03063

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

POSGRADO EN CIENCIA E INGENIERIA DE LA COMPUTACION

"PROPUESTA DE UN ALGORITMO GENETICO PARA LA RECONFIGURACION EN LINEA DE UN SISTEMA DE TIEMPO REAL BASADO EN RT-CORBA"

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN INGENIERIA

(COMPUTACION)

P R E S E N T A:

HONORATO SAAVEDRA HERNANDEZ

DIRECTOR DE TESIS:

DR. HECTOR BENITEZ PEREZ

MEXICO, D. F. 2005

M346716

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

Autorizo a la Dirección General de Bibliotecas de la UNAM a difundir en formato electrónico e impreso el contenido de mi trabajo recepcional.
NOMBRE: Honorata Saavedra Hernández
FECHA: 10 de agosto de 2005
FIRMA:

Dedico este trabajo a mi familia, especialmente a
David y Josefina por ser mi esperanza y apoyo.

Agradezco la infinita paciencia de Héctor y
la ayuda de todos los que en la UNAM
hacen posible que esto suceda.

Índice de contenido

1. Introducción

Los sistemas de cómputo en general y los sistemas distribuidos de tiempo real en particular, se han vuelto cada vez más complejos y difíciles de mantener y modificar. Se han logrado avances importantes con respecto al hardware creando sistemas más económicos, pequeños y confiables, mientras que el desarrollo de software sigue siendo una actividad en la que aún no se ha adoptado un proceso bien definido que garantice una producción rápida y ordenada para crear sistemas confiables, seguros y fáciles de manejar. Una de las tendencias actuales para mejorar esta problemática, es el uso de sistemas de apoyo o intermediarios (middleware) sobre los cuales se construyen aplicaciones que utilizan servicios o herramientas de esta capa intermedia de software para simplificar tareas comunes evitando que se reinventen las mismas funciones básicas desde cero para cada proyecto.

Los sistemas distribuidos de tiempo real están presentes en muchas actividades humanas y debido a su importancia se han creado diversas plataformas y herramientas para ayudar a diseñar y construir este tipo de programas. Una de estas plataformas es la especificación de CORBA (OMG, 2001), (Vinoski, 1997) de tiempo real (RT-CORBA) (OMG, 1999), la cual describe la arquitectura básica necesaria sobre la que se hace la construcción de sistemas que cumplirán con restricciones de tiempo al momento de generar resultados utilizando CORBA. Una de las instanciaciones más importantes de esta especificación es TAO (The ACE ORB) (Schmidt, et al., 1998), el cual es una instanciación de CORBA con extensiones para el manejo de restricciones de tiempo real. Es precisamente esta plataforma la que se utiliza para construir la aplicación sobre la que trata el presente trabajo relacionado con la reconfiguración de sistemas distribuidos de tiempo real.

En los sistemas distribuidos de tiempo real es importante la manera en la que se organiza la atención de las distintas tareas a ejecutar para garantizar sus términos. Al construir sobre TAO, es posible elegir uno de varios métodos de planificación disponibles y lograr este objetivo de la mejor forma posible, dependiendo de la naturaleza de la aplicación. Para sistemas dinámicos en los cuales existen múltiples variables y donde se necesita aprovechar al máximo el tiempo del procesador, es recomendable utilizar un método dinámico o semi-dinámico de planificación como pueden ser "primero el de límite más cercano" (EDF, Earliest Deadline First), "primero el de laxidad mínima" (MLF, Minimum Laxity First) o "primero el de máxima urgencia" (MUF, Maximum Urgency First) (Levine, et al., 1998) de manera que el sistema produzca los mejores resultados. Además de esta estrategia de planificación, en un sistema distribuido se deben tomar en cuenta los problemas relacionados con la comunicación y las posibles fallas de los elementos, así como los cambios en el ambiente sobre el cual opera la aplicación.

Debido a la necesidad de que los sistemas de tiempo real sean confiables y adaptables, se les debe dotar con la capacidad de cambiar la manera en la que atienden las tareas dependiendo del estado actual de sus componentes. Como la planificación de tiempo real en un sistema con un número grande de tareas es un problema abierto debido al tamaño de su espacio de solución, es factible obtener cambios imprevistos en el comportamiento dinámico, así como enfrentar restricciones y preferencias según el propósito específico de cada sistema. Este trabajo propone la utilización de algoritmos genéticos (AGs)

(Kuri, et al., 2001) para encontrar la solución (L. Karr, et al., 1999).

El problema que se desea resolver es la reconfiguración de un sistema de tiempo real utilizando herramientas de fácil adquisición o de uso común para mejorar su desempeño. Para lograr esto se utiliza un algoritmo genético que calcule los parámetros necesarios para configurar un sistema de planificación basado en prioridades que sea capaz de atender tareas con restricciones de tiempo real. Se plantea el uso de infraestructura de apoyo o intermedia para obtener un sistema de fácil modificación y mantenimiento y para demostrar que es posible obtener respuestas de tiempo real suave utilizando esta capa intermedia de software.

La planificación de tiempo real de problemas de gran escala en dominios complejos representa varias dificultades para las técnicas de búsqueda y optimización, como espacios de búsqueda grandes y complejos y problemas que cambian dinámicamente. Los algoritmos genéticos se usan en la resolución de problemas de este tipo debido a su adaptabilidad y eficiencia de búsqueda en espacios grandes y a su relativa sencillez de programación. Muchos problemas de planificación óptima de tiempo real no se pueden resolver por técnicas de investigación de operaciones tradicionales, un problema de planificación incluye asignar cada una de n tareas a un recurso con un orden en particular. Cuando tenemos un sólo recurso como puede ser el procesador de una computadora o una red de comunicaciones, el número de posibles planes es $n!$ (Montana, et al., 1998) lo cual implica un crecimiento exponencial como función del número de tareas.

La solución que se espera encontrar con este tipo de algoritmos es subóptima dada su naturaleza no determinística, ya que se debe asignar un tiempo limitado a la búsqueda de la solución por el hecho de encontrarnos en el contexto de sistemas de tiempo real. La utilización de algoritmos genéticos para encontrar resultados en tiempos acotados y reducidos implica un compromiso de diseño ya que por su misma naturaleza no se apegan a estos requerimientos.

El sistema que se analiza está construido sobre la infraestructura que otorga TAO aprovechando las ventajas que ofrece en cuanto a modularidad, facilidad de desarrollo y manejo de los procesos de comunicación. Al utilizar TAO se eliminan dificultades técnicas que tienen que ver con protocolos de comunicación y utilización de distintos sistemas operativos y lenguajes de programación. Sobre esta plataforma se prueba la factibilidad de realizar una reconfiguración en línea de los distintos procesos que intervienen en un sistema en general. Se utiliza hardware de bajo costo de uso generalizado y software disponible libremente para probar el comportamiento de este tipo de sistemas en las condiciones predominantes en el país.

1.1 Objetivos

Los objetivos de este estudio se resumen en los siguientes puntos:

- **Definir una estrategia de planificación con base en la optimización del orden de ejecución de procesos, usando un algoritmo genético para la reconfiguración en línea de un sistema distribuido de tiempo real.**
- **Evaluar el sistema mencionado basándolo en CORBA de tiempo real (RT-CORBA) para la reconfiguración en línea.**

La estrategia de planificación se realiza con el método semi-dinámico llamado MUF utilizando la información encontrada por el algoritmo genético. Este algoritmo maneja una función de adaptación que utiliza un modelo con el cual se pueden simular las salidas que son importantes controlar en el sistema para calificar a los individuos que representan las diferentes soluciones generadas en la población. En esta búsqueda se identifican las prioridades de las tareas y el período necesario para atenderlas adecuadamente siempre con el objetivo de mejorar la respuesta del sistema. El resultado más importante es lograr la reconfiguración del sistema sin detenerlo y sin afectar de manera grave su respuesta o comportamiento. El sistema a controlar o reconfigurar puede ser cualquiera que sea distribuido y de tiempo real.

El segundo objetivo se relaciona con la capacidad de utilizar una plataforma tipo CORBA para construir sistemas de tiempo real utilizando computadoras personales, el sistema operativo Linux y una red de comunicaciones basada en Ethernet. Al utilizar estos elementos, el soporte para manejar tiempo real no está completamente garantizado, pero se puede lograr identificando los límites en cuanto a tiempos de respuesta, número de tareas y periodicidad de las tareas. La utilización de TAO en sistemas de tiempo real tiene como desventaja la cantidad de recursos de cómputo necesarios para soportarlo, pero aún en estas condiciones es posible garantizar la atención de tareas y la generación de resultados en tiempos acotados. Es importante comprender que un sistema de tiempo real no tiene que ser necesariamente pequeño y responder en tiempos de microsegundos o milisegundos, sino dentro de los tiempos que previamente se hayan definido; en síntesis, debe ser predecible. La garantía de respuesta a tiempo se controla al nivel de la aplicación y dependerá siempre de las posibilidades que otorgue el hardware, el sistema operativo y la red de comunicación utilizados.

1.2 Relevancia

El uso de algoritmos genéticos para reconfiguración de tareas (Montana, et al., 1998) representa una oportunidad para construir sistemas de fácil mantenimiento y reconfiguración debido a la capacidad de adaptación inherente de estos métodos evolutivos. Trabajos similares han sido propuestos, como el manejador adaptable de recursos, que busca optimizar tareas en línea dentro de un marco de sistemas

distribuidos (Huang, et al., 1997), aún cuando su diferencia radica en el método de aproximación que en este caso es un problema de optimización. Se busca aportar con este trabajo nuevas bases y experiencia que ayuden a identificar las posibilidades y problemas que se presentan al utilizar una plataforma como CORBA en aplicaciones de tiempo real. También se pretende generar información relacionada con la posibilidad de reconfigurar, por medio de algoritmos genéticos, sistemas que utilizan middleware. El método propuesto utiliza un caso de estudio definido en el campo del control digital, pero no implica que la propuesta no pueda usarse en el diseño y construcción de otro tipo de aplicaciones basadas en un sistema de cómputo distribuido de tiempo real compuesto de tareas periódicas. La relevancia de este estudio radica en probar la posibilidad de combinar métodos evolutivos y una plataforma como CORBA en problemas de reconfiguración en tiempo real de manera exitosa y efectiva.

La elección de CORBA se realizó tomando en cuenta las siguientes ventajas:

- Se trata de un programa código abierto o libre, que puede modificarse.
- Es gratuito y se puede obtener fácilmente.
- Existen referencias suficientes para su utilización en libros y páginas web.
- Está construido utilizando las mejores técnicas de programación, por lo que se puede considerar como una buena alternativa para construir sistemas robustos.
- La programación se realiza en C++, el cual es el lenguaje que mejor conocen los desarrolladores del proyecto.
- Se puede utilizar en el sistema operativo Windows o Unix.

Este documento ha sido planeado para mostrar de manera clara los resultados obtenidos del análisis y en segundo término para ayudar a quienes se interesen en la utilización de una plataforma como CORBA a conocer de una forma más rápida los detalles técnicos, las referencias y los problemas más comunes que deberán enfrentar al manejar, construir y modificar sistemas distribuidos. Se ha organizado de la siguiente manera:

- El capítulo 2 presenta los antecedentes y referencias necesarios para entender temas como CORBA, algoritmos genéticos y sistemas distribuidos.
- El capítulo 3 contiene la propuesta de desarrollo describiendo la estructura de la aplicación, los algoritmos genéticos utilizados y la forma en la que se utiliza la infraestructura de CORBA.
- El capítulo 4 presenta el caso de estudio concreto que se ha elegido, el cual es un sistema de control digital de segundo orden con múltiples sensores y actuadores conectados a un controlador principal.
- El capítulo 5 presenta los resultados generados, en el capítulo 6 se muestran las conclusiones obtenidas y en el capítulo 7 se incluye un apéndice con el código fuente del programa.

2. Antecedentes

El presente trabajo trata sobre la construcción de un sistema de cómputo distribuido en el cual se utiliza un planificador para organizar la atención a las tareas. Estas tareas tienen restricciones de tiempo que se deben cumplir y dependen en gran medida de la infraestructura que soporta la aplicación. En este caso la infraestructura está representada por TAO. La reconfiguración de las tareas para adecuarse al estado actual del sistema y del ambiente depende de un algoritmo genético que busca la mejor solución basándose en el comportamiento del sistema. Para el correcto entendimiento de los conceptos y lenguaje utilizado es necesario introducir algunos elementos básicos de la teoría a utilizar.

Este capítulo es una guía para conocer los diferentes temas que sirven como base para el estudio de un sistema de cómputo distribuido reconfigurable y el manejo de restricciones de tiempo real. Sobre cada tema se ofrece una introducción y las referencias necesarias para iniciar un estudio más profundo. El presente capítulo habla sobre teoría de sistemas distribuidos, sistemas de tiempo real, planificadores de tareas, CORBA, TAO y algoritmos genéticos. Con esta guía se pretende facilitar al lector la búsqueda de la información necesaria para poder analizar y comprender los capítulos posteriores.

2.1 Sistemas de Cómputo Distribuido

Un sistema de cómputo distribuido (Coulouris, et al., 2001) es aquél cuyos componentes, ya sea de hardware o software, están localizados en computadoras en red que se comunican coordinando sus acciones sólo por paso de mensajes. La estructura del sistema distribuido se representa con un conjunto de nodos interconectados por un sistema de comunicaciones. Cada nodo del sistema tiene definida una interfase de comunicación que se encarga de la transmisión de información entre los diferentes nodos. Detrás de esta interfase se esconde la estructura interna de los programas que controlan a cada nodo.

Los sistemas distribuidos son de naturaleza diversa. La red de redes, Internet, permite que los usuarios de todo el mundo utilicen sus servicios donde quiera que estén situados. Cada organización administra una intranet que provee servicios locales y servicios de Internet a los usuarios locales y a otros usuarios de Internet. Es posible construir pequeños sistemas distribuidos con computadores portátiles y otros dispositivos computacionales pequeños conectados a una red inalámbrica. Compartir recursos es el principal factor que motiva la construcción de sistemas distribuidos, recursos como impresoras, archivos o registros de bases de datos.

La construcción de los sistemas distribuidos presenta muchos desafíos, a continuación se mencionan algunas características que deben tomarse en cuenta en este tipo de sistemas:

- Heterogeneidad: debe construirse sobre una variedad de diferentes redes, sistemas operativos, hardware y lenguajes de programación.

- Extensibilidad: los sistemas distribuidos deberían ser extensibles, el primer paso es la publicación de las interfases de sus componentes, pero la integración de componentes escritos por diferentes programadores es un auténtico reto.
- Seguridad: se pueden encriptar los datos para proporcionar una protección adecuada a los recursos compartidos y mantener secreta la información importante cuando se transmite un mensaje a través de la red.
- Escalabilidad: un sistema distribuido es escalable si el costo de añadir un usuario es una cantidad constante en términos de recursos que se deberán añadir. Los algoritmos empleados para acceder a los datos compartidos deben evitar cuellos de botella y los datos deben estar estructurados jerárquicamente para dar los mejores tiempos de acceso. Los datos frecuentemente utilizados pudieran estar replicados.
- Tolerancia a fallas: cualquier proceso, computadora o red puede fallar independientemente de los otros. En consecuencia, cada componente necesita estar al tanto de las formas posibles en que pueden fallar los componentes de los que depende y estar diseñado para tratar apropiadamente con estas fallas.
- Concurrencia: La presencia de múltiples usuarios en un sistema distribuido es una fuente de peticiones concurrentes a sus recursos. Cada recurso debe estar diseñado para estar disponible de forma ordenada en un entorno concurrente.

2.2 Tiempo Real

Cuando se habla de tiempo real se hace referencia a una acción que ocurre inmediatamente o dentro de un tiempo bien establecido. Actualmente el término es muy usado para describir diferentes características de los sistemas de cómputo, por ejemplo, existen sistemas operativos de tiempo real que permiten utilizar funciones y estructuras que garantizan tiempos de respuesta finitos por lo regular se usan para tareas como la navegación aérea, en la cual la computadora debe reaccionar a un flujo continuo de información sin interrupción. La mayoría de los sistemas operativos de propósito general no son de tiempo real porque muchas de sus funciones no son determinísticas al reaccionar a algún evento o acción, por lo que no es posible garantizar el tiempo de respuesta.

En todo este documento la frase "de tiempo real" se refiere a la teoría de los sistemas de tiempo real, ya que todo sistema funciona "en tiempo real" porque existe en el tiempo. La característica principal de estos sistemas, es la respuesta que se obtiene dentro de un límite de tiempo bien acotado que dependiendo de la clase de aplicación puede ser medido en milisegundos o microsegundos. Un sistema de tiempo real es aquel que de acuerdo a restricciones temporales realiza su trabajo de manera determinística. No es un requisito que el sistema sea muy rápido, sino que las respuestas se generen dentro de un tiempo finito.

La importancia del estudio de sistemas de tiempo real, se basa en la utilidad que tienen estos sistemas

en muchas tareas críticas que pueden incluso implicar la pérdida de vidas si se presenta una falla. Debido a su importancia se analizarán estrategias que aseguran que las respuestas del sistema sean las correctas y se generen a tiempo. Un sistema de tiempo real es aquel en el cual el momento en el que se produce una salida es importante, usualmente porque la entrada corresponde a algún evento en el mundo físico y la salida tiene que relacionarse con el mismo evento. La distancia entre el tiempo de entrada y el tiempo de salida debe ser suficientemente pequeña para estar dentro de un valor límite aceptado. También puede definirse como cualquier sistema que tiene que responder a entradas generadas externamente en un período finito. Se considera que un sistema de tiempo real funciona adecuadamente si el resultado lógico de la computación es correcto y se cumple con la restricción de tiempo dentro de la cual deben generarse los resultados.

En el campo de sistemas de tiempo real a menudo se hacen dos distinciones, sistemas de tiempo real estricto y sistemas de tiempo real flexible. Los sistemas de tiempo real estricto son aquellos en los que es absolutamente imperativo que las respuestas ocurran antes de un límite determinado. Los sistemas de tiempo real flexible son aquellos donde los tiempos de respuesta son importantes pero el sistema seguirá funcionando aceptablemente si los límites son sobrepasados ocasionalmente.

Mientras más control se le da a las computadoras, más imperativo se vuelve que éstas no fallen. La falla de un sistema relacionado con transferencias automáticas de fondos entre bancos puede causar que se pierdan grandes cantidades de dinero irremediablemente, un componente que no funciona correctamente en la generación de electricidad, puede causar la falla de un sistema que da soporte a una vida en una unidad de cuidados intensivos. Estos ejemplos dramáticos demuestran que el hardware y el software deben ser seguros y confiables para muchas aplicaciones. En ambientes hostiles, como los que se encuentran en las aplicaciones militares, debe ser posible diseñar y construir sistemas que fallen de manera controlada. Aún más, cuando se necesite de la interacción con un operador, se debe tener cuidado en el diseño de la interfase para minimizar la posibilidad de un error humano. También se deben hacer consideraciones sobre la capacidad de los sistemas y la demanda de trabajo a la que estarán sometidos.

El tiempo de respuesta es crucial en un sistema de tiempo real, pero es difícil diseñar y construir sistemas que garanticen que se generará la salida apropiada en los instantes apropiados bajo todas las posibles condiciones. Lograr esto y hacer uso completo de todos los recursos de cómputo todo el tiempo también es muy difícil. Por esto, los sistemas de tiempo real se construyen usualmente con procesadores de capacidad mayor a la necesaria en un caso promedio asegurando que el comportamiento en el peor caso posible no produzca retardos durante los períodos críticos de la operación del sistema. Los sistemas de tiempo real deben ser capaces de responder ante situaciones donde los requerimientos de tiempo cambian dinámicamente. Un ejemplo de esta adaptación o reconfiguración se encuentra en los sistemas de control de vuelo. Si un aeroplano experimenta despresurización, existe una necesidad inmediata de que los recursos se dediquen a manejar la emergencia.

En sistemas distribuidos de tiempo real, el subsistema de comunicación constituye una parte muy importante. Su diseño debe considerar las características de determinismo en la transmisión de información en todos los niveles de los distintos protocolos de comunicación con el fin de lograr un desempeño predecible para ofrecer a las aplicaciones. En la figura 2.1 se muestran dos sistemas que pueden considerarse de tiempo real. En un automóvil el sistema de frenos debe controlarse de la mejor manera posible porque una falla puede producir daños serios al vehículo y la muerte de personas. En un sistema de bandas transportadoras se pueden arruinar piezas o productos si están pasando por un proceso de calentamiento, presión o cualquier otro en el que el tiempo sea un factor determinante.

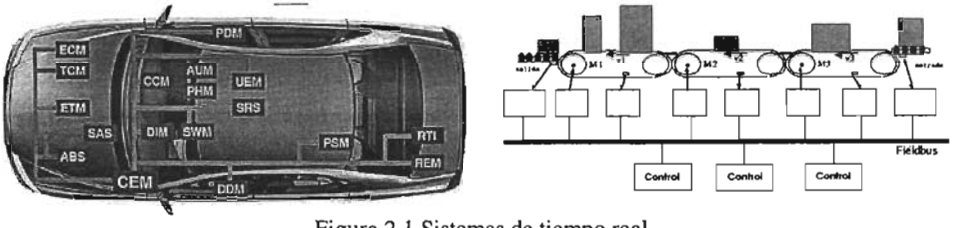


Figura 2.1 Sistemas de tiempo real

2.3 Sistemas Distribuidos de Tiempo Real

Un sistema con características de tiempo real y las de un sistema distribuido se conoce como un sistema distribuido de tiempo real (García, 2003). Un sistema de este tipo se compone de un conjunto de nodos interconectados por un sistema de comunicaciones de tiempo real que constituye el elemento más importante. Su diseño debe considerar las características de determinismo en la transmisión de información para no generar retrasos impredecibles que afecten a los nodos que forman parte del sistema.

2.4 Planificadores

En las aplicaciones de tiempo real, el funcionamiento correcto no sólo depende de los resultados de los cálculos, también depende del instante en el que éstos son obtenidos. Las aplicaciones y concretamente la política de planificación debe garantizar que todas las tareas críticas sean capaces de cumplir sus plazos de ejecución (obtener los resultados antes del plazo máximo). Desde el punto de vista de la planificación (Levine, et al., 1998), el sistema operativo trata las tareas como procesos que consumen una cierta cantidad de tiempo de procesador y a las que se debe asignar cierta cantidad de tiempo.

Un planificador es un método para asignar recursos. Un conjunto de tareas se define como factible si existe algún planificador que sea capaz de cumplir las restricciones de todas las tareas. Un planificador es útil si es capaz de planificar correctamente cualquier conjunto de tareas factible. Para estudiar el proceso de planificación se toman en cuenta tres tipos de tareas con las siguientes características:

Tareas periódicas:

- *C*: tiempo de ejecución en el peor de los casos
- *T*: período de repetición. Cada *T* unidades de tiempo se activa la tarea
- *D*: plazo de finalización o "tiempo límite": Tiempo máximo que puede transcurrir entre la activación de la tarea y la finalización de la ejecución de ésta.

Tareas esporádicas:

- *C*: tiempo de ejecución en el peor de los casos
- *T*: período mínimo entre dos activaciones consecutivas.
- *D*: plazo de finalización o "tiempo límite": Tiempo máximo que puede transcurrir entre la activación de la tarea y la finalización de la ejecución de ésta.

Tareas aperiódicas:

- *C*: tiempo de ejecución en el peor de los casos
- *T*: período mínimo entre dos activaciones consecutivas.
- *D*: plazo de finalización o "tiempo límite": Tiempo máximo que puede transcurrir entre la activación de la tarea y la finalización de la ejecución de ésta. Este atributo es opcional. Las tareas aperiódicas críticas tienen plazos de finalización, mientras que las tareas aperiódicas críticas no tienen plazos de ejecución.

Si se atiende a las características semánticas de las tareas, se dividen en dos grupos:

- Críticas: el fallo de una de estas tareas puede resultar en un sistema que deja de funcionar.
- Opcionales (no críticas): se pueden utilizar para refinar el resultado dado por una tarea crítica o para monitorear el estado del sistema.

Los estudios de planificación de este trabajo suponen que todas las tareas son periódicas y que el plazo de finalización llega cuando se activa la misma tarea de nuevo (en *T* unidades de tiempo). Los objetivos que persigue toda política de planificación de tiempo real son:

- Garantizar la correcta ejecución de todas las tareas críticas
- Administrar el uso de recursos compartidos
- Soportar cambios en los tiempos de ejecución de las tareas

Inicialmente se suponen las siguientes simplificaciones, aunque un análisis más serio deberá eliminarlas según sea necesario:

- Las tareas son independientes
- No comparten recursos ni se comunican entre ellas
- Todas las tareas son periódicas

- Los tiempos de cambio de contexto son despreciables, es decir, el tiempo de cambio de la ejecución de una tarea a otra no se contempla.
- Existe un sólo procesador

Los primeros planificadores se diseñaban a mano, esto es, se construía durante la fase de diseño del sistema un plan con todas las acciones que tenía que llevar a cabo el planificador durante la ejecución del sistema. El planificador tan sólo tenía que consultar la tabla. Las órdenes de planificación que estaban en la tabla se repetían constantemente. A estos planificadores se les llama cíclicos o estáticos. El principal problema que representan es la poca flexibilidad a la hora de modificar alguno de los parámetros de las tareas, pues ello conlleva la re-elaboración de todo el plan. El otro gran grupo de planificadores son los basados en prioridades. En un sistema manejado por prioridades, las tareas de mayor prioridad se ejecutan antes que las de menor prioridad y es posible que una tarea sea interrumpida por otra de mayor prioridad. Los planificadores basados en prioridades se dividen a su vez en estáticos y dinámicos. En los planificadores estáticos, durante la fase de diseño a cada tarea se le asigna una prioridad. Después, durante la ejecución, el algoritmo de planificación tiene que ordenar la ejecución en función de la prioridad asignada. En los planificadores dinámicos, la prioridad depende de una función del tiempo en la que se toman en cuenta las características de las tareas. Es posible lograr un plan factible y medir el porcentaje de utilización de CPU (U) que indica que tanto tiempo se utiliza de un período para ejecutar las tareas. El período que se toma en cuenta es aquel en el que todas las tareas se han activado por lo menos una vez y que es el mínimo común múltiplo de los períodos de las tareas. La factibilidad de un plan debe medirse a partir del instante crítico, que es cuando se activan todas las tareas al mismo tiempo. Si es posible garantizar en este caso que todas las tareas cumplan con sus restricciones de tiempo, entonces se puede garantizar que en cualquier otra forma en que se activen las tareas también se pueden cumplir todas las restricciones. A continuación se describen varios planificadores basados en prioridades (Levine, et al., 1998), (Burns, et al., 1990):

Rate Monotonic (RM) (Liu, et al., 1973) : Las prioridades más altas se asignan estáticamente a las tareas con períodos de activación más pequeños. Estas prioridades son fijas. El sistema operativo debe soportar interrupción de tareas para que las de mayor prioridad siempre estén corriendo.

Earliest Deadline First (EDF): Es una estrategia de planificación dinámica que ordena a las operaciones basado en el tiempo faltante para el tiempo límite más próximo. Las operaciones cuyo tiempo límite esté más próximo se ejecutan primero. Tiene como desventaja que si una tarea no puede completar su ejecución y su tiempo límite ya está cercano, de todos modos utiliza el procesador quitando tiempo a otras tareas importantes.

Minimum Laxity First (MLF): La laxidad se define como el tiempo que falta para el límite menos el tiempo de ejecución restante de la tarea. Este método refina el EDF tomando en cuenta si una tarea es capaz de completar o no su ejecución.

MUF: La estrategia de planificación MUF soporta el rigor del planificador estático RMS y la flexibilidad de las estrategias dinámicas como EDF y MLF. MUF puede asignar componentes de prioridad estáticos y dinámicos. Cada tarea tiene los siguientes componentes o propiedades:

- Nivel crítico (NC): En MUF, las operaciones con alto nivel crítico son asignadas a niveles de prioridad estáticos más altos. La asignación de prioridades estáticas de acuerdo al nivel crítico previene que las operaciones críticas para la aplicación sean interrumpidas por otras operaciones. Ordenar las operaciones por nivel crítico definido en la aplicación refleja un cambio fundamental en la noción de asignación de prioridades. En particular RMS, EDF y MLF exhiben un mapeo rígido de las características empíricas de operación a un valor de prioridad y casi no hay control sobre qué operaciones perderán sus tiempos límite bajo condiciones de sobrecarga. En contraste, MUF da a las aplicaciones la habilidad de distinguir a las operaciones arbitrariamente. MUF permite el control de cuales operaciones podrán perder sus tiempos límite, por lo tanto, se puede proteger un subconjunto crítico del total de operaciones.
- Subprioridad dinámica (SD): La subprioridad dinámica de una operación es evaluada cuando debe ser comparada con la subprioridad dinámica de otra operación. En el instante de evaluación, la subprioridad dinámica en MUF es una función de la laxidad de una operación. Asignando subprioridades dinámicas de acuerdo a la laxidad, MUF ofrece una mayor utilización del CPU que las estrategias estáticas. MUF también permite detectar fallas de tiempo límite antes de que ocurran, excepto cuando la operación es interrumpida por otra con mayor nivel crítico.
- Subprioridad estática (SE): En MUF, la subprioridad estática es una prioridad opcional específica de la aplicación. Se usa para ordenar la atención a las operaciones que tienen el mismo nivel crítico y la misma subprioridad dinámica. Así, la subprioridad estática tiene menor precedencia que el nivel crítico y que la subprioridad dinámica. La asignación de una subprioridad estática única a operaciones que tienen el mismo nivel crítico asegura un orden total en la atención de las operaciones en tiempo de ejecución.

En la figura 2.2 se muestra el orden en que se ejecutan las tareas tomando en cuenta su nivel crítico, período y tiempo de ejecución para las tres estrategias dinámicas. Se puede apreciar que MUF atiende primero a la tarea marcada con alto nivel crítico. MLF da más importancia por tener la mínima laxidad y EDF da más importancia a la tarea que tiene el tiempo límite más cercano.

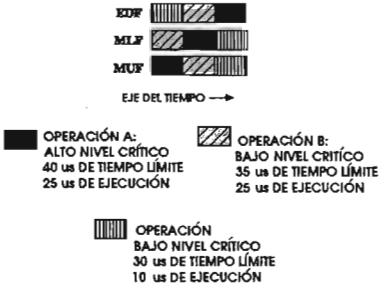


Figura 2.2 Orden de ejecución de tareas (imagen tomada de (Gill, et al., 1998))

2.5 CORBA

La introducción al manejo de esta plataforma o arquitectura demanda conocimientos de programación orientada a objetos, uso de patrones, sistemas distribuidos y en menor grado del manejo del lenguaje unificado de modelado (UML, Unified Modeling Language) (Page-Jones, 1999). Las principales fuentes de información sobre este tema son la página web del (OMG, Object Management Group) (OMG, 2001) y la página web de TAO (CDOC, 1997). En 1989 se creó el OMG, el cual desarrolla estándares para la construcción de aplicaciones en ambientes distribuidos heterogéneos, es decir, en ambientes con distintos sistemas operativos, lenguajes de programación, protocolos de comunicación y arquitecturas de computadoras. Una característica importante de las grandes redes de computadoras como la Internet, la World Wide Web (WWW) y las intranets corporativas es que son heterogéneas. Por ejemplo, una intranet corporativa puede constituirse de mainframes, servidores UNIX, computadoras personales corriendo distintas versiones de Windows, OS/2, Macintosh e incluso dispositivos como teléfonos, sensores, etc.

Las redes y los protocolos que soportan todos estos sistemas pueden incluir Ethernet, FDDI, ATM, TCP/IP, Novell, etc. Los miembros del OMG son compañías y Universidades que se interesan en el desarrollo de estos estándares. Este grupo maneja las peticiones y sugerencias de sus miembros a través de peticiones de propuestas (RFP, Requests for Proposals) que son documentos en los que se hace una proposición para que todos la evalúen y que hacen que se adopten especificaciones basándose en la tecnología disponible comercialmente. La especificación de CORBA es el producto principal del esfuerzo de este grupo en la construcción de la arquitectura de manejo de objetos (OMA, Object Management Architecture) (Vinoski, 1997) que ofrece la infraestructura conceptual sobre la cual se basan todas las especificaciones de la OMG.

La OMA está compuesta de un modelo de objetos y un modelo de referencia. El modelo de objetos define cómo los objetos distribuidos a través de un ambiente heterogéneo pueden describirse, mientras que el modelo de referencia caracteriza la interacción entre estos objetos. La arquitectura común de intermediario de peticiones de objetos (CORBA, Common Object Request Broker Architecture) es una capa intermedia de software (middleware) que permite la comunicación entre entidades o programas que funcionan en distintas máquinas. El OMG se encarga de publicar la especificación de CORBA y se puede encontrar en su página web gratuitamente. CORBA es una arquitectura e infraestructura independiente de un vendedor o fabricante que las aplicaciones de computadora usan para trabajar juntas a través de una red.

Usando protocolos estándar, un programa basado en CORBA de cualquier compañía o fabricante en cualquier computadora, sistema operativo, lenguaje de programación y red puede interactuar con otro programa basado en CORBA de cualquier otro vendedor, corriendo en otro sistema operativo, escrito en otro lenguaje de programación y funcionando en otro tipo de red. CORBA es útil en muchas situaciones. Debido a la facilidad con la que CORBA integra máquinas de tantos vendedores, con tamaños muy diferentes que van desde mainframes hasta sistemas empotrados o palms, es el

middleware elegido por muchas compañías importantes.

Las aplicaciones de CORBA se componen de objetos, que son unidades individuales de software que combinan funciones y datos. Para cada tipo de objeto se define una interfase utilizando el lenguaje de definición de interfases (IDL, Interface Definition Language) (Vinoski, 1997) de OMG. La interfase es la parte sintáctica del contrato que el objeto servidor ofrece a los clientes que lo invocan. Cualquier cliente que quiera invocar una operación sobre un objeto debe utilizar esta interfase IDL para especificar la operación que quiere utilizar y enviar los argumentos necesarios. Cuando la invocación alcanza a un objeto la misma definición de la interfase se usa para obtener los argumentos para que el objeto pueda ejecutar el método adecuado. La definición de la interfase IDL es independiente del lenguaje de programación, pero se mapea a todos los lenguajes de programación populares. OMG ha estandarizado mapeos de IDL a C, C++, Java, COBOL, Smalltalk, Ada, Lisp, Python e IDLscript. La separación de interfase e instanciación es la esencia de CORBA, la interfase de cada objeto se define estrictamente, pero la instanciación (métodos y datos) se esconde del resto del sistema en una parte del sistema que los clientes no pueden alcanzar. Los clientes usan los objetos a través de sus interfases invocando sólo aquellas operaciones expuestas a través de la interfase IDL con sólo los parámetros de entrada y salida que también están definidos en la interfase.

En CORBA, cada instancia de un objeto tiene su propia y única referencia de objeto. Los clientes usan las referencias a objetos para hacer sus invocaciones identificando la instancia exacta que quieren utilizar. La comunicación entre los clientes y los objetos que ofrecen servicios se realiza a través del Object Request Broker (ORB) que es el objeto principal de la infraestructura de CORBA y se encarga de realizar las tareas necesarias para recibir las peticiones, buscar al objeto que debe atenderlas, transmitir los argumentos, recibir la respuesta y entregarla al cliente que hizo la invocación. El ORB se encarga de todos los detalles relacionados con el ruteo de peticiones de los clientes a los objetos y regresar la respuesta a su destino.

2.6 TAO

Para los desarrolladores de aplicaciones distribuidas que tienen demandas fuertes de desempeño, The ACE ORB (TAO) es una instanciación de CORBA libre, gratuita, de código abierto desarrollado en la Universidad de California en Irvine, EUA [17] y que cumple con los estándares del OMG. Además tiene extensiones de tiempo real. TAO aplica las mejores prácticas y patrones de software para mejorar el desempeño de aplicaciones distribuidas. TAO se basa en un conjunto de librerías conocidas como ambiente adaptable de comunicación (ACE, Adaptive Communication Environment) (Schmidt, 1994). Estas librerías ofrecen el soporte necesario para construir un sistema portable y apegado a buenas prácticas de desarrollo. Existen diferentes instanciaciones de CORBA (Visibroker, TAO, Orbacus, etc.) pero este estudio se concentra en TAO por considerarlo una herramienta lo suficientemente estable y robusta para cumplir con los propósitos académicos. La figura 2.3 muestra los módulos que conforman la instanciación de CORBA en TAO.

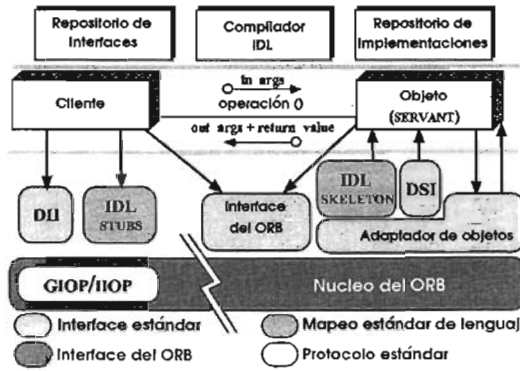


Figura 2.3 Módulos de TAO (imagen tomada de (Gill, et al., 1998))

TAO utiliza como lenguaje de programación C++ para definir el comportamiento de los objetos y funciona en varias versiones de UNIX y Windows y en sistemas con características de tiempo real como VxWorks, Chorus, QNX y otros. TAO cumple con la especificación de CORBA y de RT-CORBA añadiendo otras características para corregir problemas presentes en el manejo de sistemas de tiempo real que no están consideradas en la especificación original. TAO es un middleware que permite a los clientes invocar operaciones en objetos distribuidos sin preocuparse de la localización, del lenguaje de programación, del sistema operativo, los protocolos de comunicación o del hardware.

2.7 El ORB de Tiempo Real

TAO es un sistema intermediario de peticiones de objetos (ORB, Object Request Broker) [17] de alto desempeño de tiempo real para aplicaciones con requerimientos de calidad determinista y estadística. TAO soporta los estándares del modelo CORBA de OMG y agrega mejoras para asegurar que los requerimientos de las aplicaciones sean cumplidos.

- IDL stubs y esqueletos: los stubs y esqueletos generados por TAO realizan el marshaling y demarshaling de los parámetros de las operaciones eficientemente y aseguran que se cumplan los requerimientos de tiempo de principio a fin, es decir, todas estas operaciones son determinísticas.
- Objeto adaptador de tiempo real: Un objeto adaptador, el cual es parte importante de la especificación de CORBA, asocia objetos servidores con el ORB y demultiplexa las peticiones hacia ellos. El objeto adaptador envía las operaciones en tiempo constante $O(1)$ a los objetos que ofrecen servicios sin importar el número de conexiones, objetos servidores y operaciones definidas en las interfases IDL.
- Planificador de tiempo real del ORB: El planificador de TAO mapea los requerimientos de las aplicaciones a los recursos del sistema y de la red. Comúnmente, estos requerimientos incluyen limitar la latencia y cumplir límites de tiempo periódicos. Los recursos administrados incluyen el

procesador, memoria, conexiones de red y dispositivos de almacenamiento.

- Núcleo de tiempo real del ORB: El núcleo del ORB lleva las peticiones de los clientes al objeto adaptador y regresa las respuestas a los clientes. Este núcleo usa una arquitectura basada en múltiples hilos de ejecución, de tareas interrumpibles, concurrentes y basadas en prioridad para dar un protocolo IIOP (OMG, 2001) eficiente y predecible. Este protocolo especifica cómo se realiza la comunicación utilizando TCP/IP.

2.8 El Servicio de Planificación

El servicio de planificación de TAO (Gill, et al., 1998) está diseñado para soportar varias estrategias de planificación como RM, EDF, MLF y MUF. La arquitectura y comportamiento del servicio de planificación de TAO se muestra en la figura 2.4. Los pasos del 1 al 6 típicamente ocurren fuera de línea durante el proceso de configuración de la planificación, mientras que los pasos 7-10 ocurren en línea.

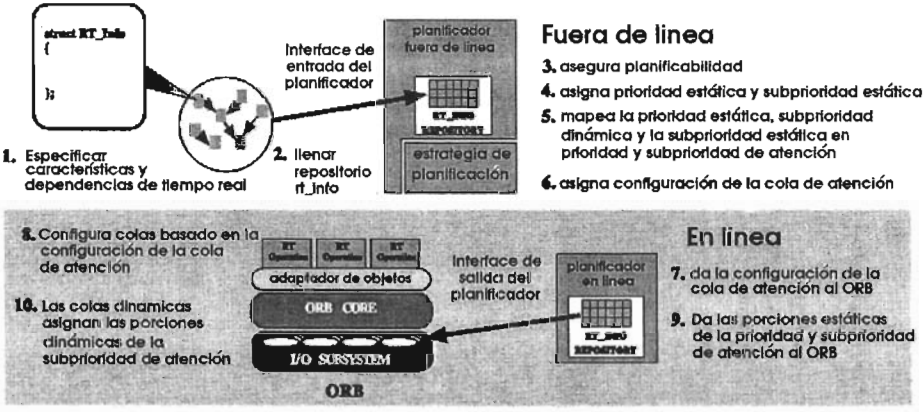


Figura 2.4 Pasos de la planificación (imagen tomada de (Gill, et al., 1998))

- Paso 1: Una aplicación CORBA especifica la información sobre la calidad del servicio (QOS, Quality Of Service) que necesita y la pasa al servicio de planificación de TAO, el cual está instanciado como un objeto CORBA. La aplicación especifica un conjunto de valores con las características de cada una de las tareas. Adicionalmente se pueden indicar las dependencias entre las tareas.

- Paso 2: En el momento de la configuración, la cual puede suceder en línea o fuera de línea, la aplicación pasa la información al servicio de planificación por medio de su interfase de entrada. El servicio de planificación construye gráficas de dependencia de las tareas basadas en la información registrada por la aplicación. Después se identifican hilos de ejecución examinando los nodos terminales de estas gráficas.
- Paso 3: El servicio de planificación indica la planificabilidad de las tareas. Un conjunto de tareas se considera planificable si todas las tareas del conjunto crítico pueden cumplir con sus requerimientos de tiempo.
- Paso 4: Se asignan prioridades y subprioridades estáticas a las tareas. Estos valores se asignan de acuerdo a la estrategia especificada para usarse (MUF, EDF, RM o MLF).
- Paso 5: Se toman la prioridad de atención y la subprioridad de atención y se calculan los demás componentes asignados estática y dinámicamente.
- Paso 6: Se crean colas para ordenar la atención de las tareas. El número de colas y su tipo depende de la estrategia a seguir.
- Paso 7: En tiempo de ejecución, la información de la configuración es utilizada por el servicio de planificación. El ORB usa el planificador en tiempo de ejecución para obtener la prioridad del hilo de ejecución con la cual serán atendidas las tareas.
- Paso 8: El ORB configura sus módulos de atención de tareas, el subsistema de entrada-salida, el núcleo del ORB y el servicio del canal de eventos de tiempo real. Este servicio sirve para organizar la ejecución de las tareas por medio del manejo de eventos y es parte de TAO. Los eventos en este contexto son llamadas a las funciones que forman parte de la interfase del canal de eventos y que indican que se desea transmitir datos a algún objeto. Los módulos de atención de tareas son los componentes por los cuales atraviesa el flujo de acciones necesarias para atender una tarea. La figura 2.5 indica cómo están organizados estos distintos niveles.

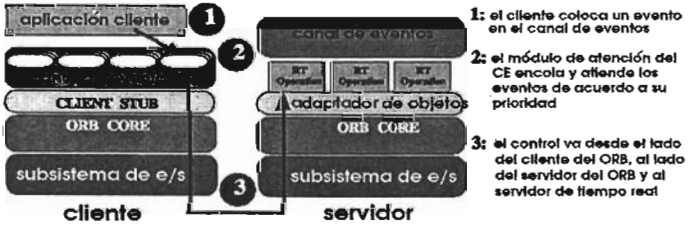


Figura 2.5 Módulos que utiliza el canal de eventos (imagen tomada de (Gill, et al., 1998))

- Paso 9: Cuando llega una petición de atención de una tarea desde un cliente, los módulos de atención identifican la cola de atención a la cual corresponde la petición e inicializan la subprioridad de atención.
- Paso 10: Si la cola de atención donde la operación estaba formada es del tipo dinámico, los componentes dinámicos de la tarea son asignados.

2.9 El Servicio del Canal de Eventos de Tiempo Real

El servicio de planificación utiliza otro servicio de CORBA llamado canal de eventos de tiempo real (Harrison, et al., 1997) para lograr la ejecución ordenada de las tareas. El servicio de canal de eventos de tiempo real es un objeto CORBA que sirve como un intermediario entre consumidores y generadores de eventos. Los eventos son acciones generadas por algún objeto que pueden activar diferentes funciones. Estos eventos sirven para enviar datos o hacer peticiones de atención entre objetos de una aplicación CORBA. Los objetos se pueden registrar en el canal de eventos para indicar que generarán eventos a la vez de especificar un tipo para diferenciarlos de los eventos de otros objetos. Los consumidores pueden registrarse con el canal de eventos para indicar que desean recibir uno o más tipos de eventos. Cuando un objeto genera un evento, éste es registrado por el canal de eventos y es enviado a los consumidores que hayan establecido interés en recibirlo. El canal de eventos de tiempo real está basado en el servicio de canal de eventos e instancia prácticamente la misma funcionalidad e interfase, pero se encarga de los detalles que tienen que ver con la garantía de ejecución de tareas de tiempo real. Es tarea del canal de eventos hacer las operaciones necesarias para elegir a que consumidores se envía un evento y a cuales no. En cada extremo de la figura 2.6 se muestran los módulos conocidos como proxies o intermediarios. Estos se encargan de registrar a los consumidores y generadores de eventos y de hacer la elección de a quienes llegarán los eventos. El canal de eventos contiene un módulo con relojes que pueden ser utilizados para generar eventos con un período específico y enviarlos a los consumidores interesados. El módulo de correlación de eventos permite especificar al canal de eventos que sólo dispare un evento basado en reglas que relacionan varios eventos, así, es posible indicar que sólo se envíe un evento si antes se ha generado otro u otros.

Es por medio del manejo de estas peticiones o eventos que el ORB es capaz de ejecutar una tarea de mayor prioridad o retrasar la ejecución de otra tarea menos importante. Es posible configurar a los consumidores y generadores en dos modos: push y pull. El modo push sirve para crear un objeto estático que reacciona a los eventos hasta que estos suceden. El modo pull sirve para que los objetos sean los que llevan el control del momento en el que se generan o se tratan de obtener eventos del canal.

La instanciación de este servicio toma en cuenta el determinismo necesario para garantizar la ejecución en tiempo real de las operaciones del canal.

El canal de eventos registra qué eventos generarán los objetos y qué objetos están interesados en estos eventos. Si un objeto se ha registrado para recibir un evento y después sufre una desconexión por algún motivo, los eventos dejan de enviarse, pero se siguen recibiendo sin que los generadores de eventos se enteren de que no están llegando a su destino. Si un generador de eventos se desconecta, los consumidores de eventos dejarán de recibirlos, pero no se enterarán de que el generador se desconectó. Esta desventaja se debe admitir para lograr un sistema en el que los consumidores no dependan directamente de los generadores y ambos puedan conectarse y desconectarse sin afectar al sistema.

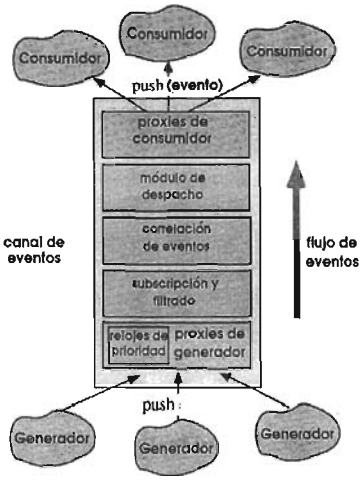


Figura 2.6 Módulos e interfases del canal de eventos (imagen tomada de (Gill, et al., 1998))

2.10 Algoritmos Genéticos

Los algoritmos genéticos (Kuri, et al., 2001) son métodos heurísticos de búsqueda inspirados en el proceso de evolución natural. La naturaleza ha sido capaz de generar organismos bien adaptados para desempeñarse en medios ambientes sumamente complejos y se han copiado sus métodos para resolver problemas y tratar de encontrar soluciones óptimas. La gran popularidad que han alcanzado los algoritmos genéticos, así como otras técnicas heurísticas, se debe, en buena parte, a que hacen posible abordar problemas en los que es muy difícil aplicar procedimientos matemáticos tradicionales. La gran plasticidad inherente a estos métodos, hace posible aplicarlos a la solución de muy diversos problemas sin hacer modificaciones sustanciales al procedimiento general, es decir, no suponen nada o casi nada acerca del problema a resolver, a diferencia de los métodos tradicionales en los que se exige que el modelo matemático del problema consista de una función con ciertas propiedades.

Los algoritmos genéticos fueron introducidos por John Holland en 1970 inspirándose en el proceso observado en la evolución natural de los seres vivos. Los biólogos han estudiado con detenimiento los mecanismos de la evolución, y aunque quedan partes por entender, muchos aspectos están bastante explicados. De manera muy general, en la evolución de los seres vivos el problema al que cada individuo se enfrenta cada día es la supervivencia. Para ello cuenta con las habilidades innatas provistas en su material genético. En el ámbito de los genes, el problema es buscar aquellas adaptaciones benéficas en un medio hostil y cambiante. Debido en parte a la selección natural, cada especie gana una cierta cantidad de "conocimiento", el cual es incorporado a la información de sus cromosomas.

Así pues, la evolución tiene lugar en los cromosomas, en donde está codificada la información del ser vivo. La información almacenada en el cromosoma varía de unas generaciones a otras. En el proceso de formación de un nuevo individuo, se combina la información cromosómica de los progenitores aunque la forma exacta en que se realiza es aún desconocida. Aunque muchos aspectos están todavía por discernir, existen principios generales ampliamente aceptados por la comunidad científica. Algunos de éstos son:

- La selección natural es el proceso por el que los cromosomas con "buenas estructuras" se reproducen más a menudo que los demás.
- En el proceso de reproducción tiene lugar la evolución mediante la combinación de los cromosomas de los progenitores. Se llama recombinación a este proceso en el que se forma el cromosoma del descendiente. También son de tener en cuenta las mutaciones que pueden alterar dichos códigos.
- La evolución biológica no tiene memoria en el sentido de que en la formación de los cromosomas únicamente se considera la información del período anterior.

Los algoritmos genéticos establecen una analogía entre el conjunto de soluciones de un problema y el conjunto de individuos de una población natural, codificando la información de cada solución en una cadena de bits a modo de cromosoma. Se introduce una función de evaluación de los cromosomas, que se conoce como función de adaptación y que está basada en la función objetivo del problema. Igualmente se introduce un mecanismo de selección de manera que los cromosomas con mejor evaluación sean escogidos para "reproducirse" más a menudo que los peores.

Los algoritmos genéticos están basados en integrar e instanciar eficientemente dos ideas fundamentales: Las representaciones simples como cadenas binarias de las soluciones del problema y la realización de transformaciones simples para modificar y mejorar estas representaciones. Para llevar a la práctica este esquema y concretarlo en un algoritmo, hay que especificar los siguientes elementos:

- · Una representación cromosómica
- · Una población inicial
- · Una medida de evaluación
- · Un criterio de selección / eliminación de cromosomas
- · Una operación de recombinación o cruza
- · Una operación de mutación

Las operaciones genéticas dependen del tipo de representación, por lo que la elección de una condiciona a la otra. La población inicial suele ser generada aleatoriamente, la evaluación de los cromosomas suele utilizar la calidad como medida de la bondad según el valor de la función objetivo en el que se puede añadir un factor de penalización para controlar la infactibilidad. Este factor puede ser estático o ajustarse dinámicamente.

La selección de los padres viene dada habitualmente mediante probabilidades según su adaptación. Uno

de los procedimientos más utilizado es el denominado de la ruleta en donde cada individuo tiene una sección circular de una ruleta que es directamente proporcional a su calidad. Para realizar una selección se realiza una tirada en la ruleta, tomando el individuo asociado a la casilla seleccionada.

Los operadores de cruza más utilizados son el de un punto, en donde se elige aleatoriamente un punto de ruptura en los padres y se intercambian sus bits, el de dos puntos, donde se eligen dos puntos de ruptura al azar para intercambiar y el uniforme, donde cada bit se elige al azar de un padre para que contribuya con su bit al del hijo, mientras que el segundo hijo recibe el bit del otro padre.

La operación de mutación más sencilla y una de la más utilizadas consiste en reemplazar con cierta probabilidad el valor de un bit. El papel que juega la mutación es el de introducir un factor de diversificación ya que, en ocasiones, la convergencia del procedimiento a buenas soluciones puede ser prematura por lo que se puede quedar atrapado en óptimos locales. Los óptimos locales son valores que son mínimos o máximos en una vecindad, pero que no lo son en todo el espacio de búsqueda. Otra forma obvia de introducir nuevos elementos en una población es recombinar elementos tomados al azar sin considerar su adaptación.

Generalmente existen múltiples criterios para evaluar a un individuo, en estos casos se utiliza una función de evaluación que es una combinación lineal de los diferentes criterios a los cuales se asocia un peso que puede ser ajustado para cumplir con ciertos compromisos (Kuri, 2003).

2.11 Resumen

Los antecedentes a los que se ha hecho referencia constituyen las herramientas básicas para el entendimiento y análisis del algoritmo y la infraestructura utilizada en este trabajo. Las referencias contienen material suficiente para profundizar en los temas que no se dominen y encontrar las definiciones o guías necesarias con respecto a sistemas distribuidos de tiempo real, algoritmos genéticos, CORBA, TAO u otro tema de interés relacionado con este documento.

Los conocimientos necesarios para entender la infraestructura de la programación orientada a objetos son los que pueden obtenerse en un curso de nivel avanzado de C++ y se recomienda el estudio de las librerías de ACE (Schmidt, 1994) para comprender con mayor facilidad los ejemplos que se encuentran en las referencias.

3. Propuesta del Algoritmo de Reconfiguración

El algoritmo se basa en el manejo de la calidad de servicio (QOS, Quality Of Service) de las tareas que deben atenderse. Según la especificación de las necesidades de cada tarea, se destinarán los recursos necesarios o se dará preferencia a éstas. La calidad del servicio indica qué ventaja tiene cada tarea con respecto a las demás o con qué importancia debe el sistema atender la ejecución de ésta. Esta indicación de calidad en la atención se forma por medio de parámetros como el nivel crítico, la importancia y la subprioridad dinámica que son transformadas por medio del algoritmo MUF en una sola prioridad que sirve para indicar al sistema operativo qué tarea debe atenderse en cada momento.

Para lograr construir un sistema distribuido de tiempo real basado en CORBA y realizar la reconfiguración en línea se ha diseñado la infraestructura de objetos necesaria y la simulación de un proceso de control de tiempo real para probar y medir los resultados. Las suposiciones o simplificaciones acerca de la naturaleza de las tareas son las siguientes:

- Son periódicas y se pueden interrumpir.
- Los tiempos de cambio de contexto se agregan al tiempo de ejecución requerido por cada tarea.
- Las tareas pueden ser o no dependientes entre sí.
- En cada momento, sólo una tarea utiliza los recursos de comunicación.

Los problemas de concurrencia son manejados por medio de varios mecanismos construidos en el ORB (Schmidt, et al., 1998), entre ellos se encuentran el canal de eventos, el uso de variables o banderas para administrar el uso de recursos, colas de atención a las tareas y otros mecanismos que forman parte de la estructura de TAO. Por medio de la administración y reserva de los recursos es que se logra planificar el orden en que se atienden las tareas y dar preferencia a las tareas críticas. El parámetro más importante de cada tarea es el nivel crítico y es principalmente sobre esta característica que se basa el cálculo de las prioridades de cada tarea.

El proceso de reconfiguración se divide en dos partes, en la primera se busca por medio de un algoritmo genético la configuración de tareas que mejor satisfaga los requerimientos de respuesta del sistema antes de iniciar el proceso de control. Esta primera parte es llamada fuera de línea porque sucede antes de que el sistema dinámico comience a funcionar. Una vez encontrada esta configuración se inician el proceso en línea de tiempo real, el reconfigurador, el planificador, el canal de eventos de tiempo real y un segundo algoritmo genético que se encargará de ajustar o reconfigurar las tareas según el comportamiento del sistema. El planificador, el reconfigurador, el canal de eventos y el segundo algoritmo genético son objetos de CORBA que existen en un mismo proceso. Todos los procesos corren concurrentemente.

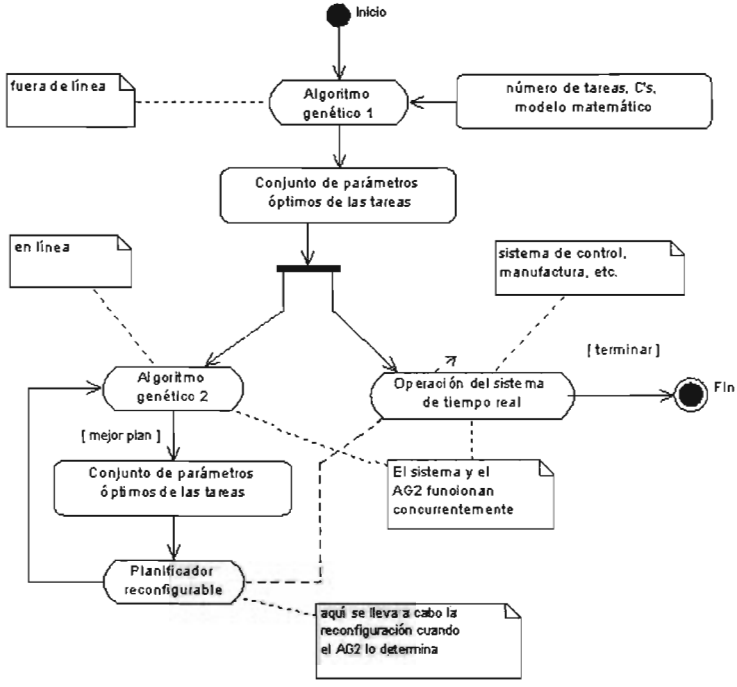


Figura 3.1 Diagrama de actividades de la reconfiguración

El procesamiento necesario para realizar la búsqueda de un mejor candidato para la reconfiguración se hace por medio de un proceso al cual se le asigna la menor prioridad de todas para que sólo realice la búsqueda cuando no se deban atender tareas de mayor importancia. El propósito es alterar el orden de atención a las tareas sin dejar de atender las peticiones recibidas. En la figura 3.1 se muestra un diagrama de actividades general de los pasos a seguir. En el diagrama se llama al algoritmo genético que funciona fuera de línea "algoritmo genético 1" y al algoritmo genético que funciona en línea se le llama "algoritmo genético 2". El planificador, el reconfigurador y el algoritmo genético 2 se muestran como dos tareas en el diagrama, aunque en realidad son tres procesos, pero es el reconfigurador el encargado de coordinarlos.

3.1 Arquitectura del Sistema

El sistema utiliza como base un ORB de tiempo real que se encargará de dar la infraestructura necesaria para los objetos CORBA. La utilización de CORBA se justifica porque es un estándar

maduro de uso generalizado y del que podemos obtener una instanciación de software libre y que podemos utilizar en un sistema operativo como Linux que es también de distribución libre. Todos los objetos utilizados han sido diseñados para tener tiempos de respuesta acotados y utilizar los servicios y facilidades de la definición de CORBA de tiempo real. Los objetos creados pueden ser sensores que se encargan de obtener datos de la planta, la planta que es un objeto que simula el comportamiento de un sistema dinámico, los actuadores que proporcionan las entradas a la planta y el controlador que se encarga de obtener datos de los sensores y de modificar el comportamiento de los actuadores. Los principales objetos CORBA creados en el sistema se describen de la siguiente manera:

- Sensores: objetos que se encargan de guardar los valores de las salidas de la planta. Periódicamente consultan a la planta para obtener estos valores.
- Actuadores: se encargan de enviar datos a la planta cada vez que son alterados por el controlador. Los valores enviados son las entradas de la planta.
- Controlador: objeto que se encarga de consultar periódicamente a los sensores y calcular las entradas que deben enviarse a la planta. Estos valores se envían por medio de los actuadores.
- Planta: objeto que simula la dinámica del sistema a reconfigurar. Periódicamente se hace el cálculo de las variables internas basándose en los valores anteriores y en las entradas.
- Planificador: objeto que recibe los datos de las tareas que participarán en el sistema. Con los datos que obtiene se calcula si un plan es factible y se configura el canal de eventos para comenzar a atender las tareas. Este objeto se puede configurar para utilizar MUF, MLF, EDF o RM. La clase que instancia al planificador es *RtecScheduler::Scheduler* definida en TAO.
- Reconfigurador: objeto que se encarga de coordinar al algoritmo genético en línea y el planificador. Cuando el algoritmo genético encuentra una solución que será utilizada en el sistema, el reconfigurador realiza las tareas necesarias para reprogramar el planificador. Una vez alterado el planificador, el canal de eventos comienza a actuar con base en los nuevos parámetros.
- Canal de eventos: objeto que se encarga de recibir y enviar eventos generados por los objetos CORBA. Recibe los eventos de los generadores y si tiene registrado algún objeto consumidor interesado en el tipo de evento recibido, lo envía. El envío y recepción de eventos se hace atendiendo a la prioridad calculada para cada operación. Cada envío y recepción de eventos es una tarea. La clase que instancia al canal de eventos es *RtecEventChannelAdmin::EventChannel* definida en TAO.

Los datos de cada tarea se organizan en estructuras del tipo *RT_Info* (Levine, et al., 1998) definidas en TAO y que sirven como datos de entrada para la interfase de configuración del planificador. En estas estructuras se puede guardar opcionalmente información acerca de las dependencias que existen entre las tareas a planificar. La figura 3.2 muestra esta estructura y la posibilidad que hay de crear dependencias entre ellas. Se muestran también como parte de la estructura el nivel crítico, el tiempo de ejecución, el período y la importancia o subprioridad estática. Todos los objetos CORBA deben registrar ante el planificador cada una de las tareas que realizan, excepto el planificador mismo y el canal de eventos.

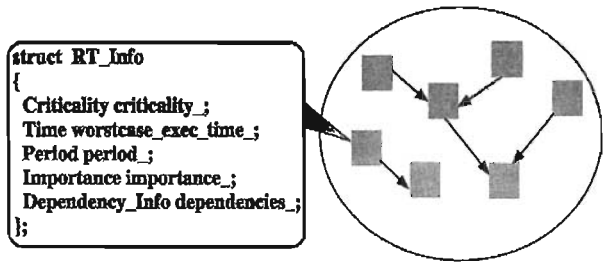


Figura 3.2 La estructura RT_Info (imagen tomada de (Gill, et al., 1998))

Estos objetos CORBA cooperan entre sí para controlar al sistema y reconfigurarlo. Toda la comunicación se realiza a través de la infraestructura que proporciona TAO. Los objetos pueden existir en una misma computadora o nodo o estar distribuidos en varios dentro de una red de comunicación (figura 3.3). Sin importar cómo estén distribuidos los objetos en la red de comunicación, la estructura del programa es la misma y no es necesario modificarlo si uno o más objetos cambian de nodo en la red. Incluso todos los objetos pueden existir en un mismo nodo sin utilizar la red de comunicación y seguir funcionando sin necesidad de hacer modificaciones.

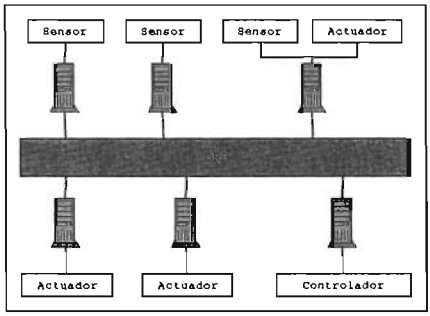


Figura 3.3 Comunicación entre objetos usando TAO

Al utilizar los objetos del planificador y el canal de eventos, se crearon los mecanismos necesarios para garantizar que la atención a tareas se lleve a cabo siguiendo las instrucciones de un planificador, de manera que las tareas con mayor importancia sean atendidas primero. El reconfigurador se encarga de coordinar las actividades del algoritmo genético en línea y de la reprogramación del planificador. Todos los objetos que forman parte del sistema se comunican por medio de eventos, un evento es un mensaje que se envía a uno o más objetos por medio del canal de eventos, que es quien se encarga de llevarlos al destino adecuado.

Cada objeto que participa en el envío y recepción de eventos pertenece a una clase que se deriva de los objetos *POA_RtecEventComm::PushConsumer* y *POA_RtecEventComm::PushSupplier* definidos en TAO, que contienen toda la infraestructura necesaria para conectarse al canal de eventos y enviar y

recibir estos eventos. Cada vez que un objeto quiere generar un evento, llama a la función *push()* del proxy al cual está conectado, con lo cual se disparan todas las actividades necesarias para hacer llegar el evento al canal de eventos y éste se encarga de llamar a la función *push()* de los objetos que están interesados, con lo cual se hace entrega de los datos y de la notificación al consumidor de eventos.

Si un elemento necesita enviar y recibir eventos, entonces se instancian dos objetos CORBA, uno para cada una de estas tareas. Los dos objetos se comunican entre sí directamente por estar en el mismo nodo y por existir en el mismo proceso, así que no hacen uso de los recursos de comunicación del sistema.

3.2 Proceso Fuera de Línea

Este procedimiento es considerado fuera de línea porque se realiza antes de que comience la operación del sistema de tiempo real. Al inicio se crea un ORB de tiempo real (*CORBA::ORB*) que servirá de base para manejar a todos los objetos que forman el sistema. Se crean también un planificador (*POA_RtecScheduler::Scheduler*) y un canal de eventos (*RtecEventChannelAdmin::EventChannel*). Se crean también dos objetos del tipo *AlgoritmoGenetico* para instanciar los dos algoritmos genéticos que se encargan de buscar la combinación de parámetros que producen un plan de atención de tareas que no provoca fallas debido a la falta de atención de alguna tarea crítica. Estos objetos reciben como parámetros de entrada el número de individuos que forman la población, el tamaño del genoma y el nombre de la función objetivo que utilizarán para evaluar a sus respectivos individuos. Después se crea el objeto de tipo *Reconfigurador* y se asocia con los dos objetos que encapsulan a los algoritmos genéticos, con el planificador y con el canal de eventos. El planificador y el canal de eventos se relacionan directamente por medio de un objeto del tipo *TAO_EC_Event_Channel_Attributes* definido en TAO que contiene varios parámetros con los que se puede modificar el comportamiento de estos dos objetos.

El algoritmo genético utiliza selección proporcional para escoger a los individuos con mejor calificación, realiza cruzamiento en un solo punto y maneja elitismo. La codificación se hace por medio de cadenas de 0's y 1's en notación pesada, lo cual es suficiente para representar a los individuos de este caso porque incluyen solo variables de tipo entero. La utilización de este tipo de algoritmo genético servirá para encontrar las soluciones necesarias sin problemas. Se utilizó este modelo sencillo para no complicar el algoritmo, aunque siempre se pueden hacer modificaciones para obtener resultados de manera más rápida.

Una vez que se tiene esta base, se declaran las tareas que participarán en el sistema. Primero se declaran los objetos que instancian los sensores, actuadores, planta y controlador y se conectan con el canal de eventos ya sea como generadores o consumidores de eventos, algunas veces con los dos papeles. En este punto no se da al sistema más información que los valores de los tiempos de ejecución (*C*) de cada tarea. Todos los demás parámetros serán calculados por el algoritmo genético. El parámetro *C* indica cuanto tiempo necesita una tarea para ejecutarse completamente, tomando en

cuenta el peor caso posible para garantizar la respuesta en tiempo real. Además se necesitan los siguientes datos:

- Número de tareas
- Modelo matemático del sistema de tiempo real.

El número total de tareas se obtiene contando el número de tareas declaradas ante el planificador y el modelo matemático de la planta está codificado en la función objetivo utilizada por los algoritmos genéticos. Cuando se tiene esta información se inicia el primer algoritmo genético. El procedimiento seguido por el algoritmo genético fuera de línea es el siguiente: se producen aleatoriamente varias combinaciones de los parámetros de las tareas que al unirse con los tiempos de ejecución (*C)* que ya se conocen completan la información necesaria para crear un plan de atención a las tareas que será calculado por el algoritmo MUF. Cada una de estas combinaciones es probada por medio del modelo del sistema y se les asigna una calificación que depende del error de estado estable del sistema, del porcentaje de utilización y del valor máximo y mínimo que puede tener un período. Si una combinación de parámetros no es factible entonces se le asigna una calificación baja. Al final se elegirán a los individuos más aptos y el mejor de ellos será utilizado para iniciar el sistema.

El algoritmo genético fuera de línea se basa en un algoritmo simple al que se le ha agregado elitismo para no perder información. La codificación del dominio utiliza un alfabeto binario (0 y 1) para representar los genomas de cada individuo, la evaluación de la población se hace con la ayuda de una función objetivo externa que depende de la aplicación. La selección se hace por medio del algoritmo de ruleta en el que se da mayor probabilidad a los individuos con mejor calificación de adaptación. La cruza se hace en un punto. Los parámetros de cruza y mutación son ajustables. El número de individuos y de generaciones durante las cuales se buscará la solución se pueden ajustar también. La función de adaptación utiliza el teorema de combinación lineal y la regla *K* (Kuri, 2003) para hacer que los individuos que no cumplan con alguna restricción obtengan una calificación baja, si por ejemplo un individuo no cumple con una o varias restricciones, se suma una cantidad negativa lo suficientemente grande para que sea considerado un individuo poco apto. Mientras menos restricciones se cumplan, más grande será el castigo para la calificación del individuo. El uso de la regla *K* y del teorema de combinación lineal es necesario porque la función de adaptación es una función multiobjetivo, es decir, no existe un criterio único sobre el cual buscar la solución, sino que existen varios que deben cumplirse a la vez. Cada restricción tiene un peso cuando se calcula la calificación total.

Cada vez que se tiene una propuesta de plan de configuración, el algoritmo genético puede calcular si es factible. Para este cálculo se necesitan los parámetros de tiempo de ejecución, período y nivel crítico (*C*, *T* y *NC*). (Levine, et al., 1998) Aún cuando un plan sea factible, se pueden alterar varios parámetros para cambiar la forma en la que se atiende a las tareas. Si se detecta que cierta tarea no tiene el tiempo suficiente para completar su ejecución, se reduce la calificación de la propuesta, si el comportamiento dinámico del sistema no es satisfactorio por tener un error de estado estable mayor al deseado, se asignará una calificación baja.

Existen cinco valores para el nivel crítico. El valor más alto indica que una tarea debe ser tratada con prioridad como una tarea de tiempo real, el valor más bajo indica que no es tan importante garantizar que una tarea cumpla siempre con su ejecución. Una solución propuesta obtendrá una calificación alta si garantiza que mas tareas de nivel alto reciban atención de tiempo real. La garantía se va otorgando de los niveles más altos a los más bajos. Mientras más niveles se garanticen, más tareas podrán cumplir con su ejecución. Un plan en el que se garantice que más niveles pueden ser atendidos en tiempo real obtendrá una mejor calificación.

Para cada tarea se calculan 3 parámetros necesarios para utilizar el algoritmo de planificación llamado MUF:

- Período de activación (*T*): indica cada cuanto tiempo se debe atender a una tarea periódica.
- Nivel crítico (*NC*): indica que tarea es más importante que otra. Siempre se dará preferencia a las tareas de mayor nivel crítico.
- Subprioridad estática (*SE*): cuando dos tareas tienen el mismo nivel crítico y la misma subprioridad dinámica, este parámetro indica a cual se atenderá primero. Este parámetro también es conocido como "importancia".

El cálculo de las tareas que pueden o no ser atendidas se lleva a cabo de la siguiente manera: Un plan es factible si y solo si todas las operaciones del conjunto crítico obtienen garantía de cumplir con sus límites de tiempo de ejecución. El conjunto crítico se identifica por el nivel crítico mínimo. Todas las operaciones que tienen una prioridad de atención mayor o igual a la del nivel crítico mínimo están en el conjunto crítico. La planificación de cada operación del conjunto crítico depende del peor caso de patrón de arribo de peticiones que es llamado el instante crítico (Gill, et al., 1998). El instante crítico de una operación ocurre cuando el retardo entre la petición y el fin de la ejecución es máximo. Si una operación es planificable en su instante crítico, se asegura su planificabilidad bajo cualquier otro patrón de arribo de las mismas operaciones. El tiempo límite para una operación en su instante crítico sucede exactamente en el instante crítico más su período *T*. Una operación debe ser capaz de completar su ejecución en este período utilizando sólo el tiempo que dejan libre otras operaciones de mayor prioridad. Todas las operaciones que tienen la misma prioridad de atención pero tienen tiempos límite en o antes del tiempo límite de la operación que se esté atendiendo en cualquier momento debe atenderse preferencialmente.

Para cada operación se realiza el siguiente análisis (Schmidt, et al., 1998): Llamamos a la operación sobre la cual se está realizando el análisis la operación "actual". El número de peticiones de atención durante el período de la operación actual de una operación con mayor prioridad está dado por $\lceil T_c / T_h \rceil$, donde T_c y T_h son los respectivos períodos de la operación actual y de la operación de mayor prioridad. El tiempo consumido por la operación de mayor prioridad durante el período de la operación actual está dado por la ecuación 3.1.

$$\lfloor T_c / T_h \rfloor * C_h + min(T_c - \lfloor T_c / T_h \rfloor * T_h, C_h) \qquad (3.1)$$

donde C_h es el tiempo de procesamiento utilizado para cada atención de la tarea de prioridad mayor. De manera similar, el número de ocurrencias de tiempos límite de otra operación que tiene la misma prioridad que la operación actual está dado por $\lfloor T_c / T_s \rfloor$, donde T_s es el período de la otra operación con la misma prioridad. El tiempo consumido por la otra operación está dado por $\lfloor T_c / T_s \rfloor * C_c$, donde C_c es el tiempo de procesamiento usado por la otra operación. Con estos datos es posible establecer que las operaciones con prioridad λ que se pueden planificar deben cumplir con la ecuación 3.2.

$$\forall \{ j, k \in S \mid (p(j) = \lambda) \wedge (p(k) >= \lambda) \}$$

$$\left(\left[\begin{array}{l} C_{wcpd(j)} + \sum\limits_{p(k)>=\lambda} \lfloor T_j / T_k \rfloor C_k + \\ \\ \sum\limits_{p(k)>\lambda} \min (T_j - \lfloor T_j / T_k \rfloor T_k , C_k) \end{array} \right] <= T_j \right) \qquad (3.2)$$

S es el conjunto de todas las operaciones en el plan. La función $p(j)$ regresa la prioridad asignada a la tarea j. j y k son dos operaciones que se comparan en cada momento. $C_{wcpd(j)}$ es el tiempo del peor caso de retardo por interrupciones de otras tareas para la tarea j. La operación j sufre un retardo por interrupciones si y sólo si tiene una petición mientras una operación de la misma prioridad que no tiene un tiempo límite dentro del período de la operación j se está ejecutando.

Además de realizar este cálculo, se establece el porcentaje de utilización del plan y se verifica que los períodos obtenidos para cada tarea estén dentro de un rango que pueda ser manejado por el sistema dinámico, es decir, que no sean períodos menores al lapso más pequeño que pueda generarse y que no sean mayores al lapso que garantiza que el sistema sigue siendo estable. El porcentaje de utilización se calcula por medio de la ecuación 3.3.

$$U = \Sigma \, C_i / T_i \qquad (3.3)$$

3.3 Proceso en Línea

Una vez que se tiene el mejor candidato para configurar el planificador, se inicia el proceso de tiempo real. Los valores encontrados para cada tarea son copiados al planificador y son reportados al canal de eventos. El reconfigurador recibe también una copia del mejor individuo para que sea tomado en cuenta en el algoritmo genético que seguirá buscando una mejor solución. Este proceso está controlado por el reconfigurador. El planificador recibe datos del algoritmo genético en línea que se encarga de buscar un mejor candidato. El planificador se encarga también de coordinar la ejecución de las distintas tareas y del algoritmo genético. El proceso del reconfigurador tiene la prioridad más alta de todo el sistema para

que se garantice la organización de las demás tareas y la posibilidad de realizar la reconfiguración en el menor tiempo posible. El proceso del algoritmo genético tiene el nivel crítico más bajo y la subprioridad estática más baja para que no interrumpa la ejecución de ninguna otra tarea.

Los datos generados por el algoritmo genético se registran en el planificador por medio de una llamada a las funciones *create()* y *set()* cuando se trata de un consumidor de eventos, esta función recibe como parámetros el nivel crítico, la subprioridad estática y el tiempo de ejecución. Cuando se trata de un generador de eventos se llama a *create()* y se declara un objeto del tipo *ACE_SupplierQOS_Factory* el cual guarda datos como el identificador del objeto y el tipo de evento. Después de este paso para cada tipo de objeto, se declaran las tareas en el canal de eventos por medio de las funciones *connect_push_supplier()* y *connect_push_consumer()*. Estas funciones reciben como parámetro el objeto donde se declararon las características de calidad de servicio (QOS) de la tarea. Cada objeto se conecta al canal de eventos como consumidor o generador de eventos, la conexión al canal se hace por medio de un objeto intermediario conocido como proxy que se encargará de manejar todas las comunicaciones y que está definido en la clase *RtecEventChannelAdmin::ProxyPushConsumer* cuando se necesita conectar un generador de eventos y *RtecEventChannelAdmin::ProxyPushSupplier* cuando se necesita conectar un consumidor de eventos. Un generador de eventos se conecta a un *ProxyPushConsumer* porque éste objeto representa al consumidor al cual están dirigidos los eventos, pero sólo es un intermediario. Lo mismo sucede con los consumidores de eventos. Estas clases están definidas en TAO. Cada objeto tiene su propio proxy. De esta manera se evita conectar dos objetos directamente, ninguno conoce la existencia de los otros y sólo pueden enviar eventos directamente al canal de eventos. El objeto del canal de eventos genera estos proxies por medio de dos interfases conocidas como *RtecEventChannelAdmin::SupplierAdmin* que genera los proxies para conectar generadores y *RtecEventChannelAdmin::ConsumerAdmin* que genera los proxies para conectar consumidores. Una vez que están conectados todos los objetos al canal de eventos, se llama a la función *compute_scheduling()* del planificador y a la función *activate()* del canal de eventos. Es entonces cuando todos los objetos comienzan a comunicarse entre sí.

Cuando el sistema comienza a funcionar se inicia también el proceso del reconfigurador. Este proceso se encarga de correr periódicamente el algoritmo genético en línea para buscar una solución que mejore el estado actual del sistema. Con respecto al algoritmo genético que funciona en línea, existen varias diferencias al compararlo con el primero:

- Si el problema de optimización es dinámico, la meta ya no es encontrar el extremo, sino seguir su progreso a través del espacio de soluciones lo mejor posible, asumiendo que los cambios son relativamente pequeños. Lo que quiere decir que ya no se está buscando una solución fija, sino que ésta cambiará a través del tiempo, pero estará relativamente cerca de la primer solución encontrada.
- Lo que busca este algoritmo son desviaciones de la configuración actual que generen errores en el sistema y trata de corregirlas utilizando cambios o mutaciones en la población de manera que la diversidad se mantenga constante pero sin alejarse demasiado de la solución anterior. La solución encontrada al inicio no será descartada por completo.

- Una población reducida de individuos (10 a 20) permitirá que el proceso de búsqueda sea rápido y suficientemente aceptable para un sistema de tiempo real. Como la búsqueda se hace periódicamente, una población pequeña permite que el tiempo de ejecución no interfiera con otras tareas más importantes.

El problema con la mayoría de los algoritmos genéticos es que convergen a un óptimo local y por lo tanto pierden la diversidad necesaria para explorar eficientemente el espacio de búsqueda y consecuentemente su habilidad para adaptarse a los cambios en el ambiente (Branke, 1999). Se puede lograr la búsqueda eficientemente incrementando la diversidad drásticamente, aumentando la mutación durante algunas generaciones, aumentando la mutación gradualmente cuando se detecta un cambio en el ambiente hasta que se encuentre una mejor solución, inicializando parte de la población con nuevos individuos o buscando entre las soluciones parecidas la mejor. El algoritmo genético en línea construido inicializa parte de la población con nuevos individuos continuamente.

Cada vez que se cumpla un período de reconfiguración se intentará cambiar el planificador si es que ya se cuenta con una mejor propuesta de solución. En la figura 3.3 se muestra un ejemplo considerando un período de 15 unidades de tiempo y una ejecución de 1 unidad de tiempo para el algoritmo genético (las cantidades se usan sólo como ejemplo).

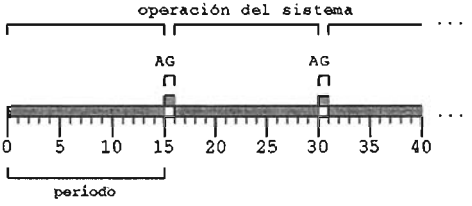


Figura 3.3 Diagrama de tiempos en la reconfiguración

Cada vez que se compara la solución actual con la que genera el algoritmo genético puede ser que se decida reconfigurar o no. Sólo se llevará a cabo la reconfiguración si se calcula obtener un beneficio en la estabilidad del sistema. Cuando se desea interrumpir todo el proceso se envían las señales apropiadas al planificador, al canal de eventos y al algoritmo genético en línea para terminar la ejecución.

La decisión de reconfigurar se toma considerando que el proceso puede causar retrasos en las tareas y afectar al sistema. No se considera que la reconfiguración pueda desestabilizar al sistema porque el tiempo necesario para realizar esta acción ya está contemplado en la planeación de las tareas. Sólo deberá reconfigurarse si los beneficios que se estima obtener son lo suficientemente grandes como para utilizar los recursos del sistema. Al hacer la reconfiguración no se descartan tareas que se estén ejecutando en ese momento, pero la prioridad que cada tarea tiene cambia debido a que sus propiedades de calidad de servicio han sido alteradas. Cualquier tarea que se esté ejecutando en un momento dado será interrumpida por otra de mayor prioridad, pero no será descartada.

La reconfiguración se hace alterando las características de las tareas registradas en el planificador por medio de la función *reset()*, la cual cambia los valores de nivel crítico, subprioridad estática, período y tiempo de ejecución. También se deben alterar los períodos de las tareas mismas para que se sincronicen con los declarados en el planificador. Después se llama a la función *recompute_scheduling()* para que estos cambios tengan efecto. Es en este paso donde se corre el riesgo de perder la ejecución de alguna tarea debido al uso de los recursos que se utilizan durante la reconfiguración.

3.4 Resumen

La reconfiguración de tiempo real significa actualizar continuamente las prioridades de las tareas en respuesta al comportamiento del sistema. Estas prioridades se calculan a partir de la especificación de calidad de servicio (QOS) de cada tarea que participa en el sistema. La especificación de calidad de servicio de cada tarea es calculada por medio de un algoritmo genético que califica las soluciones encontradas basándose en la respuesta del sistema dinámico y en el porcentaje de utilización de las tareas. Al encontrarse una respuesta que garantice un error menor al deseado y un plan factible, se inicia el funcionamiento del sistema con los parámetros encontrados. Al entrar en operación el sistema, se inicia también la búsqueda de una mejor solución basándose en los valores que se obtienen en tiempo real del sistema dinámico. Cuando se encuentra una solución que mejora el comportamiento del sistema, se hace la reconfiguración, esta consiste en cambiar las características de calidad de servicio de las tareas para que el planificador comience a tratarlas de manera diferente dando prioridad a las que obtuvieron mejores condiciones de atención. La reconfiguración se hace utilizando los objetos estándar que están definidos para CORBA de tiempo real, con lo cual la flexibilidad de mantenimiento que se logra por el uso de la infraestructura es aumentada debido a la capacidad de adaptación del sistema.

4. Caso de Estudio

El sistema a construir tiene la ventaja de poder adaptarse a distintos tipos de sistemas o aplicaciones en las cuales sea posible realizar la reconfiguración. Estos sistemas pueden ser de comunicaciones, control, manufactura, etc. El sistema que se eligió para probar la reconfiguración corresponde a una simulación de un sistema dinámico de segundo orden que representa a un péndulo invertido. Esta selección se justifica por las siguientes características:

- El sistema del péndulo necesita una respuesta de control de tiempo real que evite la pérdida del equilibrio, lo cual sucede en lapsos de 20 a 30 milisegundos. El sistema puede tolerar fallas ocasionales, por lo que es un buen ejemplo de tiempo real suave.
- El sistema del péndulo se puede dividir en módulos y éstos se pueden distribuir en una red como objetos CORBA.

El uso del sistema del péndulo tiene como objetivo probar la capacidad que tiene el sistema de reconfiguración y sólo sirve como ejemplo de un sistema cualquiera que puede ser adaptado para este fin. Debe tenerse claro que el sistema del péndulo no es una parte esencial de la infraestructura de reconfiguración que se está construyendo en este trabajo, sólo sirve como un ejemplo básico que puede cambiarse por otro.

El caso de estudio se centra en la reconfiguración de un sistema dinámico de segundo orden modificado para darle la característica de distribuido y de tiempo real (Sanz, et al., 2001A), (Sanz, et al., 2001B) con el objetivo de incrementar la capacidad del sistema de control aún cuando hayan pérdidas de elementos locales. La aplicación del algoritmo de planificación a este caso en particular podrá probar la capacidad del algoritmo genético para hacer más flexible el comportamiento del sistema en general y la posibilidad de construir un sistema de tiempo real utilizando herramientas básicas estándar disponibles libremente. Este capítulo presenta las características específicas del ejercicio elegido para construir y medir un sistema de tiempo real.

4.1 Infraestructura Utilizada

La ventaja que ofrece el uso de la infraestructura de CORBA es el aislamiento de los detalles relacionados con la comunicación entre diferentes computadoras o dispositivos dentro de una red. Al tratarse de un sistema distribuido, muchas veces se debe enfrentar un ambiente heterogéneo en el que no se cuenta con las mismas herramientas en los distintos dispositivos que conforman el sistema. En CORBA se utilizan diferentes tipos de computadoras, sistemas operativos y aplicaciones construidas con distintos lenguajes de programación en un sólo sistema que integra a todas sus partes. Esta unión se logra por medio de la imposición de una forma de comunicación en la cual se utilizan objetos para realizar llamadas remotas entre ellos.

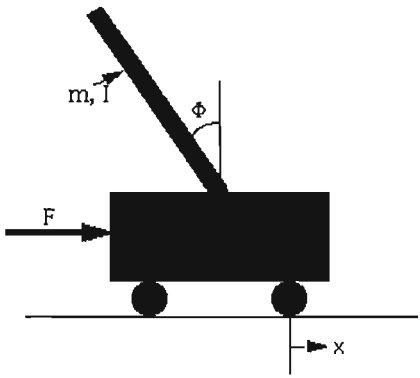
Aunque se incurre en una carga de memoria mayor por toda la infraestructura que debe configurarse e inicializarse antes de poder utilizar los servicios de CORBA, se obtiene un beneficio por la facilidad de crear nuevas aplicaciones y modificar lo ya construido con base en el manejo común de recursos de software. Para la construcción del sistema se utilizó la versión 5.4.2 de TAO y se probó en computadoras con procesador Athlon XP a 1.7 GHz. y 256 MB de RAM utilizando el sistema operativo Mandrake Linux 9.1 3.2.2-3mdk. El lenguaje de programación a utilizado es C++ y el compilador utilizado es gcc versión 3.2.2.

Aunque TAO tiene la capacidad de ordenar las tareas de tiempo real por medio del planificador y del canal de eventos, existen otros factores que afectan la respuesta del sistema. El uso de una red ethernet no garantiza que la transmisión de información se haga en un tiempo bien definido y el sistema operativo linux necesita modificaciones para dar el respaldo de tareas interrumpibles y el manejo de prioridades de tiempo real. Como el sistema se desarrolla utilizando C++, la utilización de métodos virtuales implica un costo mayor de ejecución (Harrison, et al., 1997), pero aún así el tiempo de respuesta está acotado. Para poder utilizar los niveles más altos de prioridad en el sistema operativo, todas las pruebas se corrieron usando el usuario root y utilizando el menor número de procesos posible. La interfase gráfica y la mayoría de los servicios se apagaron.

4.2 Modelo del Sistema Dinámico

El sistema en el cual se basa el ejercicio, representa un péndulo invertido que debe ser equilibrado sobre un carro (Messner, et al., 1997) (figura 4.1). Se eligió este sistema porque es posible construir una instanciación separando varios módulos que al interactuar entre sí a través de una red de comunicación simulan el comportamiento de este sistema.

Figura 4.1 Péndulo invertido sobre un carro



Con las ecuaciones dinámicas de movimiento y la representación en variables de estado se construyó una instanciación incluyendo un controlador en lazo cerrado (figura 4.2) para estabilizar el sistema. Se separaron los módulos principales del sistema y cada uno se convirtió en un objeto CORBA cuyas propiedades y métodos permiten simular el sistema en un entorno distribuido. El sistema se desestabiliza rápidamente si no se aplica el control de manera adecuada y por lo tanto resulta de utilidad para comprobar la capacidad de respuesta del sistema de tiempo real. En la instanciación se programaron actuadores, sensores, el sistema del péndulo al que llamaremos de ahora en adelante "la planta" y un controlador. El controlador recibe como entrada la diferencia entre la salida del ángulo del péndulo deseada y la que se está obteniendo.

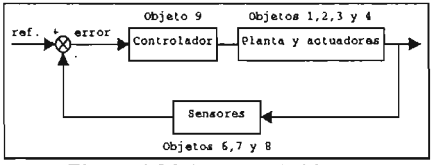


Figura 4.2 Diagrama de bloques

Tomando en cuenta la masa del carro, del péndulo, la fricción entre las ruedas y el piso, la longitud del péndulo, la fuerza de gravedad y la fuerza aplicada al carro podemos derivar las ecuaciones dinámicas del sistema y después la ecuación de transferencia. Los pasos necesarios para derivar las ecuaciones dinámicas de los diagramas de cuerpo libre se pueden consultar en la referencia (Messner, et al., 1997). El sistema no es lineal, pero se consideró así en un ángulo de 3.1416 radianes alrededor de la vertical para obtener un modelo más sencillo. Para utilizar un ejemplo en particular se establecen los siguientes valores:

M = masa del carro = 0.5 Kg

m = masa del péndulo = 0.2 Kg

b = fricción del carro = 0.1 N/ms

I = inercia del péndulo = 0.006 Kg * m^2

g = gravedad = 9.8 m/s²

l = longitud al centro de masa del péndulo = 0.3 m

u = fuerza aplicada al carro

X = posición del carro

ϕ = Ángulo del péndulo con respecto a la vertical

La ecuación de transferencia del sistema se muestra en la ecuación 4.1.

$$\frac{\Phi(s)}{U(s)} = \frac{\frac{ml}{q}s}{s^3 + \frac{b(I+ml^2)}{q}s^2 - \frac{(M+m)mgl}{q}s - \frac{bmgl}{q}} \tag{4.1}$$

donde: $q = [(M+m)(I+ml^2) - (ml)^2]$

El modelo en variables de estado se muestra en la ecuación 4.2.

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\phi} \\ \ddot{\phi} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{-(I+ml^2)b}{I(M+m)+Ml^2} & \frac{m^2gl^2}{I(M+m)+Ml^2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{-mlb}{I(M+m)+Ml^2} & \frac{mgl(M+m)}{I(M+m)+Ml^2} & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \phi \\ \dot{\phi} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{I+ml^2}{I(M+m)+Ml^2} \\ 0 \\ \frac{ml}{I(M+m)+Ml^2} \end{bmatrix} u$$

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \phi \\ \dot{\phi} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} u \tag{4.2}$$

El vector de salidas incluye a las variables X y ϕ y sus primeras derivadas. La variable X representa la posición del carro y ϕ el ángulo del péndulo con respecto a la vertical. Al cambiar el modelo a variables discretas, cada elemento se calcula ahora en función del período de muestreo. Si utilizamos un período de muestreo de 10 ms, encontramos la ecuación 4.3.

$$\begin{bmatrix} x(k) \\ \dot{x}(k+1) \\ \phi(k) \\ \dot{\phi}(k+1) \end{bmatrix} = \begin{bmatrix} 1 & 0.01 & 0.0001 & 0 \\ 0 & 0.9982 & 0.0267 & 0.0001 \\ 0 & 0 & 1.0016 & 0.01 \\ 0 & -0.0045 & 0.3119 & 1.0016 \end{bmatrix} \begin{bmatrix} x(k-1) \\ \dot{x}(k-1) \\ \phi(k-1) \\ \dot{\phi}(k-1) \end{bmatrix} + \begin{bmatrix} 0.0001 \\ 0.0182 \\ 0.0002 \\ 0.0454 \end{bmatrix} [u(k-1)] \quad (4.3)$$

$$y(k-1) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x(k-1) \\ \dot{x}(k) \\ \phi(k-1) \\ \dot{\phi}(k) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} [u(k-1)]$$

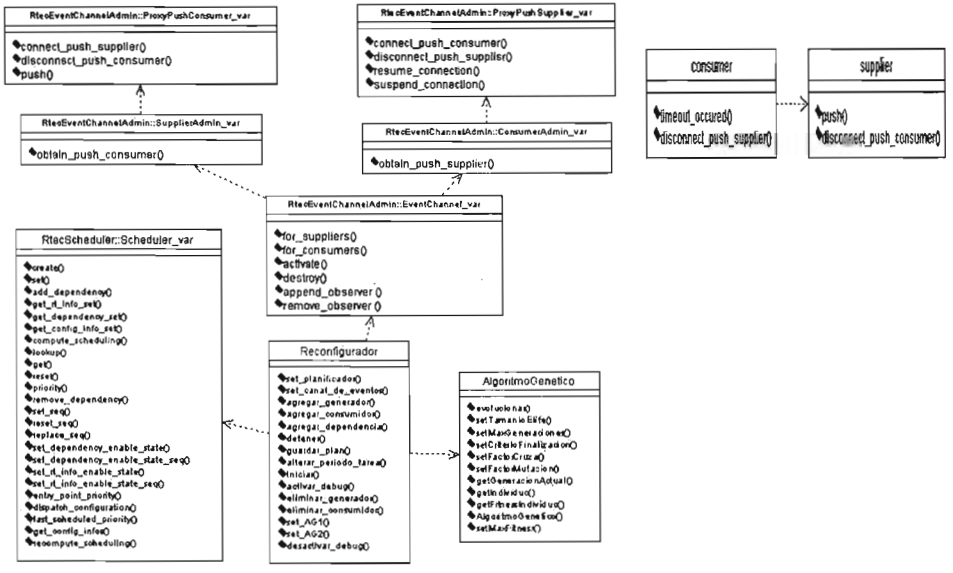
La discretización del sistema (Ogata, 1997) es necesaria para construir un modelo en un programa de computadora. Con este modelo es posible programar el comportamiento del sistema en función de los valores obtenidos cada vez que se toma una muestra de las salidas en los diferentes módulos. Las constantes que aparecen en las matrices de la ecuación 4.3 sólo son válidas para un período fijo de muestreo de 10 ms, pero en el funcionamiento real del sistema, este período dependerá del tiempo transcurrido entre cada muestreo. Los valores de las matrices dependen del tiempo real transcurrido entre un muestreo y otro (Lian, et al., 2002), (Walsh, et al., 2003).

Como el control de la planta se hará únicamente por medio del valor del ángulo del péndulo con respecto a la vertical, únicamente se necesita una entrada a la planta para controlarla. Para aumentar el número de tareas que debe soportar el sistema se generan tres entradas con el mismo valor y se obtiene el promedio. El sistema de control utilizado se describe en (Ogata, 1997).

4.3 Modelo de Objetos

Además de las clases que forman parte de TAO, se generó una infraestructura específica de objetos para este ejercicio. Las clases creadas se relacionan con el manejo de algoritmos genéticos, la reconfiguración y los módulos que tienen que ver con los sensores, actuadores, la planta y el controlador. La figura 4.3 muestra las principales clases utilizadas en el desarrollo del sistema. Aunque los sensores, los actuadores, la planta y el controlador son en principio muy diferentes, todos heredan de los objetos *POA_RtecEventComm::PushConsumer* y *POA_RtecEventComm::PushSupplier* para enviar y recibir eventos.

Figura 4.3 Diagrama de clases



A continuación se detallarán las clases construidas omitiendo la descripción de algunas funciones para enfocar el análisis en aquellas que tienen que ver con la comunicación entre objetos. La clase que instancia los sensores se muestra a continuación:

```
class sensor : public POA_RtecEventComm::PushSupplier
{
public:
  sensor ();
  // Constructor
  virtual void disconnect_push_supplier (ACE_ENV_SINGLE_ARG_DECL_NOT_USED)
    ACE_THROW_SPEC ((CORBA::SystemException));
  // The skeleton methods.
  void timeout_occured (ACE_ENV_SINGLE_ARG_DECL);
  int get_id (ACE_ENV_SINGLE_ARG_DECL);
  void set_valor (CORBA::Float valor ACE_ENV_ARG_DECL);
  void set_id(RtecEventComm::EventSourceID id);
  void set_proxy(RtecEventChannelAdmin::ProxyPushConsumer_ptr consumer_proxy);
private:
  RtecEventComm::EventSourceID id_;
  RtecEventChannelAdmin::ProxyPushConsumer_ptr consumer_proxy_;
  CORBA::Float valor_;
};
```

La interfase de la clase sensor tiene la función *set_valor()* que es llamada cuando se desea guardar el valor de una salida de la planta. En los datos miembro de la clase existe una variable de tipo real para guardar este valor.

La declaración de la clase para los actuadores es la siguiente:

```
class actuador : public POA_RtecEventComm::PushSupplier
{
public:
  actuador ();
  // Constructor
  virtual void disconnect_push_supplier (ACE_ENV_SINGLE_ARG_DECL_NOT_USED)
    ACE_THROW_SPEC ((CORBA::SystemException));
  // The skeleton methods.
  void timeout_occured (ACE_ENV_SINGLE_ARG_DECL);
  void set_valor (CORBA::Float valor ACE_ENV_ARG_DECL);
  void set_id(RtecEventComm::EventSourceID id);
  void set_proxy(RtecEventChannelAdmin::ProxyPushConsumer_ptr consumer_proxy);
private:
  RtecEventComm::EventSourceID id_;
  RtecEventChannelAdmin::ProxyPushConsumer_ptr consumer_proxy_;
  CORBA::Float valor_;
};
```

La clase actuador es parecida a la clase sensor, pero su comportamiento es diferente cuando interactúa con otros objetos. La función *set_valor()* es utilizada por el controlador para generar una entrada en la planta.

El controlador se declara de la siguiente forma:

```
class control : public POA_RtecEventComm::PushSupplier
{
public:
  control ();
  // Constructor
  virtual void disconnect_push_supplier (ACE_ENV_SINGLE_ARG_DECL_NOT_USED)
    ACE_THROW_SPEC ((CORBA::SystemException));
  // The skeleton methods.
  void timeout_occured (ACE_ENV_SINGLE_ARG_DECL);
  void set_entrada1 (float x ACE_ENV_ARG_PARAMETER);
  void set_entrada2 (float x ACE_ENV_ARG_PARAMETER);
  void set_entrada3 (float x ACE_ENV_ARG_PARAMETER);
  void set_entrada4 (float x ACE_ENV_ARG_PARAMETER);
```

```
  void set_id(RtecEventComm::EventSourceID id);
  void set_proxy(RtecEventChannelAdmin::ProxyPushConsumer_ptr consumer_proxy);
private:
  RtecEventComm::EventSourceID id_;
  RtecEventChannelAdmin::ProxyPushConsumer_ptr consumer_proxy_;
  CORBA::Float valor_[4][1];
  CORBA::Float U;
  CORBA::Float Nbar;
  CORBA::Float K[1][4];
  CORBA::Float Referencia;
};
```

La clase controlador tiene una interfase que permite controlar cuatro entradas que vienen de cuatro sensores conectados a la planta. Con estos valores y la matriz *K* se hace el cálculo necesario para obtener *U* que es la salida que se manda a los actuadores. La variable *Referencia* guarda el estado deseado de la variable a controlar.

La planta se declara de la siguiente forma:

```
class planta : public POA_RtecEventComm::PushSupplier
{
public:
  planta ();
  // Constructor
  virtual void disconnect_push_supplier (ACE_ENV_SINGLE_ARG_DECL_NOT_USED)
    ACE_THROW_SPEC ((CORBA::SystemException));
  // The skeleton methods.
  void timeout_occured (ACE_ENV_SINGLE_ARG_DECL);
  void set_entrada(int source, float valor ACE_ENV_ARG_PARAMETER);
  void set_id(RtecEventComm::EventSourceID id);
  void set_proxy(RtecEventChannelAdmin::ProxyPushConsumer_ptr consumer_proxy);
private:
  RtecEventComm::EventSourceID id_;
  RtecEventChannelAdmin::ProxyPushConsumer_ptr consumer_proxy_;
  // Las salidas son nuestras variables de estado, por lo tanto es correcto guardarlas
  CORBA::Float salida1_;
  CORBA::Float salida2_;
  CORBA::Float salida3_;
  CORBA::Float salida4_;
  // Las estradas sirven para hacer cálculos, las guardamos sólo por comodidad, pero no forman parte
  // esencial del estado actual del objeto.
  CORBA::Float entrada1_;
  CORBA::Float entrada2_;
  CORBA::Float entrada3_;
  // valores de las matrices del modelo de espacio de estados de la planta
```

```
  CORBA::Float A[4][4];
  CORBA::Float B[4][1];
  CORBA::Float C[2][4];
  CORBA::Float D[2][1];
  CORBA::Float U[1][1];
  CORBA::Float dX[4][1];
  CORBA::Float X[4][1];
  CORBA::Float Y[2][1];
  // para calcular el tiempo que ha pasado desde el último cálculo
  ACE_High_Res_Timer reloj;
  ACE_Time_Value lapso;
  CORBA::Float dt;
  CORBA::Float temp;
  int i,j,k,n,m,p;
};
```

La clase planta permite utilizar la función *set_entrada()* la cual sirve para manejar tres diferentes. Las variables miembro de la clase sirven para manejar el modelo discreto en variables de estado.

La infraestructura que instancia los algoritmos genéticos está compuesta de varias clases. La que se utiliza directamente es la clase *AlgoritmoGenetico*:

```
class AlgoritmoGenetico
{
  public:

      AlgoritmoGenetico(int tamanioPoblacion, int longitudGenoma,
                        float (*objetivo)(char *), float (*decodificador)(int, char*));
      ~AlgoritmoGenetico();
      void evolucionar ();
      void setTamanioElite (int tamanio);
      void setMaxGeneraciones (int maximo);
      void setCriterioFinalizacion (int criterio);
      void setFactorCruza (float pc);
      void setFactorMutacion (float pm);
      void setMaxFitness (float fitness);
      void setIndividuo (int posicion, char *nuevo);
      long getGeneracionActual();
      char *getIndividuo(int posicion);
      float getFitnessIndividuo (int posicion);
      float getVariable(int individuo, int posicion);
  private:
      Poblacion *poblacion1, *poblacion2;
      int tamanioElite;
      int tamanioPoblacion;
```

```
    int maxGeneraciones;
    float maxFitness;
    int criterioFinalizacion;
    float factorCruza;
    float factorMutacion;
    long generacionActual;
    int contadorNuevosIndividuos;
};
```

En esta clase es posible ver que el constructor sirve para establecer la cantidad de individuos de la población, el tamaño del genoma y la función objetivo. Por medio de otras funciones se pueden establecer también el tamaño de la élite, el número máximo de generaciones durante las cuales se hará la búsqueda, el criterio de finalización que sirve para indicar si se buscará durante un número de generaciones o hasta que se logre llegar a un cierto nivel de calificación para el mejor individuo. El factor de cruza y de mutación también se pueden establecer. También es posible obtener el genoma y la calificación del mejor individuo e introducir el genoma de un individuo a la población para ayudar o guiar al algoritmo genético cuando se requiera. Las otras clases creadas son *Poblacion* e *Individuo*.

4.4 Reconfiguración

Para encontrar un plan se necesitan las características de 7 tareas principales que son las que gobiernan a las demás heredándoles algunas de sus características. Cada tarea tiene un nivel crítico, una subprioridad estática, un tiempo de ejecución y un período. Una de estas tareas corresponde al reconfigurador y no se formará parte del genoma de los individuos debido a que los parámetros se establecen manualmente. Por lo tanto se tienen 18 variables que en conjunto representan un plan de ejecución. Con estos datos es posible iniciar el servicio de planificación utilizando el algoritmo MUF.

El algoritmo genético codifica estas variables en un genoma que tiene 3 bits para el *NC* y 3 para la *SE*. El período *T* utiliza 7 bits y por lo tanto el genoma es de (3+3+7)*6=78 bits. La función de adaptación realiza varias pruebas sobre el genoma y le asigna una calificación más alta mientras más pruebas pase. Las pruebas que realiza la función de adaptación son las siguientes:

1. Tamaño del período: El tamaño de *T* no debe ser menor a 10 ms ni mayor a 60 ms. 10 ms es el límite inferior que soporta el sistema operativo entre cada llamada a una tarea y 60 ms es el límite que soporta el sistema antes de volverse inestable.

2. Porcentaje de utilización: El porcentaje de utilización debe ser menor a 1.

3. Error de estado estable: La simulación del comportamiento del sistema en los primeros tres segundos debe dar como resultado un error de estado estable menor al 1%.

Como la evaluación es un problema multiobjetivo se utiliza la regla *K* (Kuri, 2003) que castiga a los individuos disminuyendo la calificación cada vez que no se cumple con alguna de las pruebas. El algoritmo genético usa elitismo para no perder las mejores soluciones encontradas y se pueden configurar la probabilidad de cruza, de mutación y el número de individuos que forman la élite. Una vez que se encuentra el mejor individuo, las variables obtenidas son utilizadas para configurar las tareas del sistema. Se puede elegir una de dos opciones para terminar la búsqueda del mejor individuo, por número de generaciones o por un nivel de calificación que debe alcanzarse. La población elegida es de 300 individuos y se corre el algoritmo hasta que la calificación obtenida sea mayor a cero, con lo cual se obtiene un individuo que cumple con todas las restricciones.

Los tiempos de ejecución *C* de cada proceso fueron calculados considerando el peor caso posible, que es el tiempo de mayor duración. La naturaleza de la aplicación genera diferencias en la duración de la misma tarea durante cada ejecución, por lo que se considera que el valor *C* corresponde al tiempo más grande.

Los valores obtenidos para cada tarea son los siguientes:

Tarea 1 – Sensor 1: 2 ms

Tarea 2 – Sensor 2: 2 ms

Tarea 3 – Sensor 3: 2 ms

Tarea 4 – Sensor 4: 2 ms

Tarea 5 – Controlador: 2 ms

Tarea 6 – Planta: 8 ms

Tarea 7 – Reconfigurador: 50 ms

Para los valores de los períodos (*T*) existe un máximo que es impuesto por el propio sistema, pero se puede buscar un valor menor que mejore la respuesta y que no lo sobrecargue haciendo que otros procesos no puedan ser atendidos. Para que el algoritmo genético sea capaz de encontrar la mejor opción, es necesario contar con un modelo de la planta a controlar y evaluar cómo afectará el nuevo plan a ésta. El objetivo del algoritmo genético es minimizar los períodos *T* sin afectar otras actividades que requieren de una respuesta de tiempo real. Minimizar los períodos *T* significa aumentar la frecuencia con que se leen o muestrean los valores de las variables de control y por lo tanto asignar más recursos a las tareas que realizan estas actividades. Si se les asignan recursos que son necesarios para otras actividades, entonces el sistema no funcionará correctamente. El valor de *T* de la tarea relacionada con la reconfiguración se fijó en 2 segundos, que es el período que pasará para que se intente reconfigurar.

Los valores de *C* y *T* están en milisegundos, pero los valores del *NC* y la *SE* son enteros y pueden

guardar los siguientes valores:

```
//(RtecScheduler::Criticality_t) [ VL_C = 0, L_C = 1, M_C = 2, H_C = 3, VH_C = 4]
//(RtecScheduler::Importance_t)  [ VL_I = 0, L_I = 1, M_I = 2, H_I = 3, VH_I = 4]
```

Los valores significan: Muy Bajo Nivel Crítico (VL_C, Very Low Criticality), Bajo Nivel Crítico (L_C, Low Criticality), Nivel Crítico Medio (M_C, Medium Criticality), Alto Nivel Crítico (H_C, High Criticality), Muy Alto Nivel Crítico (VH_C, Very High Criticality), Muy Baja Importancia (VL_I, Very Low Importance), Baja Importancia (L_I, Low Importance), Importancia Media (M_C, Medium Importance), Alta Importancia (H_I, High Importance), Muy Alta Importancia (VH_I, Very High Importance). Mientras más bajo sea el valor de una de estas constantes, más baja es la prioridad que obtienen las tareas con las que están asociadas. Estas constantes están declaradas en TAO y se utilizan al momento de declarar una tarea ante el planificador.

Se pueden calcular los períodos *(T)* de manera que se logre un mejor control del sistema sin saturar la capacidad de procesamiento. Cuando se diseña el controlador se elige una frecuencia de muestreo para los elementos como los sensores y actuadores. Esta elección debe considerar la frecuencia mínima para no perder información (el doble de la frecuencia máxima de la señal muestreada) y la máxima para no superar la capacidad de procesamiento. La frecuencia mínima es un dato que debe darse al algoritmo genético. La frecuencia máxima será resultado de la capacidad de procesamiento del sistema. Así que, como resultado del cálculo fuera de línea del algoritmo genético, se obtiene un grupo de planes que atienden a todas las tareas con la mayor cantidad de recursos posibles.

El algoritmo genético que funciona en línea utiliza una población de 20 individuos para poder evolucionar toda una generación cada vez que tiene posibilidad de utilizar recursos y no interferir con las demás tareas. Al crear una población pequeña, el tiempo necesario para hacer una revisión de todos los individuos y encontrar alguno que sea mejor que el actual se reduce y se tiene un mayor control de la búsqueda. Cada vez que se llama a este algoritmo genético se evoluciona una generación y se incluyen nuevos individuos para conservar la diversidad. La élite de esta población es de sólo dos individuos. Las condicion inicial del algoritmo genético es una población aleatoria con un elemento obtenido del algoritmo genético fuera de línea.

En el primer algoritmo genético se utiliza la opción de buscar la solución durante un número de generaciones desconocida siempre y cuando no se alcance un nivel de calificación indicado, porque no se corre el peligro de retrasar ninguna otra tarea cuando el sistema aún no entra en funcionamiento, pero en el algoritmo que corre en línea se acota el tiempo de ejecución indicando durante cuantas generaciones se debe hacer la búsqueda. Cuando este algoritmo reinicia la búsqueda, no se pierde lo que ya se ha encontrado antes.

4.5 Resumen

El sistema utilizado es un péndulo invertido sobre un carro, que es representado mediante un sistema de ecuaciones de segundo orden y que puede ser dividido para simular un sistema distribuido de tiempo real. El sistema es dividido en sensores, actuadores, un controlador y un sistema principal o planta que encierra el comportamiento dinámico del sistema. Cada parte del sistema se representa por medio de un objeto CORBA capaz de enviar y recibir información a los otros objetos por medio del canal de eventos. La información se envía en forma de eventos y no está dirigida a un objeto en especial, sino que se utiliza el modelo publicista-subscriptor en el que los objetos que están interesados en cierto tipo de evento, se subscriban al canal de eventos para recibirlos.

Junto con los objetos CORBA y el canal de eventos se activa un planificador y un reconfigurador, ambos instanciados como objetos CORBA y que se encargan de calcular la prioridad de ejecución de cada tarea y de cambiar las características de calidad de servicio de las mismas respectivamente. El reconfigurador utiliza un algoritmo genético para encontrar los parámetros de cada tarea que hacen que el sistema se comporte de manera estable y se disminuya el error obtenido en las salidas.

El algoritmo genético utiliza individuos cuyo genoma contiene codificadas las variables necesarias para representar el nivel crítico, la subprioridad estática y el período de ejecución de 7 tareas básicas de las que se desprenden 28 diferentes tareas que realizan las actividades del sistema en su totalidad.

5. Resultados

Los resultados se obtienen de la capacidad del sistema para funcionar de manera continua cuando se realiza la reconfiguración. Un resultado interesante que se espera obtener es la magnitud de los tiempos que se generan por utilizar una infraestructura como CORBA. Las pruebas consisten en ejecutar el sistema y obtener datos con respecto a los retardos cada vez que se hace un cambio de plan poniendo atención especial al lapso en el que se realiza este cambio. También se evalúa la respuesta del sistema dentro de ciertos límites de error aún cuando se presenten fallas locales o debido a la comunicación, es decir, que el error mostrado en la figura 5.1 se mantenga dentro de un cierto porcentaje. Dicha verificación se lleva a cabo mediante el cálculo del error del caso de estudio el cual debe ser limitado.

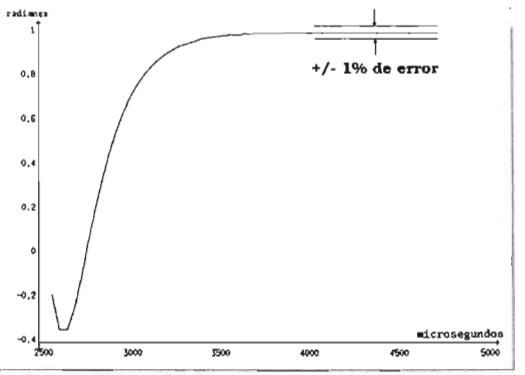


Figura 5.1 Error de estado estable.

Las mediciones se hacen por medio de la función *gettimeofday()* con la que obtenemos el momento en que inicia y termina cada tarea. Con estos datos se pueden hacer los cálculos necesarios para conocer la duración de las tareas. La función *gettimeofday()* tiene resolución de microsegundos y puede dar una imagen muy clara de lo que sucede en muy poco tiempo de ejecución del sistema. El cálculo inicial de cuánto tarda cada tarea se hizo del mismo modo. El primer individuo encontrado después de varias pruebas se describe a continuación junto con los valores que se obtienen de él:

```
El mejor individuo obtuvo un 5666.041504.
0000010000010000001000011011001011100000010010001110101110110001010011100111O1
T[0]=10 C[0]=2 NC[0]=1 SE[0]=3
T[1]=10 C[1]=1 NC[1]=1 SE[1]=0
T[2]=10 C[2]=1 NC[2]=0 SE[2]=4
T[3]=30 C[3]=1 NC[3]=4 SE[3]=1
T[4]=30 C[4]=1 NC[4]=2 SE[4]=4
T[5]=10 C[5]=3 NC[5]=4 SE[5]=3
T[6]=1000 C[6]=20 NC[6]=4 SE[6]=4
El porcentaje de utilización es de 0.786667
```

Esta es la salida del programa y puede verse el porcentaje de utilización es de 0.78. La calificación del individuo es de 5666 y la cadena de unos y ceros es el genoma del individuo ganador. Una vez que se encontró al mejor individuo, se puede iniciar el sistema introduciendo este valor en el algoritmo genético si no se desea comenzar la búsqueda desde cero. Esta práctica se llevo a cabo en algunas pruebas para ganar tiempo y es la indicada para iniciar un sistema en producción.

Al medir el comportamiento del sistema dinámico después de encontrar la primer solución por medio del algoritmo genético se obtiene la respuesta de la figura 5.2:

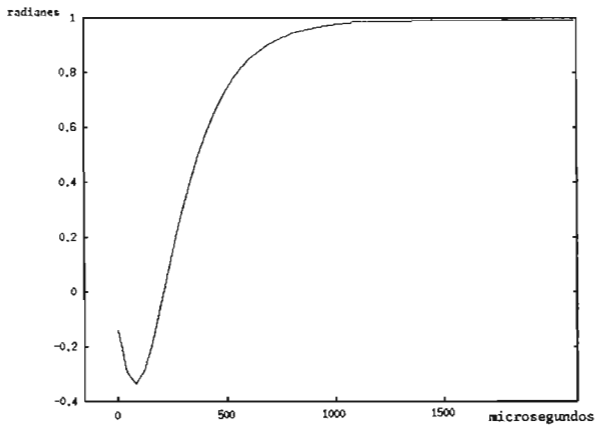


Figura 5.2: Respuesta del sistema dinámico.

Cuando el algoritmo genético encuentra a un individuo con calificación aceptable, se detiene la búsqueda y comienza la ejecución del sistema dinámico. El tiempo de asentamiento es de 1 segundo y el error de estado estable es de 0.7%. Aproximadamente el 3% de los individuos tiene una respuesta parecida a la del mejor. Los individuos que se consideran aptos como posible respuesta tienen una calificación mayor a cero, lo que indica que cumplen con todas las restricciones de la función objetivo. La figura 5.3 muestra 3 respuestas correspondientes a individuos de menor calificación. Los tres individuos obtuvieron una calificación negativa, lo cual lleva a la inestabilidad del sistema.

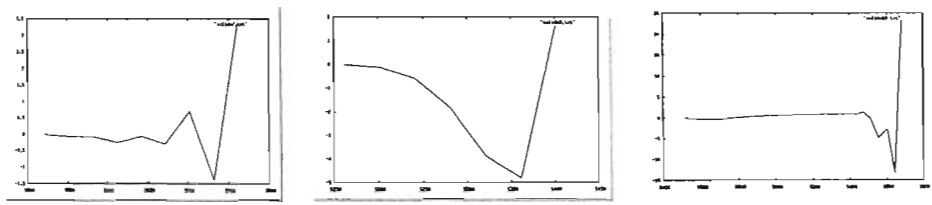


Figura 5.3: Respuestas de soluciones no aptas.

La reconfiguración en línea tiene dos partes, en la primera, el algoritmo genético busca una solución avanzando 2 generaciones. En la segunda parte se decide si se reconfigura o no dependiendo de la solución encontrada. La primer parte del proceso se corre con la menor prioridad posible para no interrumpir otras tareas. La reconfiguración se intenta cada segundo para dar oportunidad al sistema de rectificar su estado cuando se utiliza algún plan que no es adecuado para la dinámica del sistema. La primer parte de la reconfiguración, que es la búsqueda que hace el algoritmo genético puede tardar hasta 550 ms debido a las constantes interrupciones de las tareas de mayor prioridad. El porcentaje de utilización del tiempo por parte de la las tareas de la segunda parte de la reconfiguración es de 0.4% si tomamos el peor caso posible, lo cual ocurre si siempre se reconfigura el sistema. El sistema no se reconfigura a menos que se haya encontrado una solución mejor a la actual.

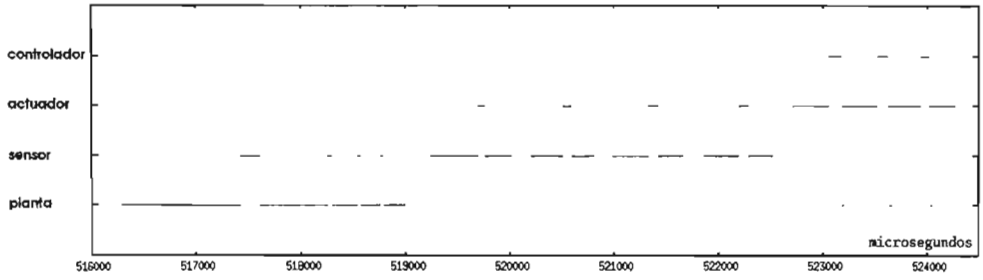


Figura 5.4: Ejecución de las tareas

La figura 5.4 muestra cómo se van ejecutando las distintas tareas del sistema. El eje horizontal representa el tiempo en microsegundos. El eje vertical representa a las tareas que corresponden a la planta, a los sensores, el controlador y los actuadores. La planta hace llamadas a los objetos sensores, estos a su vez llaman al controlador. Después el controlador llama a los actuadores y estos a la planta. Todo esto sucede en 10 ms. Después el ciclo se vuelve a repetir. La figura 5.5 muestra a las tareas en un lapso mayor en el que puede verse que cada 10 ms la planta, dos de los sensores y el controlador se ejecutan. Los otros dos sensores se ejecutan cada 30 ms, por lo que antes de la ejecución de la planta se puede ver una línea que corresponde a esos sensores cada 3 ciclos de 10 ms.

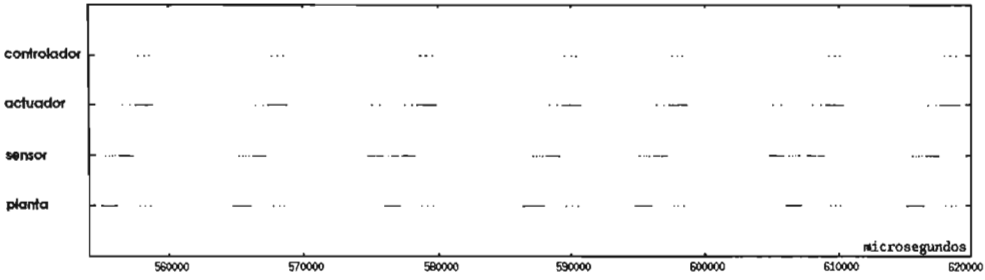


Figura 5.5 Ejecución de las tareas durante 60 milisegundos

Con respecto a la reconfiguración del sistema, las medidas tomadas arrojan un tiempo de 4 ms para ejecutar las funciones necesarias. En el comportamiento del sistema no se ve ninguna falla o retraso debido a esta acción, debido a la corta duración de las operaciones necesarias para alterar los datos de las tareas, además de que este tiempo está considerado en el planificador y por lo tanto está previsto su utilización. En la figura 5.6 se muestra la ejecución de las tareas cuando sucede una reconfiguración.

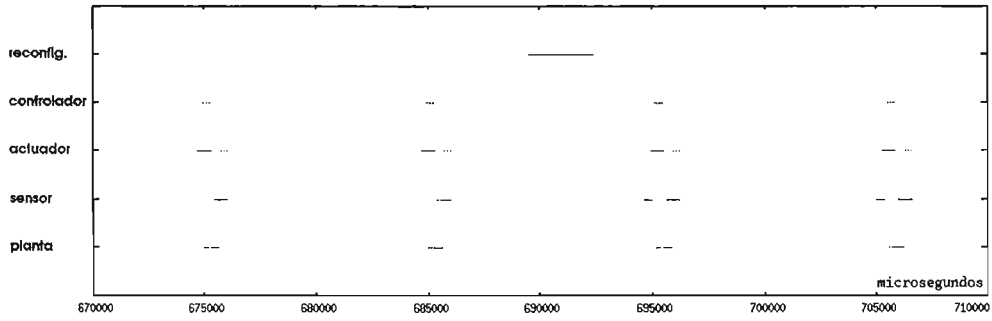


Figura 5.6 Ejecución de tareas durante la reconfiguración

Cuando se lleva a cabo la reconfiguración, cada tarea comienza a ejecutarse con su nuevo período hasta que termina la ejecución que ya había sido planificada hasta ese momento. La reconfiguración de la figura 5.6 se llevó a cabo con el siguiente individuo encontrado por el algoritmo genético.

```
AG 2 -  7039.3 00010100000010000010100010100000101000001001111100010111010101111101100011100
T[0]=10 C[0]=2 NC[0]=2 SE[0]=4
T[1]=10 C[1]=1 NC[1]=3 SE[1]=3
T[2]=10 C[2]=1 NC[2]=4 SE[2]=4
T[3]=10 C[3]=1 NC[3]=0 SE[3]=3
T[4]=10 C[4]=1 NC[4]=4 SE[4]=0
T[5]=10 C[5]=3 NC[5]=4 SE[5]=3
T[6]=1000 C[6]=20 NC[6]=4 SE[6]=4
El nuevo porcentaje de utilizacion es de 0.920000
```

La nueva calificación es de 7039 y lo que cambió es el período de las tareas 3 y 4. Como puede verse en la figura 5.6, la reconfiguración se lleva a cabo en el tiempo libre entre un grupo de llamadas y otras, por lo que no afecta al sistema. Después de reconfigurar, el sistema sigue estable y funcionando normalmente. El principal limitante con respecto a los períodos que se pueden manejar en el sistema lo da el sistema operativo, pues no es posible crear períodos de menos de 10 milisegundos. Las tareas pueden durar ejecutándose menos de un milisegundo y entre una y otra los tiempos de cambio de contexto son de 20 μs, los cuales pueden despreciarse o sumarse a los tiempos de ejecución cuando sea posible o necesario.

6. Conclusiones

6.1 Conclusiones Generales

El presente trabajo desarrolla un método para lograr la reconfiguración en línea de un proceso dinámico con base en un sistema de cómputo de tiempo real. Dicho método aprovecha las ventajas de utilizar sistemas de apoyo (middleware) para efectuar una instanciación acorde a la complejidad del problema con la capacidad de utilizar una herramienta para evitar conflictos en el manejo de los recursos del sistema.

En este trabajo se logra modificar el comportamiento de un sistema distribuido de tiempo real por medio de la reasignación de recursos a las diferentes tareas que compiten por su uso. Al considerarse varios nodos capaces de procesar información que se comunican constantemente entre sí, el principal recurso es la red de comunicaciones, por lo que, por medio de la administración de este recurso, se logra imponer un orden en todo el sistema. El elemento que sirvió como árbitro entre las diferentes tareas es el canal de eventos de tiempo real y aunque en casos extremos de tránsito en la red puede comportarse como un cuello de botella, resultó suficiente para el caso de estudio. La modificación del comportamiento del sistema se logra cambiando las características de cada tarea que indican su importancia para ser atendidas antes que cualquier otra. Esta modificación es la que llamamos reconfiguración y es en línea porque al momento de realizarse, el sistema está funcionando y no se detiene.

La reconfiguración en línea tiene como fin la modificación del estado actual del sistema sin la intervención de un operador para lograr un funcionamiento continuo dentro de los parámetros establecidos o para mejorar el estado actual. No es motivo de este estudio lograr una adaptación inmediata y continua del sistema relacionada con el medio ambiente o con las circunstancias que lo afectan, como pueden ser las posibles fallas de comunicación o internas. El estudio se centra en el proceso de alterar el sistema por medio del manejo de los recursos disponibles. La búsqueda en línea tiene como fin mejorar la respuesta que se encontró al inicio para probar el comportamiento al momento de cambiar las características del sistema.

La manera en la que se genera el plan para organizar las tareas se basó en MUF por ofrecer las ventajas de los planificadores estáticos y dinámicos. Teniendo como base esta forma de organizar las tareas, se buscaron los parámetros de cada una de ellas de manera que la salida del sistema fuera la más estable posible. La estrategia incluye un planificador semidinámico para lograr adaptarse a los cambios provocados por las duraciones variables de las tareas y un algoritmo genético que permite buscar las mejores condiciones que garanticen la respuesta de tiempo real. Con esta combinación se logra construir un sistema muy flexible y al mismo tiempo fácil de modificar.

Con respecto al uso de algoritmos genéticos, con ellos se obtiene la ventaja de que el cálculo de los parámetros para las tareas no está ligado directamente con una aplicación en particular, con esto, se logra facilitar el mantenimiento y modificación del sistema por no existir dependencias entre los módulos del sistema en sí y de la parte encargada de hacer el cálculo inicial y la reconfiguración. Al realizar el cálculo de los parámetros para iniciar el sistema, se cumplieron con los requerimientos registrados y al hacer la reconfiguración en línea se conservaron estos requerimientos. El sistema logró mantener su funcionamiento después de la reconfiguración del orden de ejecución de las tareas, incluyendo el consumo de recursos por parte del algoritmo genético para buscar nuevos planes de atención a las tareas.

El parámetro principal en el que se basa la reconfiguración es el error de estado estable que debe obtenerse en la salida del sistema. Del trabajo realizado se concluye que es posible integrar el uso de algoritmos genéticos en un sistema de tiempo real si éste no depende de la respuesta del algoritmo genético en tiempos específicos.

La modificación del algoritmo genético para realizar la búsqueda en un tiempo menor puede ayudar a reducir el tiempo de búsqueda inicial y a reducir el tiempo necesario para encontrar una solución mejor cuando el sistema está funcionando, pero no servirá para garantizar la respuesta de tiempo real, porque el momento en el cual se encuentra la respuesta no está acotado.

Se verificó también que la utilización de TAO en la construcción de un sistema distribuido de tiempo real facilita su diseño y mantenimiento al aislar los detalles de comunicación entre los diferentes nodos del sistema. La utilización de las especificaciones de CORBA facilita además las tareas de extensión pues es más sencillo generar nuevos módulos y conectarlos con los ya existentes.

Con respecto a la capacidad de manejo de tiempo real por medio de una computadora personal con el sistema operativo Linux, una red ethernet y CORBA, se comprobó que se pueden manejar tiempos de hasta 10 milisegundos en los diferentes períodos de las tareas sin tener pérdidas que puedan afectar continuamente el funcionamiento del sistema. Esto es, por supuesto, dependiente del hardware utilizado y del ambiente en el que se hagan las pruebas. Todas las pruebas que se hicieron no tuvieron interferencia de otras aplicaciones ni de intentos de comunicación por parte de otros nodos en la red utilizada.

Las principales aportaciones de este trabajo se hacen en el análisis de la capacidad de reconfigurar un sistema utilizando el canal de eventos de tiempo real y su aplicación en un sistema que demanda tiempos de respuesta del orden de milisegundos para lograr un funcionamiento correcto. También se probó la capacidad de encontrar una solución al problema de planificación por medio de un algoritmo genético.

6.2 Trabajo Futuro

El trabajo futuro se relaciona con la adaptación de la búsqueda en línea para que sea capaz de reaccionar a los cambios del medio ambiente que afecten de manera negativa el desempeño del sistema, También se deben hacer pruebas utilizando un sistema operativo que garantice respuestas de tiempo real y una red de comunicaciones que garantice tiempos de respuesta acotados.

La utilización de un algoritmo genético permite en muchos casos trabajar con sistemas en los que no se conoce la dinámica interna, de manera que es el algoritmo genético el que adapta las entradas por medio de la información que obtiene del comportamiento del sistema. En el caso de este trabajo, es necesario conocer la dinámica del sistema para hacer simulaciones tomando en cuenta que en un sistema real sería costoso o no aceptable hacer funcionar el sistema mientras el algoritmo genético no ha encontrado una respuesta adecuada. Sin embargo, la reconfiguración en línea puede modificarse para hacer estas pruebas porque los individuos que debe probar son relativamente buenos y producen un comportamiento aceptable dentro de los parámetros de control y tiempos de respuesta. Esta modificación generará el trabajo futuro de este proyecto.

7. Apéndice A – Código Fuente

Este apéndice incluye el código fuente de los dos principales archivos que forman parte del sistema. Los archivos son servicio.cpp y Reconfigurador.cpp. El primer archivo es el programa principal y entre las clases que utiliza se encuentra Reconfigurador.cpp. En ésta clase se encuentran los principales métodos utilizados para hacer la reconfiguración en línea del sistema. Se pueden analizar estos programas siguiendo el algoritmo de reconfiguración explicado en el capítulo 3.

servicio.cpp

```
//Servicio.cpp 1.0 10/01/2005 22:46:14 Saavedra

#include "orbsvcs/Sched/Reconfig_Scheduler.h"
#include "orbsvcs/Runtime_Scheduler.h"
#include "orbsvcs/Event_Service_Constants.h"
#include "orbsvcs/Event_Utilities.h"
#include "orbsvcs/Scheduler_Factory.h"
#include "orbsvcs/Event/EC_Event_Channel.h"
#include "orbsvcs/Event/EC_Default_Factory.h"
#include "orbsvcs/Event/EC_Kokyu_Factory.h"

#include "ace/Get_Opt.h"
#include "ace/Sched_Params.h"
#include "ace/Auto_Ptr.h"
#include "ace/SString.h"
#include "ace/OS_NS_strings.h"
#include "ace/Thread.h"

#include "funcionesAG.cpp"
#include "AG/AlgoritmoGenetico.cpp"

#include "AG2.cpp"
#include "Reconfigurador.cpp"

#include "planta.h"
#include "sensor.h"
#include "actuador.h"
#include "control.h"

// ****************************************************************

namespace
{
  int config_run = 0;
  //ACE_CString sched_type ="rms";
  ACE_CString sched_type ="muf"; //esto lo agregué para no dar la opción en la línea de comando
}

int parse_args (int argc, char *argv[]);

typedef TAO_Reconfig_Scheduler<TAO_RMS_FAIR_Reconfig_Sched_Strategy, TAO_SYNCH_MUTEX> RECONFIG_RMS_SCHED_TYPE;

typedef TAO_Reconfig_Scheduler<TAO_MUF_FAIR_Reconfig_Sched_Strategy, TAO_SYNCH_MUTEX> RECONFIG_MUF_SCHED_TYPE;

// ****************************************************************

// ****************************************************************

int main (int argc, char* argv[])
{

  // Utilizando Kokyu
  TAO_EC_Kokyu_Factory::init_svcs ();

  ACE_DECLARE_NEW_CORBA_ENV;
  ACE_TRY
    {
      // ORB initialization boiler plate...
      CORBA::ORB_var orb =
        CORBA::ORB_init (argc, argv, "" ACE_ENV_ARG_PARAMETER);
      ACE_TRY_CHECK;

      if (parse_args (argc, argv) == -1)
        {
          ACE_ERROR ((LM_ERROR,
                      "Parametros incorrectos\n"));
          return 1;
        }
```

```
CORBA::Object_var object =
  orb->resolve_initial_references ("RootPOA" ACE_ENV_ARG_PARAMETER);
ACE_TRY_CHECK;
PortableServer::POA_var poa =
  PortableServer::POA::_narrow (object.in () ACE_ENV_ARG_PARAMETER);
ACE_TRY_CHECK;
PortableServer::POAManager_var poa_manager =
  poa->the_POAManager (ACE_ENV_SINGLE_ARG_PARAMETER);
ACE_TRY_CHECK;
poa_manager->activate (ACE_ENV_SINGLE_ARG_PARAMETER);
ACE_TRY_CHECK;

// *********************************************************

// Crea un planificador
POA_RtecScheduler::Scheduler* sched_impl = 0;

if (ACE_OS::strcasecmp(sched_type.c_str(), "rms") == 0)
  {
    ACE_DEBUG ((LM_DEBUG, "Creando un planificador RMS\n"));
    ACE_NEW_RETURN (sched_impl,
                    RECONFIG_RMS_SCHED_TYPE,
                    1);
  }
else if (ACE_OS::strcasecmp(sched_type.c_str(), "muf") == 0)
  {
    ACE_DEBUG ((LM_DEBUG, "Creando un planificador MUF\n"));
    ACE_NEW_RETURN (sched_impl,
                    RECONFIG_MUF_SCHED_TYPE,
                    1);
  }

RtecScheduler::Scheduler_var scheduler =
  sched_impl->_this (ACE_ENV_SINGLE_ARG_PARAMETER);
ACE_TRY_CHECK;

// *********************************************************

// atributos de un canal

ACE_DEBUG ((LM_DEBUG, "Creando un canal de eventos\n"));

TAO_EC_Event_Channel_Attributes attributes (poa.in (),
                                            poa.in ());

// aquí se ligan el scheduler y los atributos de un canal de eventos
attributes.scheduler = scheduler.in (); // no need to dup

attributes.consumer_reconnect = 0;
attributes.supplier_reconnect = 0;

//ACE_DEBUG ((LM_DEBUG, "Disconnect callbacks %d\n",attributes.disconnect_callbacks));

// se crea el canal event_channel
TAO_EC_Event_Channel ec_impl (attributes);
RtecEventChannelAdmin::EventChannel_var event_channel =
  ec_impl._this (ACE_ENV_SINGLE_ARG_PARAMETER);
ACE_TRY_CHECK;

// *********************************************************************************
// *********************************************************************************
// *********************************************************************************
// *********************************************************************************
// *********************************************************************************
// Aquí está la acción

////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////
ACE_DEBUG ((LM_DEBUG, "Inicia nuestro trabajo\n"));

//individuos,bits,funcionFitness,decodificador
ACE_DEBUG ((LM_DEBUG, "Configurando AG1\n"));
//AlgoritmoGenetico AG1(200, 6*7+12*3+12*3, funcionAG1, ObtenerValor);
AlgoritmoGenetico AG1(300, 6*7+6*3+6*3, funcionAG1, ObtenerValor);
AG1.setMaxGeneraciones(100);
AG1.setMaxFitness(6000);
AG1.setTamanioElite(5);
AG1.setFactorMutacion(0.2);
//AG1.setCriterioFinalizacion(0);  //0 - Se dejará de buscar al llegar a max_generaciones
AG1.setCriterioFinalizacion(1);  //1 - Se dejará de buscar al llegar a max_fitness

ACE_DEBUG ((LM_DEBUG, "Configurando AG2\n"));
AlgoritmoGenetico AG2(10, 6*7+6*3+6*3, funcionAG1, ObtenerValor);
AG2.setMaxGeneraciones(2);
AG2.setTamanioElite(4);
AG2.setFactorMutacion(0.04);
AG2.setCriterioFinalizacion(0);

ACE_DEBUG ((LM_DEBUG, "Creamos el reconfigurador\n"));
Reconfigurador pendulo(orb.in (), poa.in ());

//configuracion
ACE_DEBUG ((LM_DEBUG, "Conectamos el planificador y el canal de eventos\n"));
pendulo.set_planificador(scheduler.in ());
pendulo.set_canal_de_eventos(event_channel.in ());
pendulo.set_AG1(&AG1); // Aqui evoluciona el algoritmo
pendulo.set_AG2(&AG2); // Aqui no.
```

```
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
//nuestros amados generadores (tipo POA_RtecEventComm::PushSupplier)
planta   supplier_impl_planta;
sensor   supplier_impl_sensor1;
sensor   supplier_impl_sensor2;
sensor   supplier_impl_sensor3;
sensor   supplier_impl_sensor4;
actuador supplier_impl_actuador1;
actuador supplier_impl_actuador2;
actuador supplier_impl_actuador3;
control  supplier_impl_control;

////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
//creamos los apuntadores a generadores
RtecEventComm::PushSupplier_var supplier_planta =
supplier_impl_planta._this (ACE_ENV_SINGLE_ARG_PARAMETER);
ACE_TRY_CHECK;

RtecEventComm::PushSupplier_var supplier_sensor1 =
supplier_impl_sensor1._this (ACE_ENV_SINGLE_ARG_PARAMETER);
ACE_TRY_CHECK;

RtecEventComm::PushSupplier_var supplier_sensor2 =
supplier_impl_sensor2._this (ACE_ENV_SINGLE_ARG_PARAMETER);
ACE_TRY_CHECK;

RtecEventComm::PushSupplier_var supplier_sensor3 =
supplier_impl_sensor3._this (ACE_ENV_SINGLE_ARG_PARAMETER);
ACE_TRY_CHECK;

RtecEventComm::PushSupplier_var supplier_sensor4 =
supplier_impl_sensor4._this (ACE_ENV_SINGLE_ARG_PARAMETER);
ACE_TRY_CHECK;

RtecEventComm::PushSupplier_var supplier_actuador1 =
supplier_impl_actuador1._this (ACE_ENV_SINGLE_ARG_PARAMETER);
ACE_TRY_CHECK;

RtecEventComm::PushSupplier_var supplier_actuador2 =
supplier_impl_actuador2._this (ACE_ENV_SINGLE_ARG_PARAMETER);
ACE_TRY_CHECK;

RtecEventComm::PushSupplier_var supplier_actuador3 =
supplier_impl_actuador3._this (ACE_ENV_SINGLE_ARG_PARAMETER);
ACE_TRY_CHECK;

RtecEventComm::PushSupplier_var supplier_control =
supplier_impl_control._this (ACE_ENV_SINGLE_ARG_PARAMETER);
ACE_TRY_CHECK;

////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
ACE_DEBUG ((LM_DEBUG, "Conectamos los generadores\n"));
RtecScheduler::handle_t manej_planta =
pendulo.agregar_generador(supplier_planta.in(), "supplier_planta", 0, 1);
//la planta genera eventos tipo 1 cada 0 ms
//agregar_generador (POA_RtecEventComm::PushSupplier generador, Integer tipo_evento. Integer Id)
supplier_impl_planta.set_id(pendulo.get_supplier_id(manej_planta));
supplier_impl_planta.set_proxy(pendulo.get_proxy_push_consumer(manej_planta));

RtecScheduler::handle_t manej_sensor1 =
pendulo.agregar_generador(supplier_sensor1.in(), "supplier_sensor1", 10, 2);
supplier_impl_sensor1.set_id(pendulo.get_supplier_id(manej_sensor1));
supplier_impl_sensor1.set_proxy(pendulo.get_proxy_push_consumer(manej_sensor1));

RtecScheduler::handle_t manej_sensor2 =
pendulo.agregar_generador(supplier_sensor2.in(), "supplier_sensor2", 11, 2);
supplier_impl_sensor2.set_id(pendulo.get_supplier_id(manej_sensor2)+1);
supplier_impl_sensor2.set_proxy(pendulo.get_proxy_push_consumer(manej_sensor2));

RtecScheduler::handle_t manej_sensor3 =
pendulo.agregar_generador(supplier_sensor3.in(), "supplier_sensor3", 12, 2);
supplier_impl_sensor3.set_id(pendulo.get_supplier_id(manej_sensor3)+2);
supplier_impl_sensor3.set_proxy(pendulo.get_proxy_push_consumer(manej_sensor3));

RtecScheduler::handle_t manej_sensor4 =
pendulo.agregar_generador(supplier_sensor4.in(), "supplier_sensor4", 13, 2);
supplier_impl_sensor4.set_id(pendulo.get_supplier_id(manej_sensor4)+3);
supplier_impl_sensor4.set_proxy(pendulo.get_proxy_push_consumer(manej_sensor4));

RtecScheduler::handle_t manej_actuador1 =
pendulo.agregar_generador(supplier_actuador1.in(), "supplier_actuador1", 3, 6);
supplier_impl_actuador1.set_id(pendulo.get_supplier_id(manej_actuador1));
supplier_impl_actuador1.set_proxy(pendulo.get_proxy_push_consumer(manej_actuador1));

RtecScheduler::handle_t manej_actuador2 =
pendulo.agregar_generador(supplier_actuador2.in(), "supplier_actuador2", 3, 6);
supplier_impl_actuador2.set_id(pendulo.get_supplier_id(manej_actuador2)+1);
supplier_impl_actuador2.set_proxy(pendulo.get_proxy_push_consumer(manej_actuador2));

RtecScheduler::handle_t manej_actuador3 =
pendulo.agregar_generador(supplier_actuador3.in(), "supplier_actuador3", 3, 6);
supplier_impl_actuador3.set_id(pendulo.get_supplier_id(manej_actuador3)+2);
supplier_impl_actuador3.set_proxy(pendulo.get_proxy_push_consumer(manej_actuador3));
```

```
RtecScheduler::handle_t manej_control =
pendulo.agregar_generador(supplier_control.in(), "supplier_control", 2, 9);
supplier_impl_control.set_id(pendulo.get_supplier_id(manej_control));
supplier_impl_control.set_proxy(pendulo.get_proxy_push_consumer(manej_control));

pendulo.conectar_generadores();

////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
//nuestros queridos consumidores (tipo POA_RtecEventComm::PushConsumer)

Timeout_planta    consumidor_planta    (&supplier_impl_planta);
Timeout_sensor    consumidor_sensor1   (&supplier_impl_sensor1);
Timeout_sensor    consumidor_sensor2   (&supplier_impl_sensor2);
Timeout_sensor    consumidor_sensor3   (&supplier_impl_sensor3);
Timeout_sensor    consumidor_sensor4   (&supplier_impl_sensor4);
Timeout_actuador consumidor_actuador1 (&supplier_impl_actuador1);
Timeout_actuador consumidor_actuador2 (&supplier_impl_actuador2);
Timeout_actuador consumidor_actuador3 (&supplier_impl_actuador3);
Timeout_control  consumidor_control   (&supplier_impl_control);

////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
//creamos los apuntadores a consumidores
RtecEventComm::PushConsumer_var consumer_planta =
  consumidor_planta._this (ACE_ENV_SINGLE_ARG_PARAMETER);
ACE_TRY_CHECK;

RtecEventComm::PushConsumer_var consumer_sensor1 =
  consumidor_sensor1._this (ACE_ENV_SINGLE_ARG_PARAMETER);
ACE_TRY_CHECK;

RtecEventComm::PushConsumer_var consumer_sensor2 =
  consumidor_sensor2._this (ACE_ENV_SINGLE_ARG_PARAMETER);
ACE_TRY_CHECK;

RtecEventComm::PushConsumer_var consumer_sensor3 =
  consumidor_sensor3._this (ACE_ENV_SINGLE_ARG_PARAMETER);
ACE_TRY_CHECK;

RtecEventComm::PushConsumer_var consumer_sensor4 =
  consumidor_sensor4._this (ACE_ENV_SINGLE_ARG_PARAMETER);
ACE_TRY_CHECK;

RtecEventComm::PushConsumer_var consumer_actuador1 =
  consumidor_actuador1._this (ACE_ENV_SINGLE_ARG_PARAMETER);
ACE_TRY_CHECK;

RtecEventComm::PushConsumer_var consumer_actuador2 =
  consumidor_actuador2._this (ACE_ENV_SINGLE_ARG_PARAMETER);
ACE_TRY_CHECK;

RtecEventComm::PushConsumer_var consumer_actuador3 =
  consumidor_actuador3._this (ACE_ENV_SINGLE_ARG_PARAMETER);
ACE_TRY_CHECK;

RtecEventComm::PushConsumer_var consumer_control =
  consumidor_control._this (ACE_ENV_SINGLE_ARG_PARAMETER);
ACE_TRY_CHECK;

////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
ACE_DEBUG ((LM_DEBUG, "Conectamos los consumidores\n"));
RtecScheduler::handle_t manej_timer_planta =
pendulo.agregar_consumidor(consumer_planta.in(), "consumer_planta", 3);
//la pianta espera eventos tipo 2 cada 2 ms con NC=0 y SE=0
//agregar_consumidor (POA_RtecEventComm::PushConsumer consumidor, char *nombre,
//Integer tipo_evento)
ACE_UNUSED_ARG(manej_timer_planta);

RtecScheduler::handle_t manej_consumidor_sensor1 =
pendulo.agregar_consumidor(consumer_sensor1.in(), "consumer_sensor1", 0);
ACE_UNUSED_ARG(manej_consumidor_sensor1);
//ACE_DEBUG ((LM_DEBUG, "1\n"));

RtecScheduler::handle_t manej_consumidor_sensor2 =
pendulo.agregar_consumidor(consumer_sensor2.in(), "consumer_sensor2", 0);
ACE_UNUSED_ARG(manej_consumidor_sensor2);
//ACE_DEBUG ((LM_DEBUG, "2\n"));

RtecScheduler::handle_t manej_consumidor_sensor3 =
pendulo.agregar_consumidor(consumer_sensor3.in(), "consumer_sensor3", 0);
ACE_UNUSED_ARG(manej_consumidor_sensor3);
//ACE_DEBUG ((LM_DEBUG, "3\n"));

RtecScheduler::handle_t manej_consumidor_sensor4 =
pendulo.agregar_consumidor(consumer_sensor4.in(), "consumer_sensor4", 0);
ACE_UNUSED_ARG(manej_consumidor_sensor4);
//ACE_DEBUG ((LM_DEBUG, "4\n"));

RtecScheduler::handle_t manej_consumidor_actuador1 =
pendulo.agregar_consumidor(consumer_actuador1.in(), "consumer_actuador1", 2);
ACE_UNUSED_ARG(manej_consumidor_actuador1);
//ACE_DEBUG ((LM_DEBUG, "5\n"));

RtecScheduler::handle_t manej_consumidor_actuador2 =
```

```
pendulo.agregar_consumidor(consumer_actuador2.in(), "consumer_actuador2", 2);
ACE_UNUSED_ARG(manej_consumidor_actuador2);
//ACE_DEBUG ((LM_DEBUG, "6\n"));

RtecScheduler::handle_t manej_consumidor_actuador3 =
pendulo.agregar_consumidor(consumer_actuador3.in(), "consumer_actuador3", 2);
ACE_UNUSED_ARG(manej_consumidor_actuador3);
//ACE_DEBUG ((LM_DEBUG, "7\n"));

RtecScheduler::handle_t manej_consumidor_control1 =
pendulo.agregar_consumidor(consumer_control.in(), "consumer_control1", 10);
ACE_UNUSED_ARG(manej_consumidor_control1);
//ACE_DEBUG ((LM_DEBUG, "8\n"));

RtecScheduler::handle_t manej_consumidor_control2 =
pendulo.agregar_consumidor(consumer_control.in(), "consumer_control2", 11);
ACE_UNUSED_ARG(manej_consumidor_control2);
//ACE_DEBUG ((LM_DEBUG, "9\n"));

RtecScheduler::handle_t manej_consumidor_control3 =
pendulo.agregar_consumidor(consumer_control.in(), "consumer_control3", 12);
ACE_UNUSED_ARG(manej_consumidor_control3);
//ACE_DEBUG ((LM_DEBUG, "10\n"));

RtecScheduler::handle_t manej_consumidor_control4 =
pendulo.agregar_consumidor(consumer_control.in(), "consumer_control4", 13);
ACE_UNUSED_ARG(manej_consumidor_control4);
//ACE_DEBUG ((LM_DEBUG, "11\n"));

pendulo.conectar_consumidores();

////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
//Declaramos los timers

ACE_DEBUG ((LM_DEBUG, "Declaracion de timers\n"));
RtecScheduler::handle_t manej_timer_1 =
pendulo.agregar_reloj(consumer_planta.in(), "timer_planta");
//ACE_DEBUG ((LM_DEBUG, "1\n"));

RtecScheduler::handle_t manej_timer_2 =
pendulo.agregar_reloj(consumer_sensor1.in(), "timer_sensor1");
//ACE_DEBUG ((LM_DEBUG, "2\n"));

RtecScheduler::handle_t manej_timer_3 =
pendulo.agregar_reloj(consumer_sensor2.in(), "timer_sensor2");
//ACE_DEBUG ((LM_DEBUG, "3\n"));

RtecScheduler::handle_t manej_timer_4 =
pendulo.agregar_reloj(consumer_sensor3.in(), "timer_sensor3");
//ACE_DEBUG ((LM_DEBUG, "4\n"));

RtecScheduler::handle_t manej_timer_5 =
pendulo.agregar_reloj(consumer_sensor4.in(), "timer_sensor4");
//ACE_DEBUG ((LM_DEBUG, "5\n"));

RtecScheduler::handle_t manej_timer_6 =
pendulo.agregar_reloj(consumer_control.in(), "timer_control");
//ACE_DEBUG ((LM_DEBUG, "6\n"));

pendulo.conectar_relojes();

////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
//Declaramos dependencias

ACE_DEBUG ((LM_DEBUG, "Declaracion de dependencias\n"));
pendulo.agregar_dependencia(manej_timer_1, manej_planta);
pendulo.agregar_dependencia(manej_timer_2, manej_sensor1);
pendulo.agregar_dependencia(manej_timer_3, manej_sensor2);
pendulo.agregar_dependencia(manej_timer_4, manej_sensor3);
pendulo.agregar_dependencia(manej_timer_5, manej_sensor4);
pendulo.agregar_dependencia(manej_consumidor_actuador1, manej_actuador1);
pendulo.agregar_dependencia(manej_consumidor_actuador2, manej_actuador2);
pendulo.agregar_dependencia(manej_consumidor_actuador3, manej_actuador3);
pendulo.agregar_dependencia(manej_timer_6, manej_control);


//nos interesa ver que pasa
pendulo.activar_debug();

//Guardamos el plan
//pendulo.guardar_plan("pendulo.out");

ACE_DEBUG ((LM_DEBUG, "Inicia el reconfigurador\n"));
pendulo.iniciar();

ACE_hthread_t thr_handle;
ACE_Thread::self (thr_handle);
int prio = ACE_Sched_Params::priority_max (ACE_SCHED_FIFO);
ACE_OS::thr_setprio (thr_handle, prio);

ACE_DEBUG ((LM_DEBUG, "Activamos el canal de eventos\n"));
ec_impl.activate (ACE_ENV_SINGLE_ARG_PARAMETER);

ACE_DEBUG ((LM_DEBUG, "Listo y esperando...\n"));
```

```
      pendulo.relojes();

      orb->run (ACE_ENV_SINGLE_ARG_PARAMETER);

      // ************************************************************
      // ************************************************************
      // ************************************************************
      // ************************************************************
      // ************************************************************

      // We should do a lot of cleanup (disconnect from the EC,
      // deactivate all the objects with the POA, etc.) but this is
      // just a simple demo so we are going to be lazy.

    }
  ACE_CATCHANY
    {
      ACE_PRINT_EXCEPTION (ACE_ANY_EXCEPTION, "Servicio");
      return 1;
    }
  ACE_ENDTRY;
  return 0;
}

// ************************************************************

int parse_args (int argc, char *argv[])
{
  ACE_Get_Opt get_opts (argc, argv, "cs:");
  int c;

  while ((c = get_opts ()) != -1)
    switch (c)
      {
      case 's':
        sched_type = ACE_TEXT_ALWAYS_CHAR(get_opts.opt_arg ());
        break;

      case '?':
      default:
        ACE_ERROR_RETURN ((LM_ERROR,
                           "uso:  %s %s"
                           "\n",
                           argv [0],
                           "-s <rms|muf>"),
                          -1);
      }
  // Indicates sucessful parsing of the command line
  return 0;
}
```

Reconfigurador.cpp

```
//Reconfigurador.cpp

#include "Reconfigurador.h"

// Reloj thread.
static void *
reloj (void *arg)
{

  // Shared data via a reference.
  //int& data = *(int *) arg;
  struct arg_hilo_reloj & data = *(struct arg_hilo_reloj *) arg;

  ACE_Time_Value retardo;
  ACE_Time_Value ahora;
  // Calculate (work).
  //ACE_DEBUG ((LM_DEBUG, "(%t) writer: working for %d secs\n", work_time));

  RtecEventComm::EventSet event (1);

  event.length (1);
  event[0].header.type    = data.tipo_evento;
  event[0].header.ttl     = 1;

  //ACE_Time_Value ahora;

  while(1){
    //ahora = ACE_OS::gettimeofday ();
    //ACE_DEBUG ((LM_DEBUG,"%d\n",ahora.sec()*1000+ahora.usec()/1000));
    data.implementacion->push(event);
    retardo.msec (data.periodo-10);
    ACE_OS::sleep (retardo);
    //data.implementacion->push(event);

  }

  // Wake up reader.
  //ACE_DEBUG ((LM_DEBUG, "(%t) writer: calculation complete, waking reader\n"));
```

```
  //if (EVENT::instance ()->signal () == -1)
  //  {
  //    ACE_ERROR ((LM_ERROR, "thread signal failed"));
  //    ACE_OS::exit (0);
  //  }

  return 0;
}

struct arg_hilo_reloj hilos[10];

Reconfigurador::Reconfigurador (CORBA::ORB_ptr orb, PortableServer::POA_ptr poa)
: orb_ (CORBA::ORB::_duplicate (orb)),
  poa_ (poa)
{

  debug_ = 0;
  min_os_priority_ = ACE_Sched_Params::priority_min (ACE_SCHED_FIFO, ACE_SCOPE_THREAD);
  max_os_priority_ = ACE_Sched_Params::priority_max (ACE_SCHED_FIFO, ACE_SCOPE_THREAD);
  contador_generadores_ = 0;
  contador_consumidores_ = 0;
  contador_relojes_ = 0;
  contador_dependencias_ = 0;
  AG_en_linea_ = new Timeout_AG2;
  AG_en_linea_->set_reconfigurador (this);
  consumer_AG2_ = AG_en_linea_->_this ();
  supplier_AG2_ = this->_this();

  tm_mio_ = ACE_Thread_Manager::instance ();

}

void Reconfigurador::relojes ()
{
  int i;

  for(i=0;i<contador_relojes_;i++){
    hilos[i].timer = i;
    hilos[i].tipo_evento = ACE_ES_EVENT_UNDEFINED + i + 100;
    hilos[i].periodo = relojes_[i].periodo;
    hilos[i].proxy_reloj = supplier_relojes_proxy_[i];
    hilos[i].implementacion = relojes_[i].implementacion;
    // Create reader thread.
    tm_mio_->spawn ((ACE_THR_FUNC) reloj, (void *) &hilos[i]);

  }
}

void Reconfigurador::set_planificador (RtecScheduler::Scheduler_ptr planificador)
{
  scheduler_ = planificador;
}

void Reconfigurador::set_canal_de_eventos (RtecEventChannelAdmin::EventChannel_ptr canal)
{
  event_channel_ = canal;
  supplier_admin_ = event_channel_->for_suppliers();
  consumer_admin_ = event_channel_->for_consumers();
}

RtecEventComm::EventSourceID
Reconfigurador::get_supplier_id(RtecScheduler::handle_t manejador)
{

  int i;

  for(i=0;i<contador_generadores_;i++){
    if(generadores_[i].manejador == manejador){
      return generadores_[i].id;
    }
  }

  return 0;

}

RtecEventChannelAdmin::ProxyPushConsumer_ptr
Reconfigurador::get_proxy_push_consumer(RtecScheduler::handle_t manejador)
{

 int i;

  for(i=0;i<contador_generadores_;i++){
    if(generadores_[i].manejador == manejador){
      return consumer_proxy_[i].in();
    }
  }

  return 0;

}

void Reconfigurador::conectar_generadores ()
{

  int i;
```

```
  generador_AG2_ = agregar_generador(supplier_AG2_.in(), "supplier_AG2", ACE_ES_EVENT_UNDEFINED + 100, 99);
  set_id(get_supplier_id(generador_AG2_));
  set_proxy(get_proxy_push_consumer(generador_AG2_));

  //ACE_DEBUG ((LM_DEBUG, "Conectando suppliers\n"));
  for(i=0;i<contador_generadores_;i++){
    consumer_proxy_[i]->connect_push_supplier(generadores_[i].implementacion,
                                              generadores_[i].qos.get_SupplierQOS());
  }

}

void Reconfigurador::conectar_consumidores ()
{

  int i;

  //el AG2
  consumidor_AG2_ = agregar_consumidor(consumer_AG2_.in(), "consumer_AG2", ACE_ES_EVENT_UNDEFINED + 100);

  //ACE_DEBUG ((LM_DEBUG, "Conectando consumers\n"));
  for(i=0;i<contador_consumidores_;i++){
    supplier_proxy_[i]->connect_push_consumer(consumidores_[i].implementacion,
                                              consumidores_[i].qos.get_ConsumerQOS());
  }

}

void Reconfigurador::conectar_relojes ()
{

  int i;

  // el AG2
  timer_AG2_ = agregar_reloj(consumer_AG2_.in(), "timer_AG2");

  //ACE_DEBUG ((LM_DEBUG, "Conectando relojes\n"));
  for(i=0;i<contador_relojes_;i++){
    supplier_relojes_proxy_[i]->connect_push_consumer(relojes_[i].implementacion,
                                              relojes_[i].qos.get_ConsumerQOS());
  }

}

void Reconfigurador::iniciar ()
{

  int i;

  agregar_dependencia(timer_AG2_, generador_AG2_);
  //agregar_dependencia(generador_AG2_, relojes_[0].manejador);

  ACE_DEBUG ((LM_DEBUG, "Agregando dependencias\n"));
  for(i=0;i<contador_dependencias_;i++){
    scheduler_->add_dependency(dependencias_[i].tarea, dependencias_[i].dependencia,
                               1,RtecBase::TWO_WAY_CALL);
  }

  ACE_DEBUG ((LM_DEBUG, "Calculando plan\n"));
  scheduler_->compute_scheduling (min_os_priority_, max_os_priority_, infos_.out (),
                                  dependencies_.out (), configs_.out (),
                                  anomalies_.out () ACE_ENV_ARG_PARAMETER);

  //Guardando el plan
  guardar_plan("pendulo.out");

  //Copiamos el mejor individuo
  AG2_->setIndividuo(0, AG1_->getIndividuo(0));

}

void Reconfigurador::detener ()
{
  event_channel_->destroy();
  orb_->shutdown (0);
  exit(0);
}

RtecScheduler::handle_t
Reconfigurador::agregar_generador (RtecEventComm::PushSupplier_ptr generador, char * nombre,
                                   int tipo_evento, int id)
{

  ACE_Time_Value tv(0,0);
  TimeBase::TimeT tmp;
  tv.set(0,1000*relojes_[generadores_[contador_generadores_].timer].tiempo_de_ejecucion);
  ORBSVCS_Time::Time_Value_to_TimeT (tmp,tv);
  tv.set(0,1000*relojes_[generadores_[contador_generadores_].timer].periodo);


  RtecScheduler::Importance_t importancia_tmp;
  importancia_tmp =
   (RtecScheduler::Importance_t)relojes_[generadores_[contador_generadores_].timer].subprioridad_estatica;

  RtecScheduler::Criticality_t nivel_critico_tmp;
  nivel_critico_tmp =
```

```
  (RtecScheduler::Criticality_t)relojes_[generadores_[contador_generadores_].timer].nivel_critico;

  generadores_[contador_generadores_].implementacion = generador;
  generadores_[contador_generadores_].tipo_evento = ACE_ES_EVENT_UNDEFINED + tipo_evento;
  generadores_[contador_generadores_].manejador = scheduler_->create(nombre);
  generadores_[contador_generadores_].id = id;
  generadores_[contador_generadores_].qos.insert(generadores_[contador_generadores_].id,
                                                 generadores_[contador_generadores_].tipo_evento,
                                                 generadores_[contador_generadores_].manejador, 1);

  if(generadores_[contador_generadores_].tipo_evento >= ACE_ES_EVENT_UNDEFINED + 100){
    generadores_[contador_generadores_].qos.insert(generadores_[contador_generadores_].id,
                                                 generadores_[contador_generadores_].tipo_evento + 1,
                                                 generadores_[contador_generadores_].manejador, 1);

    generadores_[contador_generadores_].qos.insert(generadores_[contador_generadores_].id,
                                                 generadores_[contador_generadores_].tipo_evento + 2,
                                                 generadores_[contador_generadores_].manejador, 1);

    generadores_[contador_generadores_].qos.insert(generadores_[contador_generadores_].id,
                                                 generadores_[contador_generadores_].tipo_evento + 3,
                                                 generadores_[contador_generadores_].manejador, 1);

    generadores_[contador_generadores_].qos.insert(generadores_[contador_generadores_].id,
                                                 generadores_[contador_generadores_].tipo_evento + 4,
                                                 generadores_[contador_generadores_].manejador, 1);

    generadores_[contador_generadores_].qos.insert(generadores_[contador_generadores_].id,
                                                 generadores_[contador_generadores_].tipo_evento + 5,
                                                 generadores_[contador_generadores_].manejador, 1);

    generadores_[contador_generadores_].qos.insert(generadores_[contador_generadores_].id,
                                                 generadores_[contador_generadores_].tipo_evento + 6,
                                                 generadores_[contador_generadores_].manejador, 1);

  }

  scheduler_->set(generadores_[contador_generadores_].manejador,
                  nivel_critico_tmp,
                  tmp,tmp,tmp,
                  time_val_to_period(tv),
                  importancia_tmp,
                  tmp, 0, RtecScheduler::OPERATION);


  consumer_proxy_[contador_generadores_] = supplier_admin_->obtain_push_consumer();

  contador_generadores_++;

  return generadores_[contador_generadores_ - 1].manejador;
}

RtecScheduler::handle_t
Reconfigurador::agregar_consumidor (RtecEventComm::PushConsumer_ptr consumidor,
                                    char * nombre, int tipo_evento)
{

  consumidores_[contador_consumidores_].implementacion = consumidor;
  consumidores_[contador_consumidores_].tipo_evento = ACE_ES_EVENT_UNDEFINED + tipo_evento;
  consumidores_[contador_consumidores_].manejador = scheduler_->create(nombre);

  ACE_Time_Value tv(0,0);
  TimeBase::TimeT tmp;
  //tv.set(0,1000*consumidores_[contador_consumidores_].tiempo_de_ejecucion);
  ORBSVCS_Time::Time_Value_to_TimeT (tmp,tv);
  //tv.set(0,0);

  RtecScheduler::Importance_t importancia_tmp;
  importancia_tmp =
   (RtecScheduler::Importance_t)relojes_[consumidores_[contador_consumidores_].timer].subprioridad_estatica;

  RtecScheduler::Criticality_t nivel_critico_tmp;
  nivel_critico_tmp =
  (RtecScheduler::Criticality_t)relojes_[consumidores_[contador_consumidores_].timer].nivel_critico;

  scheduler_->set(consumidores_[contador_consumidores_].manejador,
                  nivel_critico_tmp,
                  tmp,tmp,tmp,
                  time_val_to_period(tv),
                  importancia_tmp,
                  tmp, 0, RtecScheduler::OPERATION);

              consumidores_[contador_consumidores_].qos.insert_type(consumidores_[contador_consumidores_].tipo_evento,
consumidores_[contador_consumidores_].manejador);

  supplier_proxy_[contador_consumidores_] = consumer_admin_->obtain_push_supplier();

  contador_consumidores_++;

  return consumidores_[contador_consumidores_ - 1].manejador;
}

RtecScheduler::handle_t
Reconfigurador::agregar_reloj (RtecEventComm::PushConsumer_ptr consumidor, char * nombre)
{

  ACE_Time_Value tv(0,0);
```

```
  TimeBase::TimeT tmp;
  tv.set(0,1000*relojes_[contador_relojes_].tiempo_de_ejecucion);
  ORBSVCS_Time::Time_Value_to_TimeT (tmp,tv);
  tv.set(0,1000*relojes_[contador_relojes_].periodo);

  RtecScheduler::Importance_t importancia_tmp;
  importancia_tmp =
   (RtecScheduler::Importance_t)relojes_[contador_relojes_].subprioridad_estatica;

  RtecScheduler::Criticality_t nivel_critico_tmp;
  nivel_critico_tmp =
  (RtecScheduler::Criticality_t)relojes_[contador_relojes_].nivel_critico;

  relojes_[contador_relojes_].implementacion = consumidor;
  relojes_[contador_relojes_].manejador = scheduler_->create(nombre);
  relojes_[contador_relojes_].tipo_evento = ACE_ES_EVENT_UNDEFINED + contador_relojes_ + 100;

  scheduler_->set(relojes_[contador_relojes_].manejador,
                  nivel_critico_tmp,
                  tmp,tmp,tmp,
                  time_val_to_period(tv),
                  importancia_tmp,
                  tmp, 0, RtecScheduler::OPERATION);

relojes_[contador_relojes_].qos.insert_type(relojes_[contador_relojes_].tipo_evento,
relojes_[contador_relojes_].manejador);

  supplier_relojes_proxy_[contador_relojes_] = consumer_admin_->obtain_push_supplier();

  contador_relojes_++;

  return relojes_[contador_relojes_ - 1].manejador;
}

void
Reconfigurador::agregar_dependencia (RtecScheduler::handle_t tarea,
                                     RtecScheduler::handle_t dependencia)
{
  dependenclas_[contador_dependencias_].tarea = tarea;
  dependencias_[contador_dependencias_].dependencia = dependencia;

  contador_dependenclas_++;
}

void Reconfigurador::guardar_plan (char *archivo)
{

  scheduler_->get_rt_info_set (infos_.out());
  scheduler_->get_dependency_set (dependencles_.out());
  scheduler_->get_config_info_set (configs_.out());

  ACE_Scheduler_Factory::dump_schedule (infos_.in (), dependencies_.in (), configs_.in (),
                                        anomalies_.in (), archivo);
}

void Reconfigurador::alterar_periodo_tarea (RtecScheduler::handle_t tarea, int periodo)
{
  //espero no usarla
  ACE_UNUSED_ARG(tarea);
  ACE_UNUSED_ARG(periodo);
}

void Reconfigurador::eliminar_generador (RtecScheduler::handle_t generador)
{
  //espero no usarla
  ACE_UNUSED_ARG(generador);
}

void Reconfigurador::eliminar_consumidor (RtecScheduler::handle_t consumidor)
{
  //espero no usarla
  ACE_UNUSED_ARG(consumidor);

}

void Reconfigurador::set_AG1 (AlgoritmoGenetico* ag)
{
  int i;
  AG1_ = ag;

  //Buscamos el mejor individuo
  ACE_DEBUG ((LM_DEBUG,"Comienza AG1\n"));
  AG1_->evolucionar();

  fitness_actual_ = AG1_->getFitnessIndividuo(0);
  ACE_DEBUG ((LM_DEBUG,"El mejor individuo obtuvo un %f.\n", AG1_->getFitnessIndividuo(0)));

  ACE_DEBUG ((LM_DEBUG,"%s\n",AG1_->getIndividuo(0)));

  relojes_[0].periodo = (int)AG1_->getVariable(0,1);   //T0
  relojes_[1].periodo = (int)AG1_->getVariable(0,2);   //T1
  relojes_[2].periodo = (int)AG1_->getVariable(0,3);   //T2
  relojes_[3].periodo = (int)AG1_->getVariable(0,4);   //T3
  relojes_[4].periodo = (int)AG1_->getVariable(0,5);   //T4
  relojes_[5].periodo = (int)AG1_->getVariable(0,6);   //T5

 for(i=0;i<6;i++){
```

ESTA TESIS NO SALE
DE LA BIBLIOTECA

```
  if(relojes_[i].periodo%10!=0) relojes_[i].periodo = (relojes_[i].periodo/10)*10+10;
}

 relojes_[0].tiempo_de_ejecucion = 2;
 relojes_[1].tiempo_de_ejecucion = 1;
 relojes_[2].tiempo_de_ejecucion = 1;
 relojes_[3].tiempo_de_ejecucion = 1;
 relojes_[4].tiempo_de_ejecucion = 1;
 relojes_[5].tiempo_de_ejecucion = 3;

 //Para la reconfiguracion
 relojes_[6].tiempo_de_ejecucion = 20;
 relojes_[6].nivel_critico = 4;  //VL_C
 relojes_[6].subprioridad_estatica = 4; //VL_1
 relojes_[6].periodo = 1000;

 //Dependencias
 generadores_[0].timer = 0;
 generadores_[1].timer = 1;
 generadores_[2].timer = 2;
 generadores_[3].timer = 3;
 generadores_[4].timer = 4;
 generadores_[5].timer = 5;
 generadores_[6].timer = 5;
 generadores_[7].timer = 5;
 generadores_[8].timer = 5;
 generadores_[9].timer = 6;

 consumidores_[0].timer = 5;
 consumidores_[1].timer = 0;
 consumidores_[2].timer = 0;
 consumidores_[3].timer = 0;
 consumidores_[4].timer = 0;
 consumidores_[5].timer = 5;
 consumidores_[6].timer = 5;
 consumidores_[7].timer = 5;
 consumidores_[8].timer = 1;
 consumidores_[9].timer = 2;
 consumidores_[10].timer = 3;
 consumidores_[11].timer = 4;
 consumidores_[12].timer = 6;

 for(i=0;i<6;i++){
   relojes_[i].nivel_critico = (int)AG1_->getVariable(0,i+6);
   if(relojes_[i].nivel_critico > 4) relojes_[i].nivel_critico = 4;
 }

 for(i=0;i<6;i++){
   relojes_[i].subprioridad_estatica = (int)AG1_->getVariable(0,i+12);
   if(relojes_[i].subprioridad_estatica > 4) relojes_[i].subprioridad_estatica = 4;
 }

 for(i=0;i<7;i++){
   ACE_DEBUG((LM_DEBUG,"T[%d]=%d C[%d]=%d NC[%d]=%d SE[%d]=%d\n",i,relojes_[i].periodo,
                                                          i,relojes_[i].tiempo_de_ejecucion,
                                                          i,relojes_[i].nivel_critico,
                                                          i,relojes_[i].subprioridad_estatica));
 }

 float Uu=0.0;
 for(i=0;i<7;i++){
   Uu = Uu + ((float)relojes_[i].tiempo_de_ejecucion / (float)relojes_[i].periodo);
 }

 ACE_DEBUG((LM_DEBUG,"El porcentaje de utilizacion es de %f\n",Uu));

}

void Reconfigurador::set_AG2 (AlgoritmoGenetico* ag)
{
  AG2_ = ag;
}

void Reconfigurador::activar_debug ()
{
  debug_ = 1;
}

void Reconfigurador::desactivar_debug ()
{
  debug_ = 0;
}

void Reconfigurador::corre_AG2 ()
{

  float fitness_temp;
  int i;

  //ACE_DEBUG ((LM_DEBUG,"Comienza AG2\n"));
AG2_->setIndividuo(0,"0001010000001000001010001010000010100000100111110001011101010111110110000011100");     //7039     -
estable

  AG2_->evolucionar();

  //Verificar si el fitness encontrado es mejor que el actual. De ser así, se reconfigura.
  fitness_temp = AG2_->getFitnessIndividuo(0);
```

```
if (fitness_actual_ < fitness_temp){

  ACE_DECLARE_NEW_CORBA_ENV;
  ACE_TRY
  {

  ahora = ACE_OS::gettimeofday ();
  ACE_DEBUG ((LM_DEBUG,"%d%6d REI\n",ahora.sec(), ahora.usec()));

  //ACE_DEBUG ((LM_DEBUG,"Comienza reconfiguración\n"));
  //ACE_DEBUG ((LM_DEBUG,"Mejor individuo: %s\n",AG2_->getIndividuo(0)));

  //se obtienen los valores de AG2

  relojes_[0].periodo = (int)AG2_->getVariable(0,1);   //T0
  relojes_[1].periodo = (int)AG2_->getVariable(0,2);   //T1
  relojes_[2].periodo = (int)AG2_->getVariable(0,3);   //T2
  relojes_[3].periodo = (int)AG2_->getVariable(0,4);   //T3
  relojes_[4].periodo = (int)AG2_->getVariable(0,5);   //T4
  relojes_[5].periodo = (int)AG2_->getVariable(0,6);   //T5

  for(i=0;i<6;i++){
    if(relojes_[i].periodo%10!=0) relojes_[i].periodo = (relojes_[i].periodo/10)*10+10;
  }

  for(i=0;i<6;i++){
    relojes_[i].nivel_critico = (int)AG2_->getVariable(0,i+6);
    if(relojes_[i].nivel_critico > 4) relojes_[i].nivel_critico = 4;
  }

  for(i=0;i<6;i++){
    relojes_[i].subprioridad_estatica = (int)AG2_->getVariable(0,i+12);
    if(relojes_[i].subprioridad_estatica > 4) relojes_[i].subprioridad_estatica = 4;
  }

  for(i=0;i<7;i++){
    ACE_DEBUG((LM_DEBUG,"T[%d]=%d C[%d]. %d NC[%d]=%d SE[%d]=%d\n",i,relojes_[i].periodo,
                                                    i,relojes_[i].tiempo_de_ejecucion,
                                                    i,relojes_[i].nivel_critico,
                                                    i,relojes_[i].subprioridad_estatica));
  }

  float Uu=0.0;
  for(i=0;i<7;i++){
    Uu = Uu + ((float)relojes_[i].tiempo_de_ejecucion / (float)relojes_[i].periodo);
  }

  ACE_DEBUG((LM_DEBUG,"El nuevo porcentaje de utilizacion es de %f\n",Uu));

  //se desconectan y conectan de nuevo los timers
  ACE_DEBUG ((LM_DEBUG,"Resets\n"));

  //Alteramos el valor de las estructuras
  for(i=0;i<contador_relojes_;i++){
    hilos[i].periodo = relojes_[i].periodo;
  }

  //se reconfigura el scheduler
  ACE_Time_Value tv(0,0);
  TimeBase::TimeT tmp;

  RtecScheduler::Importance_t importancia_tmp;
  RtecScheduler::Criticality_t nivel_critico_tmp;

  for(i=0;i<6;i++){

    tv.set(0,1000*relojes_[i].tiempo_de_ejecucion);
    ORBSVCS_Time::Time_Value_to_TimeT (tmp,tv);
    tv.set(0,1000*relojes_[i].periodo);

    importancia_tmp =
    (RtecScheduler::Importance_t)relojes_[1].subprioridad_estatica;

    nivel_critico_tmp =
    (RtecScheduler::Criticality_t)relojes_[i].nivel_critico;

    scheduler_->reset(relojes_[i].manejador,
                  nivel_critico_tmp,                    //poco critico
                  tmp, tmp, tmp,                        //worstC,typicalC,cachedC
                  time_val_to_period (tv),              //P
                  importancia_tmp,                      //poco importante
                  tmp,                                  //quantum
                  0,                                    //threads
                  RtecScheduler::OPERATION              //tipo de operación
                  ACE_ENV_ARG_PARAMETER);

    ACE_TRY_CHECK;
  }

  for(i=0;i<12;i++){

    tv.set(0,0);
    ORBSVCS_Time::Time_Value_to_TimeT (tmp,tv);

    importancia_tmp =
    (RtecScheduler::Importance_t)relojes_[consumidores_[i].timer].subprioridad_estatica;
```

```
      nivel_critico_tmp =
      (RtecScheduler::Criticality_t)relojes_[consumidores_[i].timer].nivel_critico;

      scheduler_->reset(consumidores_[i].manejador,
                        nivel_critico_tmp,                    //nivel critico
                        tmp, tmp, tmp,                        //worstC,typicalC, cachedC
                        time_val_to_period (tv),              //P
                        importancia_tmp,                      //importancia
                        tmp,                                  //quantum
                        0,                                    //threads
                        RtecScheduler::OPERATION              //tipo de operación
                        ACE_ENV_ARG_PARAMETER);

      ACE_TRY_CHECK;
    }

    ACE_DEBUG ((LM_DEBUG,"Recompute\n"));
    scheduler_->recompute_scheduling (min_os_priority_,
                                      max_os_priority_,
                                      anomalies_.out ()
                                      ACE_ENV_ARG_PARAMETER);
    ACE_TRY_CHECK;

    //Guardando el plan
    guardar_plan("pendulo_reconf.out");

      ACE_DEBUG ((LM_DEBUG,"Termina evento\n"));
    }
    ACE_CATCHANY
    {
      ACE_PRINT_EXCEPTION (ACE_ANY_EXCEPTION, "Reconfigurador falla");
    }
    ACE_ENDTRY;

    //Actualizamos el fitness para la proxima reconfiguracion
    fitness_actual_ = fitness_temp;

    ahora = ACE_OS::gettimeofday ();
    ACE_DEBUG ((LM_DEBUG,"%d%6d REF\n",ahora.sec(), ahora.usec()));

  } //if
}

void Reconfigurador::disconnect_push_supplier (ACE_ENV_SINGLE_ARG_DECL_NOT_USED)
    ACE_THROW_SPEC ((CORBA::SystemException))
{

}

void
Reconfigurador::set_id(RtecEventComm::EventSourceID id){
  id_ = id;
}

void
Reconfigurador::set_proxy(RtecEventChannelAdmin::ProxyPushConsumer_ptr consumer_proxy){
  reconfigurador_proxy_ = consumer_proxy;
}


// definición de una función inline que regresa un RtecScheduler:Period_t.
// Sirve para convertir un ACE_Time_Value
// en un Period_t. Este valor esta dado en 100 nanosegundos por unidad
RtecScheduler::Period_t Reconfigurador::time_val_to_period (const ACE_Time_Value &tv)
{
  //100s of nanoseconds  (cientos de nanosegundos, no centesimas)
  return (tv.sec () * 1000000 + tv.usec ())*10;
}
```

8. Referencias

(Branke, 1999) Branke J.; "Evolutionary approaches to dinamyc optimization problems – A survey-"; en *GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems*, pp. 134-137, 1999.

(Burns, et al., 1990) Burns A. y Wellings A.; "Real-time systems and their programming languages"; segunda edición. Addison-wesley. Gran Bretaña. 1990.

(CDOC, 1997) Center for Distributed Object Computing; "The ACE ORB (TAO)"; http://www.cs.wustl.edu/~schmidt/TAO.html, Universidad de Washington. Último acceso: 25 de mayo de 2005.

(Coulouris, et al., 2001) Coulouris G., Dollimore J. y Kindberg T.; "Sistemas distribuidos, conceptos y diseño"; tercera edición. Pearson Educación, S. A., Madrid, 2001.

(García, 2003) García Z. A.; "Propuesta de un método de planificación para la reconfiguración en línea en un sistema de tiempo real"; *Tesis de Maestría en Ciencia e Ingeniería de la Computación, UNAM*. México, D. F., 2003.

(Gill, et al., 1998) Gill C., Levine D. y Schmidt D.; "The Design and performance of a Real-Time CORBA Scheduling Service"; en *International Journal of Time-Critical Computing Systems, special issue on Real-Time Middleware*, agosto 10, 1998.

(Harrison, et al., 1997) Harrison T., Levine D. y Schmidt D.; "The Design and Performance of a Real-time CORBA Event Service"; *Object Oriented Programming, Systems, Languages and Applications '97*, (Atlanta, GA), pp. 184-199, ACM, octubre de 1997.

(Huang, et al., 1997) Huang J., Jha R., Heimerdinger W., Muhammad M., Lauzac S., Kannikeswaran B., Schwan K., Zhao W. y Bettati R.; "RT-ARM: A Real-Time Adaptive Resource Management System for Distributed Mission-Critical Applications"; *Workshop on Middleware for Distributed Real-Time Systems, RTSS-97*, (San Francisco, California), IEEE, 1997.

(Kuri, 2003) Kuri M. A.; "Solution of Simultaneous Non-Linear Equations using Genetic Algorithms"; *World Scientific and Enginnering Academy and Society Transactions on systems*, pp. 44-51, WSEAS Press, Issue 1, Vol 2, enero 2003.

(Kuri, et al., 2001) Kuri M. A., Galaviz C. J.; "Algoritmos Genéticos"; Instituto Politécnico Nacional, Universidad Nacional Autónoma de México, Fondo de Cultura Económica, México, 2002.

(L. Karr, et al., 1999) L. Karr C., Freeman L. Michael. "Industrial applications of genetic algorithms". The CRC press. USA, 1999.

(Levine, et al., 1998) Levine D., Gill C. y Schmidt D.; "Dynamic Scheduling Strategies for Avionics

Mission Computing"; en *17th IEEE/AIAA Digital Avionics System Conference*, 31 octubre - 6 noviembre 1998.

(Lian, et al., 2002) Lian F., Moyne J., Tilbury D.; "Network design considerations for distributed control systems"; en *IEEE transactions on control systems technology*, pp. 297-307, Vol. 10 No. 2, marzo 2002.

(Liu, et al., 1973) Liu C. y Layland J.; "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Enviroment"; *Journal of the ACM*, pp. 46-61, Vol. 20, enero de 1973.

(Messner, et al., 1997) Messner B. y Tilbury D.; "Control Tutorials for Matlab", http://www.engin.umich.edu/group/ctm, Universidad de Michigan, 1997. Último acceso: 25 de mayo de 2005.

(Montana, et al., 1998) Montana D., Brinn M., Moore S., Bidwell G.; "Genetic Algorithms for complex, Real-Time Scheduling"; en *IEEE Conference on Systems, Man, and Cybernetics*, 1998.

(Ogata, 1997) Ogata K.; "Discrete Time Control Systems"; Prentice Hall, Segunda edición, 1997.

(OMG, 1999) Object Management Group; "Real-Time CORBA"; OMG TC Document ptc/99-05-03 mayo 28, 1999.

(OMG, 2001) Object Management Group; "The Common Object Request Broker: Architecture and Specification"; revisión 2.6, diciembre 2001.

(Page-Jones, 1999) Page-Jones M.; "Fundamentals of object-oriented design in UML"; Addison Wesley Profesional, primera edición, 1999.

(Sanz, et al., 2001A) Sanz R., Clavijo J., Segarra A., de Antonio A. y Alonso M.; "CORBA-Based Substation Automation Systems"; en *2001 IEEE International Conference on Control Applications*, Ciudad de México, México, septiembre 5-7, 2001.

(Sanz, et al., 2001B) Sanz R., Alonso M., López I. y García C.; "Enhancing Control Architectures using CORBA"; en *2001 IEEE International Symposium on Intelligent Control*, Ciudad de México, México, septiembre 5-7, 2001.

(Schmidt, et al., 1998) Schmidt D., Levine D. y Mungee S.; "The Design of the TAO Real-Time Object Request Broker"; *17 IEEE/AIAA Digital Avionics Systems Conference*, 31 octubre - 6 noviembre 1998.

(Schmidt, 1994) Schmidt D., "ACE: an Object-Oriented Framework for Developing Distributed Applications", *6th USENIX C++ Conference*, Cambridge, Massachusetts, abril 1994.

(Vinoski, 1997) Vinoski S.; "CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Enviroments"; *IEEE Communications Magazine*, Vol. 35, No. 2, febrero de 1997.

(Walsh, et al., 2003) Walsh G., Ye H., Bushnell L.; "Stability analysis of networked control systems"; en *IEEE transactions on control systems technology*, pp. 438-446,Vol. 10 No. 3, mayo 2003.