1- Introducción a los sistemas de tiempo real

1. Sistemas en tiempo real

Un **sistema en tiempo real** (STR) es cualquier sistema en el que la entrada al sistema corresponde a un estímulo externo, y cuya salida estará relacionada a dicha entrada, debiendo darse esta salida en un periodo de tiempo finito y determinado. La corrección no depende solo del resultado lógico de la computación, sino además del tiempo en que se producen dichos resultados.

Los STR son inherentemente concurrentes al estar generalmente integrados en un sistema más grande, ya que deben modelar el paralelismo que existe en los objetos del mundo real que están monitorizando y controlando.

Clasificación de los Sistemas en Tiempo Real

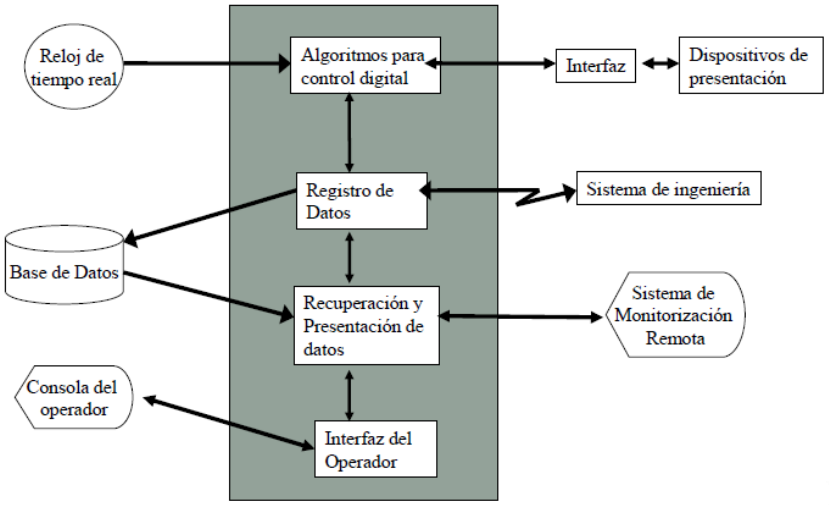
Los STR se pueden clasificar en tres conjuntos:

- **Estrictos (*hard*)**: es crucial que las respuestas ocurran dentro del tiempo límite especificado (por ejemplo, sistemas de control de vuelo).
- **No Estrictos o Flexibles (*soft*)**: los tiempos de respuesta son importantes, pero el sistema seguirá funcionando correctamente incluso si ocasionalmente no se cumplen los tiempos (por ejemplo, sistemas de adquisición de datos).
- **Firme (*firm*)**: son STR no estrictos donde el tiempo límite puede no cumplirse ocasionalmente, aunque una respuesta tardía no tiene valor (por ejemplo, sistemas multimedia).

La pérdida del tiempo límite está asociada en los STR mediante una **función de coste**. Además, en los STR pueden diferenciarse:

- **Sistemas de tiempo explícito (*Time-aware*)**: en su definición hacen referencia explícita al tiempo (por ejemplo, abrir una puerta a las 9.00).
- **Sistemas reactivos**: deben producir una respuesta cuando se produce un evento (por ejemplo, se recibe un dato del entorno). Estos eventos pueden ser:
    - Eventos de tiempo: deben producir una respuesta cuando ocurre un evento temporal (por ejemplo, lanzar un proceso a las 9.00 o cada 25 ms como actividad periódica).
    - Eventos internos o externos: deben producir una respuesta cuando ocurre un evento interno o externo.

2. Características generales de los STR



- **Grande y complejo**: son sistemas que varían desde sencillos sistemas empotrados de un solo procesador (con cientos de líneas de código) hasta sistemas distribuidos multiplataforma y multilenguaje (con millones de líneas de código). Pueden además contener una gran variedad de subsistemas. Deben ser capaces de abordar un **cambio continuo** que requiera un mantenimiento constante.
- **Manipulación de números reales:** debe ser capaz de soportar procesamiento matemáticamente complejo con un grado de precisión elevado.
- **Extremadamente fiable y seguro:** controlan el entorno en el que operan. El software debe estar diseñado con la máxima integridad, y los programas deben ser tolerantes a fallos y continuar funcionando, aunque proporcionen un servicio degradado. En el peor de los casos, un STR debe garantizar la seguridad del medio antes de apagarse de una forma controlada pues pueden producir consecuencias catastróficas de su mal funcionamiento
- **Control concurrente de los distintos componentes del sistema**: los diferentes componentes trabajan en paralelo en el mundo real. Se debe modelar este paralelismo por entidades de funcionamiento concurrente, para que todo funcione de una forma coordinada.
- **Funcionalidades de tiempo real:** se debe poder predecir con precisión el **peor caso de tiempo de respuesta**. En un STR, la eficiencia en su funcionamiento es importante pero la predictibilidad es esencial.
- **Interacción con interfaces hardware:** se debe ser capaz de poder programar una gran variedad de dispositivos (sensores/actuadores) de forma fiable.

3. Ciclos de desarrollo de un STR

La parte más importante del desarrollo de un STR es la construcción de un **diseño consistente** que satisfaga una **especificación acreditada** de los requisitos. Para el desarrollo de un STR se emplea una metodología descendente (top-down), para lo cual es necesaria una compresión de qué es realizable en los niveles inferiores.

Las fases de diseño de un STR son las siguientes:

1. **Especificación de Requisitos**

Proporciona una especificación acreditada de requisitos de la que partirá el diseño del sistema. Se suele usar un lenguaje natural para proporcionar esta especificación, aunque cada vez son más usados métodos formales. Se define la funcionalidad del sistema (comportamiento temporal, requisitos de fiabilidad y comportamiento en caso de fallos) y los test de aceptación. Se construye un modelo del entorno de aplicación.

2. **Diseño de Arquitectura o Arquitectónico**

Para el diseño estructurado de cualquier lenguaje de STR se utiliza la descomposición y la abstracción. Con la **descomposición** se divide el sistema complejo en partes más pequeñas, hasta poder aislar componentes que puedan ser comprendidos y diseñados de forma individual o en pequeños grupos. Por cada nivel de descomposición debe existir un nivel de descripción apropiado, así como un método que documente o exprese dicha descripción. Por su lado, con la **abstracción** se permite tener una visión simplificada del sistema y de los objetos contenidos en él, pero seguirá teniendo las propiedades y características esenciales.

Un diseño estructurado de alto nivel debe ser una especificación acreditada de los requisitos. Cuando la especificación completa puede verificarse teniendo en cuenta únicamente la especificación de sus componentes se habla de **especificación composicional**. En el marco del diseño podemos encontrar:

- **Encapsulamiento:** se da cuando la especificación completa del sistema software puede verificarse teniendo en cuenta únicamente la especificación de los subcomponentes inmediatos (la descomposición es *composicional*).
- **Cohesión y acoplamiento:** son dos medidas que describen la vinculación entre módulos. La cohesión expresa el grado de unión (su fuerza interna) de un módulo, y el acoplamiento expresa la interdependencia entre dos módulos de un programa, pudiendo también entenderse como la facilidad con la que puede eliminarse un módulo de un sistema ya completo y ser este remplazado por un módulo alternativo.
- **Aproximaciones formales:** el sistema concurrente se representa mediante un conjunto de máquinas de estado finito, y se emplean técnicas de exploración de estados para investigar los posibles comportamientos del sistema. Desgraciadamente, se han desarrollado pocos lenguajes de implementación con una descripción formal en mente.

3. Diseño Detallado

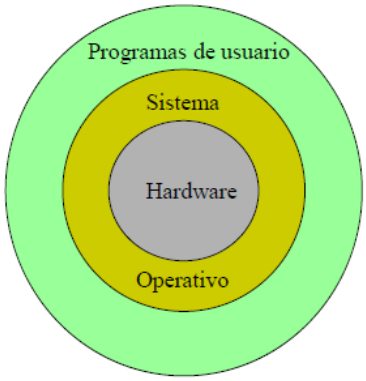
Durante este ciclo se especifica el diseño completo del sistema. Se suele hacer en dos estados: la descomposición en módulos, y el diseño interno de cada módulo.

4. Implementación

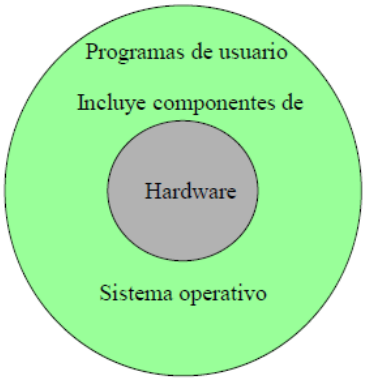
Solo es posible en esta etapa el poder alcanzar una aproximación apropiada al diseño. Entre los lenguajes para los sistemas de tiempo real pueden considerarse:

- Con soporte del SO:
  - Lenguajes ensambladores
  - Lenguajes de implementación de sistemas secuenciales (RTL/2, Coral 66, Jovial, C).
- Sin soporte del SO:
  - Lenguajes concurrentes de alto nivel (Ada, Java, Pearl, Modula-2). La especificación Java usada para STR es Java/Real-Time Java.

Frente a la configuración típica, en los sistemas embebidos, los programas de usuario incluyen componentes de sistema operativo para poder interactuar directamente con el hardware.



Configuración típica



Configuración típica de Sistema embebido

Prueba y Validación

Las pruebas deben ser muy exigentes debido a la alta confiabilidad que conforman la esencia de la mayoría de los STR. Una estrategia de prueba completa incluirá muchas técnicas, la mayoría de las cuales son aplicables a cualquier producto de software. Los métodos apropiados de diseño formal no evitan la necesidad de hacer prueba, sino que son estrategias complementarias.

En los STR es importante probar el correcto comportamiento en entornos adecuados, así como la confiabilidad de su comportamiento en un entorno arbitrariamente incorrecto. Las pruebas se pueden ir haciendo a los diferentes módulos del sistema. Así mismo, los problemas más difíciles de tratar surgen de interacciones sutiles entre procesos.