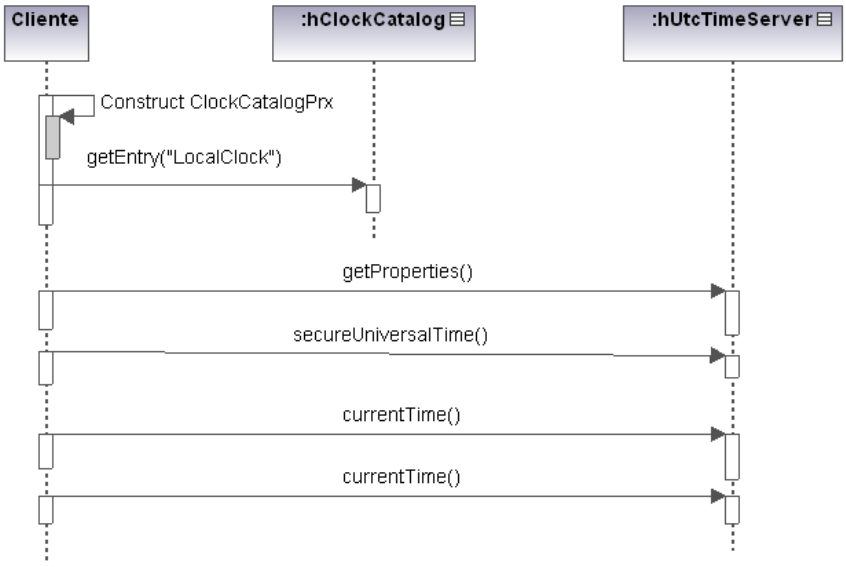

Figura 4.11: Datación de secuencias de eventos

Esta secuencia es semejante a la que se ha explicado en el ejemplo anterior, salvo que la datación de los eventos se realiza utilizando una primera datación haciendo uso de la operación *universalTime()* que incorpora la información de inexactitud de los tiempos, y las restantes dataciones se realizan utilizando la operación *currentTime().*
La comparación de los datos temporales se debe realizar extrapolando la información de inexactitud del primer dato a los restantes. Esto se muestra en el ejemplo del apartado 5.3.

5.Implementación de hControlledClock sobre ICE

5.1 Estrategia de implementación de los relojes de misión

La clase hControlledClock implementa relojes especiales, que permiten medir el tiempo a partir de un determinado evento, puede ser el tiempo escalado y controlado su flujo (detenido, reanudado, reseteado, etc.). Un hControlledClock implementa las interfaces definidas en el estándar Clock y ControlledClock.

Cada vez que el cliente necesita un ControlledClock, puede requerirlo a través de ClockCatalog utilizando la clave “ControlledClock”. El TimeService crea un nuevo ControlledClock, y proporciona un Proxy para su gestión. Si un mismo ControlledClock debe ser utilizado por diferentes clientes, el cliente que lo crea puede proporcionar el acceso a los otros, bien directamente, o a través del ClockCatalog registrando su acceso con una clave conocida por todos los clientes.

En el siguiente diagrama de clases se muestra los elementos básicos de un hControlledClock:

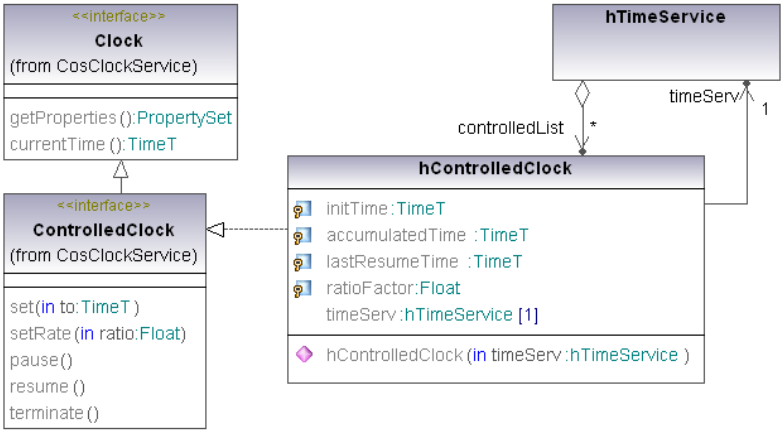

Figura 5.1: Diagrama de clases de un reloj de misión

Atributos de la clase:

initTime: TimeT => Tiempo inicial que utiliza para referencia de los tiempos que mide.
accumuledTime: TimeT => Tiempo acumulado del reloj en unidades físicas. Corresponde al tiempo medido hasta que se realizó la última parada (se invocó pause() por última vez).
lastResumeTime: TimeT => Si el reloj está corriendo es el tiempo en el que comenzó la actual medida de tiempo (ultima vez que se invocó resume).
isRunning : Flota => Es True si el reloj está contando tiempo, es False si el reloj está parado.
ratioFactor: Float => Razón entre el tiempo que cuenta y el tiempo físico que transcurre.
timeServ: hTimeService => Referencia al TimeService en el que es creado.

Las operaciones que ofrece un ControlledClock son:

Heredadas de la interfaz Clock:

currentTime*() : TimeT =>* Devuelve el tiempo actual medido por el reloj.

getProperties*(): PropertySet =>* Retorna las propiedades declaradas del reloj. Cada ControlledClock introduce nuevas propiedades que son específicas:

“InitTime”:”<dd:mm:yy/hh:mm:ss:ms>” => Retorna el instante de referencia del reloj controlado.

“IsRunning”:”<Bolean>” => Retorna “True” si el reloj está contando. Retorna “False” si el reloj está parado.

"RatioFactor":"<Float>" => Retorna el factor escalado del tiempo que tiene establecido el reloj. Por ejemplo:

- *"1.0" si mide tiempo físico.*
- *"-1.0" si cuenta tiempo físico hacia atrás.*
- *"2.5"por cada segundo físico que transcurre cuenta 2.5 s.*

"Clave": String => Clave con el que se puede localizar este ControlledClock en el ClockCatalog. Retorna "" (String vacío) si el controlledClock no está registrado.

Heredadas de la interfaz ControlledClock:

***set**(To:TimeT) =>* Establece la hora actual mantenida por el reloj a un valor especificado.

***setRate**(ratio: Float) =>* Permite acelerar (ratio >1), ralentizar(0<ratio<1) o correr hacia atrás (ratio<0) al reloj. El parámetro ratio es la relación entre el tiempo con que mide el reloj y el tiempo real que ha transcurrido. Por ejemplo si ratio=2.0, cuando el reloj mide un intervalo de 10 segundos, es porque la duración real del intervalo fue de 5.0 segundos.

***pause**() =>* Para aparentemente el transcurrir del tiempo.

***resume**() =>* Reanuda el transcurrir del tiempo.

***terminate**() =>* Para definitivamente el reloj.

Constructor:

hControlledClock (timeServ: hTimeService) => Recibe como parámetro la referencia del hTimeService en el que es creado.

5.2 Estructura de datos de un reloj de misión

El estado de un reloj de misión queda definido por los valores de sus atributos y por el valor del reloj UTC del TimeService. En la figura 5.2, se muestran los valores de los atributos para dos instantes t_a (con reloj parado) y t_b (con reloj contando)

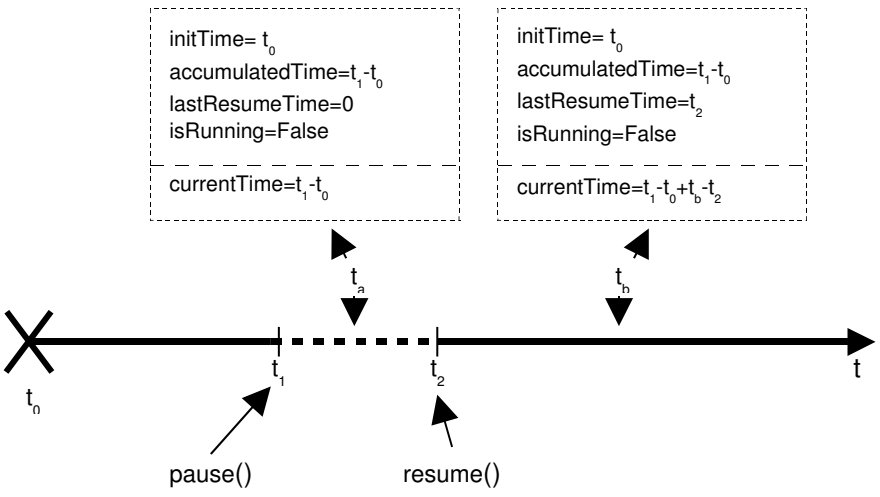

Figura 5.2: Valores de los atributos de un ControlledClock

El valor actual del reloj puede evaluarse de acuerdo con las siguientes funciones:

- Si el reloj está parado (isRunning=True):
  $$currentTime = accumulatedTime \times ratioFactor$$

- Si el reloj esta contando (isRunning=False):
  $$currentTime = (accumulatedTime + time - lastResume) \times ratioFactor$$

Para mantener la consistencia de los atributos del reloj controlado, deben ser actualizado de acuerdo con las siguientes reglas:

- Si se invoca setTime(To: TimeT)
  - accumulatedTime=to;
  - lastResume= time;
  - isRunning=isRunning;

- Si se invoca pause()
  - accumulatedTime= accumulatedTime + time - lastResume;
  - lastResume= 0;
  - isRunning=False;
- Si se invoca resume()
  - accumulatedTime= accumulatedTime;
  - lastResume= time;
  - isRunning=True

siendo time el tiempo que se mide en el reloj UTC del TimeService.

5.3 Ejemplo de uso, acceso local y remoto a los relojes de misión

Generación y control de un reloj controlado

En este ejemplo un cliente genera un reloj de misión, estableciendo para él el instante de tiempo que se considera como referencia, así como su escala de tiempo. Posteriormente data eventos con él, y lo controla, parando y reanudando la cuenta del tiempo según proceda.

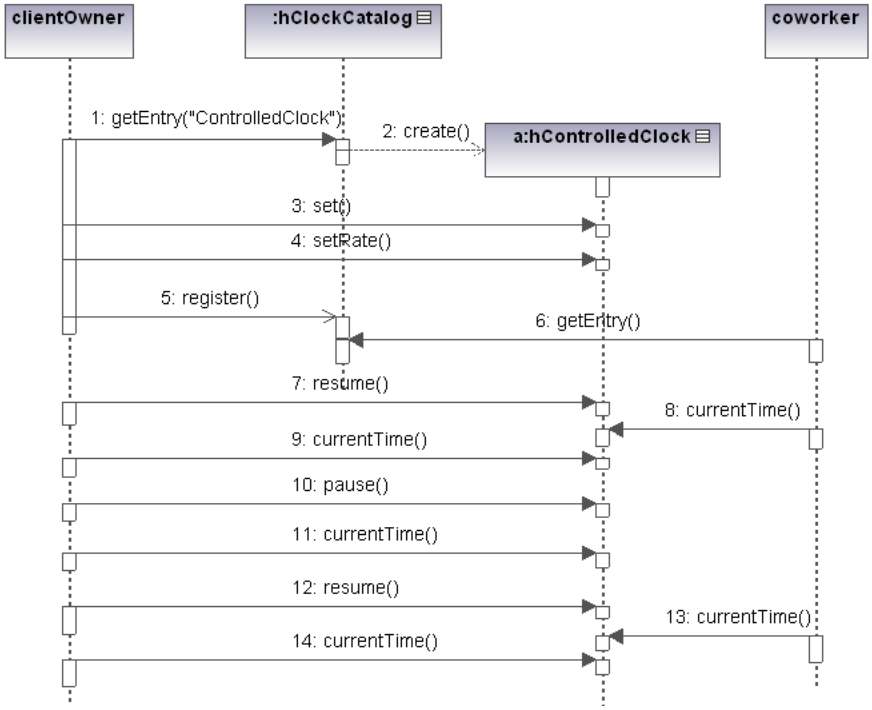

Figura 5.3: Generación y control de un reloj controlado

1. El cliente propietario requiere un reloj controlado a través del *ClockCatalog* utilizando la operación *getEntry()* con la clave “ControlledClock”.
2. El TimeService crea internamente el reloj controlado y retorna al cliente el Proxy para acceder a él.
3. El cliente clientOwner utiliza la operación *setTime()* para inicializar el ControlledClock estableciendo el instante temporal que se toma como referencia.
4. El cliente clientOwner haciendo uso de la operación *setRate()* establece en el ControlledClock la escala de tiempo que debe utilizar.
5. El cliente propietario registra en el ClockCatalog con la operación *register()* para que otros clientes puedan acceder él.
6. El cliente coworker obtiene con la operación *getEntry()* del ClockCatalog el Proxy del *ControlledClock* creado por clientOwner. Para ello debe conoce y utilizar la clave con la que clientOwner reistro el ControlledClock.

7.,10.,12, Un cliente controla el ControlledClock invocando la operación *pause()* para parar la cuenta del tiempo, y la operación *resume()* para reanudarla.

8.,9.,11.,13.,14. Los clientes datan eventos haciendo uso del ControlledClock, haciendo uso del método *currentTime()*.

6.Implementación de hExecutor sobre ICE

6.1 Estrategia de implementación de la ejecución periódica y aplazada

Las clases hExecutor y hController son la base del servicio que ofrece el TimeService para ejecutar tareas periódicas. En cada TimeService que ofrezca este servicio hay un único objeto de la clase hExecutor que ofrece la interfaz estándar *Executor*, a través de la que los clientes pueden requerir la ejecución de una tarea periódica o retrasada. Como respuesta a un requerimiento, el TimeService construye los recursos necesarios para gestionar esta invocación periódica. Por cada nueva tarea que tiene que ser invocada, el TimeService crea un objeto de la clase hController para su gestión. Los objetos de esta clase implementan la interfaz estándar *Controller*, y a través de sus funciones el cliente que ha requerido la invocación de una tarea, puede controlar el ciclo de vida de la ejecución. El acceso a la interfaz del objeto *Controller* se obtiene de *Executor* al requerir la invocación periódica o retrasada. La tarea que puede requerirse ser ejecutada debe consistir en la ejecución del método *doWork*(), de un objeto proporcionado por el cliente que debe ser una instancia de una clase que implementa la interfaz *Periodic* definida en el estándar.
En el siguiente diagrama de clases se describen los elementos que implementan el servicio de invocación periódica.

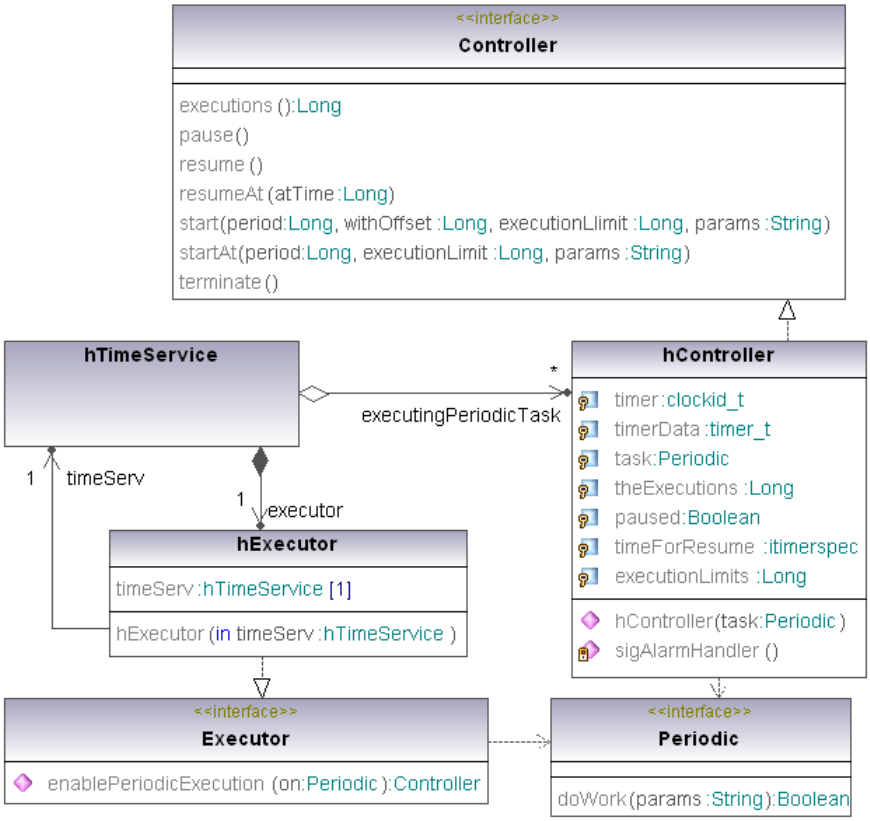

Figura 6.1: Diagrama de clases del servicio de invocación periódica

6.2 Temporizadores en Linux

Linux ofrece un conjunto de funciones para crear y gestionar temprizadores (*timers*) que generen periódicamente la señal SIGALRM [19], que a través de un manejador puede convertirse en la ejecución de tareas periodicas. Este conjunto es:

timer_create(`clockid_t *clockid, struct sigevent *evp, timer_t *timerid)=>` Esta función crea un timer por cada proceso utilizando el reloj especificado en el parámetro *clockid*. Devuelve en el argumento *timerid* el identificador del timer, del tipo timer_t utilizado para identificar el *timer*. El identificador del timer será único en el proceso que lo llama hasta que el *timer* se destruye. El reloj particular *clock_id* está definido en <time.h>. El timer cuyo identificador se retorna no estará en un estado operativo hasta que la función *timer_create* termina. El parámetro evp, en el caso de ser *NULL*, apunta a una estructura con el atributo *sigev_notify = SIGEV_SIGNAL*, el atributo *sigev_signo* tendrá el numero de señal por defecto y el atributo *sigev_value* tendrá el valor del identificador del timer. En el caso de que el parámetro *evp* sea distinto de *NULL*, apuntará a una estructura *sigevent* que guarde los valores de todos los atributos anteriormente mencionados.

timer_settime (`timer_t timerid, int flags, const struct itimerspec *value, struct itimerspec *ovalue)=>` La función timer_settime() establece el tiempo hasta la siguiente expiración del *timer* especificado en *timerid.* La temporización puede ser relativa o absoluta, según el valor de *flags.* El funcionamiento se repite periódicamente si *value.it_period* > 0. En *ovalue se devuelve el valor que quedaba de la temporización anterior.

`sigemptyset (sigset_t *set)` y `sigaddset (sigset_t *set, int sig_num) =>` Permiten inicializar un conjunto de señales vacío y añadir una señal al conjunto. Los conjuntos se utilizan para manipular la máscara de señales de un proceso (sigprocmask) y para esperar señales (sigsuspend y sigwaitinfo).

sigwaitinfo(`const sigset_t *set, siginfo_t *info`) => La función *sigwaitinfo* selecciona la señal a partir de un set de señales predeterminadas. Si no existe ninguna señal pendiente en el momento de la llamada, el thread que lo invoca se queda suspendido hasta exista una señal pendiente o hasta que sea interrumpido por una señal desbloqueante.

timer_delete (`timer_t timerid`) => La función timer_delete borra el timer especificado por *timerid*. Este timer ha de haber sido creado previamente por la función timer_create.

Los métodos y atributos definidos para el timer creado, vienen descritos en la siguiente figura.

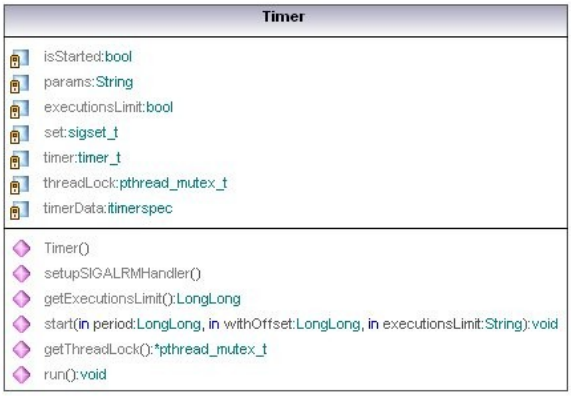

Figura 6.2: timer class

6.3 hController: Controlador local de la ejecución

Clase **hController** : Es el objeto que crea el TimeService para que el cliente que requiere una ejecución periódica pueda establecer las propiedades de la ejecución periódica y controlar su el ciclo de vida de la ejecución.

Atributos:

timer: clockid_t => Timer Linux que controla la ejecución periódica.

timerData: timer_t => Dato temporal que describe el estado del timer Linux.

task: Periodic => Referencia al objeto que implementa la interface Periodic, y que tiene definida la operación doWork(params:String) que debe ser ejecutada.

theExecutions: Long => Número de ejecuciones que ya ha sido ejecutada.

paused:Boolean => Retorna True si la ejecución periódica ha sido suspendida.

timeForResume: itimerspec => Contiene el instante de tiempo en el que se debe iniciar la primera ejecución de la tarea.

executionLimits: Long => Número de veces que debe ejecutarse la tarea. Cuando se alcanza este valor, la ejecución periódica concluye y el controlador se destruye.

Operaciones:

hController(task:Periodic) => Constructor de la clase. Recibe como parámetro la referencia al objeto que implementa la interfaz Periodic que debe ser ejecutada.

sigAlarmHandler() => Manejador de la señal SIGALARM que periódicamente es generada por el timer. Esta tarea es la que de acuerdo con el estado del controlador ejecuta la tarea requerida.

6.4 hExecutor: Planificador de ejecuciones periódicas y aplazadas

Clase **hExecutor** : Describe el objeto que ofrece TimeService para que los clientes requieran del servicio la ejecución de una tarea para su ejecución periódica o retrasada.

Atributos:
timeServ:hTimeService => Referencia al TimeService en el que está creado.

Operaciones:

enablePeriodicExecution(on:Periodic): Controller => Es invocada por el cliente para requerir que el TimeService cree un controlador de ejecución periódica o retrasada que la función retorna para que a través de él, el cliente defina las características de ejecución y controle su ciclo de vida. La tarea periódica consiste en la ejecución de la operación *doWork(params:String)* que debe implementar el objeto que se pasa como el parámetro *on*.

6.5 Ejemplo de uso local y remoto de ejecución periódica y aplazada de tareas

Ejecución periódica de una operación remota.

Un cliente requiere del TimeService que ejecute periódicamente una cierta actividad haciendo uso del servicio de ejecución periódica de tareas.

En el siguiente diagrama de secuencias se muestra los pasos que deben seguirse:

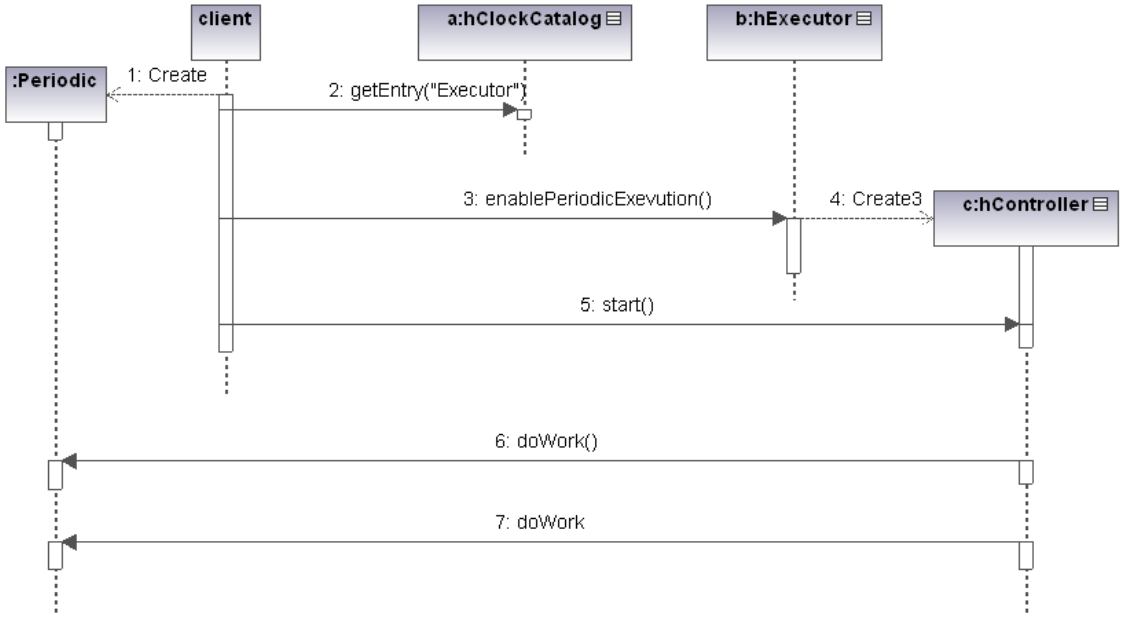

Figura 6.3: Ejecución periódica de una ejecución remota

1. El cliente crea o accede la instancia auxiliar que contiene la actividad *doWork()* que se quiere ejecutar periódicamente.
2. El cliente accede al ClockCatalog que utilizando la operación *getEntry()* con la clave “Ejecutor” que le retorna el Proxy a la interfaz *Executor* del TimeService.
3. El cliente requiere del Executor con la operación *enablePeriodicExecution()* que se cree un controlador destinado a la ejecución de la actividad definida en el objeto que implementa la interfaz *Periodic* y que se pasa en el argumento.
4. El Executor del TimeService crea el controlador que va a gestionar la ejecución periódica de la actividad.
5. El cliente accede al controlador y con la operación Start() ordena el inicio de la operación periódica en el instante y con el periodo que se requiera.

6.,7. El controlador ejecuta la tarea doWork() sobre la instancia que implementa la interfaz *Periodic* a partir del instante y con el periodo establecido.

7.Conclusiones y líneas futuras

En este trabajo se ha diseñado e implementado un servicio distribuido de tiempo que proporciona un tiempo global, fiable y comparable a los procesos, componentes y aplicaciones que se ejecutan en una plataforma distribuida heterogénea y que se comunican a través del middleware ICE. Características relevantes del servicio de tiempos desarrollado son:

- El servicio de tiempos desarrollado ofrece los servicios definidos en la especificación "Enhanced View of Time Specification" de la organización Object Management Group (formal/04-10-04). De acuerdo con ella, el servicio ofrece:
    - Acceso desde cualquier nudo a un tiempo UTC cualificado que permite una comparación fiable en cuanto a su ordenación temporal.
    - Permite la creación de relojes especiales (*controlled clock*). El cliente puede controlar su funcionamiento (parar, reanudar, inicializar) y asi mismo escalar su tasa de avance.
    - Ofrece un servicio de ejecución temporizada de tareas. A través de ella el servicio de tiempo puede ejecutar con referencia a cualquier reloj de la plataforma distribuida y en cualquier nudo de la plataforma actividades periódicas o retrasadas.
- Al servicio se puede acceder a través de interfaces ICE definidas en el lenguaje Slice, que es independiente de cualquier lenguaje de programación. Por tanto puede ser accedido desde los lenguajes Java, C/C++, C#, VisualBasic, Python y PhP.
- El servicio de tiempo se basa en un conjunto de servicios de tiempo locales que pueden ser instalados individualmente por el operador desde la consola, o remotamente desde una aplicación de gestión de los servicios de tiempo.
- Cada servicio de tiempo local mide el tiempo en base al reloj del sistema del procesador en que está instalado. El tiempo global único se consigue mediante la sincronización a través de NTP de los procesadores de los nudos dotados de servicio de tiempo local.
- El servicio de tiempo local gestiona la sincronización del reloj del sistema propio, siendo responsable de definir el servidor de tiempos con el que se sincroniza, asi como de extraer información estadística sobre el nivel de sincronización que se necesita para cualificar los tiempos con respecto a dicho servidor.

El servicio de tiempo está implementado, pero requiere aún ser analizado en cuanto a su precisión temporal. Existen varias causas que introducen incertidumbre en la medida de tiempos:

- Cuando se accede al servicio de tiempos local desde procesos que se ejecutan en el propio procesador, el acceso se realiza a través de un puerto ICE, lo cual introduce un retraso que debe ser cualificado. Dado que la base de la medida de tiempo es el propio reloj de sistema, si el retraso de acceso es excesivo, cabe la estrategia de acceder directamente al reloj del sistema desde el proceso cliente y luego cualificar el tiempo en el servicio, a fin de compararlo con tiempos obtenidos en otros nudos de la plataforma.
- Cuando se accede a un servicio de tiempo de un procesador desde un proceso que se ejecuta en otro procesador, lo cual está justificado porque en este no haya instalado un servicio de tiempos propio, el tiempo de acceso remoto al servicio de tiempos introduce incertidumbre en el tiempo que se obtiene. En este caso, el tráfico en la red influye en el nivel de incertidumbre.
- La sincronización por NTP de la red introduce una incertidumbre entre los tiempos medidos en diferentes procesadores. Esta incertidumbre se evalúa a partir de la información que proporciona la aplicación NTP, y se tiene en cuenta en la cualificación de los tiempos. En cualquier caso, es conveniente que sea validada.

Ya se han realizado algunas medidas basadas en la observación de un evento externo desde diferentes nudos, no obstante, se ha comprobado que el driver utilizado para detectar el evento externo introduce una incertidumbre del mismo orden de incertidumbre de los tiempos que se validan, por lo que no proporcionan una información completa. Un paso previo que se requiere es el desarrollo de un driver adaptado eficiente a este fin.

8.Referencias

[1] Andrew S. Tanenbaum, Maarten van Steen: *Distributed Systems. Principles and Paradigms*. 2002. Ed. Prentice Hall.

[2] Proyecto HESPERIA: **H**omeland s**E**curity: tecnología**S** **P**ara la Seguridad integ**R**al en espac**I**os públicos e infr**A**estructuras. Proyecto CENIT- 2005. https://www.proyecto-hesperia.org/.

[3] Flavin Cristian and Christof Fetzer. Probabilistic Internal Clock Synchronization. www.christof.org 1 May 2003.

[4] Thomas Gleixner (linutronix) Douglas Niehaus (University of Kansas): *Hrtimers and Beyond: Transforming the Linux Time Subsystems*. Proceedings of the Linux Symposium July 19th–22nd, 2006 Ottawa, Ontario Canada

[5] John Stultz, Nishanth Aravamudan, Darren Hart (IBM Linux Technoly Center): *We are not Getting Any Younger: A New Aproach to Time and Timers*. http://www.linuxsymposium.org/2005/linuxsymposium_procv1.pdf

[6] http://support.ntp.org/

[7] Michi Henning Mark Spruiell: *Distributed Programming with Ice*. May 2008. ZeroC. http://www.zeroc.com./

[8] *Supported Platforms for Ice 3.3*. ZeroC. http://www.zeroc.com./platforms.html

[9] Object Management Group (OMG): *Time Service Specification*. Version 1.1. May 2002.

[10] Object Management Group: *Enhanced View of Time Specification*. Version 1.2 formal/04-10-04.

[11] http://www.omg.org/

[12] http://www.linux.org/

[13] http://java.sun.com/

[14] http://standards.iso.org/

[15] http://support.microsoft.com/

[16] David Deeths: *Using NTP to control and Synchronize System Clocks Part I, II & III.* Sun BluePrintsTM OnLine – 2001

[17] David L. Mills, University of Delaware RFC1305 - *Network Time Protocol (Version 3) Specification*. Network Working Group

[18] D. Mills University of Delaware: RFC2030 - *Simple Network Time Protocol (SNTP) Version 4 for Ipv*. Network Working Group.

[19] Unix System programming communication, concurrency and threads. KAY A. Robbins, Steven Robbins. de Prentice hall 2003.

Apéndice A:

Declaración Slice de las Interfaces del servicio de tiempo.

Un aspecto importante del servicio de tiempo que se ha diseñado es la definición de sus interfaces de acceso. En la especificación de estas interfaces en la especificación de OMG están formuladas en lenguaje IDL, en este trabajo se han realizado su especificación en Slice que es el lenguaje de especificación de interfaces que se utiliza en ICE.

La transformación se ha buscado que sea lo mas próxima posible, pero al tener los dos lenguajes IDL y Slice diferentes tipos de datos, y en particular, la no existencia del tipo Any ni la capacidad de redefinir tipos en Slice, ha requerido establecer diferencias significativas entre ambas declaraciones.

A continuación se describen la declaración de las interfaces del servicio de tiempo en lenguaje Slice.

A.1 Especificación del módulo TimeBase.

```
//**************************************************************************************************
//            Grupo de Computadores y Tiempo Real – Universidad de Cantabria
//
//   Especificacion Slice del módulo TimeBase utilizado en el servicio TimeService
//   implementado en plataforma LINUX y sobre ICE
//
//    Authors: Angela del Barrio y José M. Drake
//    Version: 30-05-08
//**************************************************************************************************
module TimeBase {

   // TimeT      => Definicion Slice:  long;
   //TdfT        => Definicion Slice: short;
   //Inacuracy  => Definicion Slice:  long

   struct UtcT {
         long     time;              // (8 octets)
         long     inactdf;           // (8 octets)
         //long    inacclo: (4 octects) Included in inactdf
         //short   inacchi; (2 octets)   Included in inactdf
         //short   tdf;      (2 octets)   Included in inactdf
    };                                 // total 16 octets

   struct IntervalT {
         long lowerBound;
         long upperBound;
   };
};
```

A.2 Especificación del módulo CosClockService.

```
//**********************************************************************************************************
//            Grupo de Computadores y Tiempo Real – Universidad de Cantabria
//
//    Especificacion Slice del módulo CosClockService utilizado en el servicio TimeService
//    implementado en plataforma LINUX y sobre ICE
//
//     Authors: Angela del Barrio y José M. Drake
//     Version: 10-09-08
//**********************************************************************************************************

//  All interfaces, and associated enum and exceptions are placed in the
//  CosTime module.

#include "TimeBase.ice"
module CosClockService {

     interface Clock;
     exception TimeUnavailable{};
          // PropertyName is a string
          // PropertyValue is represented as a string
          // dictionary<PropertyName, PropertyValue>

     dictionary<string, string> propertySet;

      //basic clock interface
      interface Clock  {          //a source of time readings
           nonmutating propertySet getProperties();
           nonmutating long getCurrentTime() throws TimeUnavailable;
       };
};

module CosClockService {

     interface UtcTimeService extends Clock {
          TimeBase::UtcT universalTime() throws TimeUnavailable;
          TimeBase::UtcT secureUniversalTime() throws TimeUnavailable;
          TimeBase::UtcT absoluteTime(TimeBase::UtcT withOffset) throws TimeUnavailable;
     };
};

module CosClockService {

    dictionary<string, Clock*> ClockEntries;
    exception UnknownEntry{};

    interface ClockCatalog {
          Clock* getEntry(string withName) throws UnknownEntry;
          ClockEntries availableEntries();
          void registerClock(string name, Clock* subject);
          void deleteEntry(string withName) throws UnknownEntry;
    };
};

module CosClockService {
```

```
    exception NotSupported{};

    interface ControlledClock extends Clock {
        void set(long to) throws NotSupported;
        void setRate(float ratio) throws NotSupported;
        void pause() throws NotSupported;
        void resume() throws NotSupported;
        void terminate() throws NotSupported;
     };
};

module CosClockService {

    module PeriodicExecution {

        interface Periodic {
            bool doWork (string params);
        };

        exception TimePast{};

        interface Controller {
            void start (long period, long withOffset, long executionLimit, string params);
            void startAt(long period, long atTime, long executionLimit, string params)
                                                                        throws TimePast;
            void pause();
            void resume();
            void resumeAt(long atTime) throws TimePast;
            void terminate();
            long executions();
        };

        interface Executor extends Clock {
            Controller* enablePeriodicExecution(Periodic* on);
        };
    };
};
```

A.3 Especificación del módulo TimeService.

```
//**************************************************************************************************************
//            Grupo de Computadores y Tiempo Real – Universidad de Cantabria
//
//   Especificacion Slice del módulo TimeService utilizado en el servicio TimeService
//   implementado en plataforma LINUX y sobre ICE
//
//    Authors: Angela del Barrio y José M. Drake
//    Version: 10-09-08
//**************************************************************************************************************
#include "CosClockService.ice"

module TimeService {

    interface timeServiceManager {
        bool isAlive();
        void resynch(string configData);
        nonmutating CosClockService::propertySet getProperties();
    };
};
```