Desarrollo de Software para Sistemas Empotrados

Desarrollo de Software para Sistemas Empotrados

1. Introducción

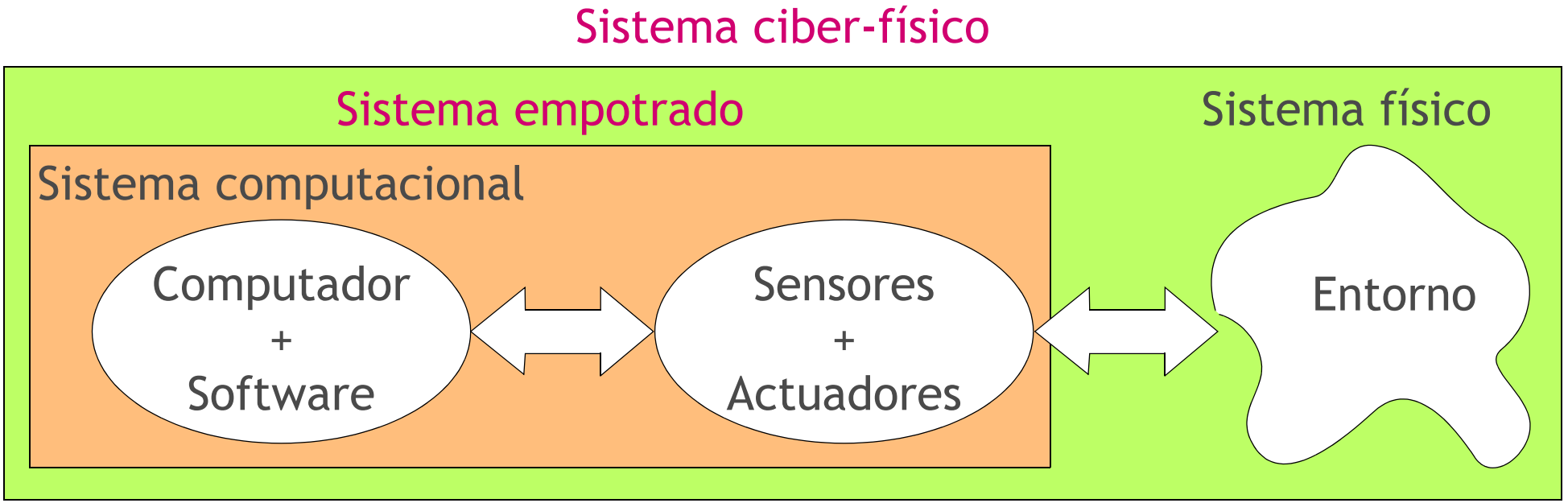
- Sistemas empotrados
- Planificación de las aplicaciones software en sistemas reactivos: dirigida por el tiempo y dirigida por eventos
- Modelos para planificación de recursos que no son de procesamiento: energía, redes de comunicación y memoria
- Variaciones al proceso de desarrollo
- El papel del desarrollo de software dirigido por modelos

1.1 Sistemas empotrados

- IEEE dictionary: Embedded computer system
  - A computer system that is part of a larger system and performs some of the requirements of that system; for example, a computer system used in an aircraft or rapid transit system.
  - A computer (and its software) is considered embedded if it is an integral component of a larger system and is used to control and/or directly monitor that system, using special hardware devices
- ARTIST Roadmap:
  - An embedded system is an electronic programmable subsystem that is generally a integral part of a larger heterogeneous system
- Wikipedia (Extraído de la definición de Sep 2022):
  - An embedded system is a computer system that has a dedicated function within a larger mechanical or electronic system. It is embedded as part of a complete device... Because it typically controls physical operations of the machine that it is embedded within, it often has real-time computing constraints.

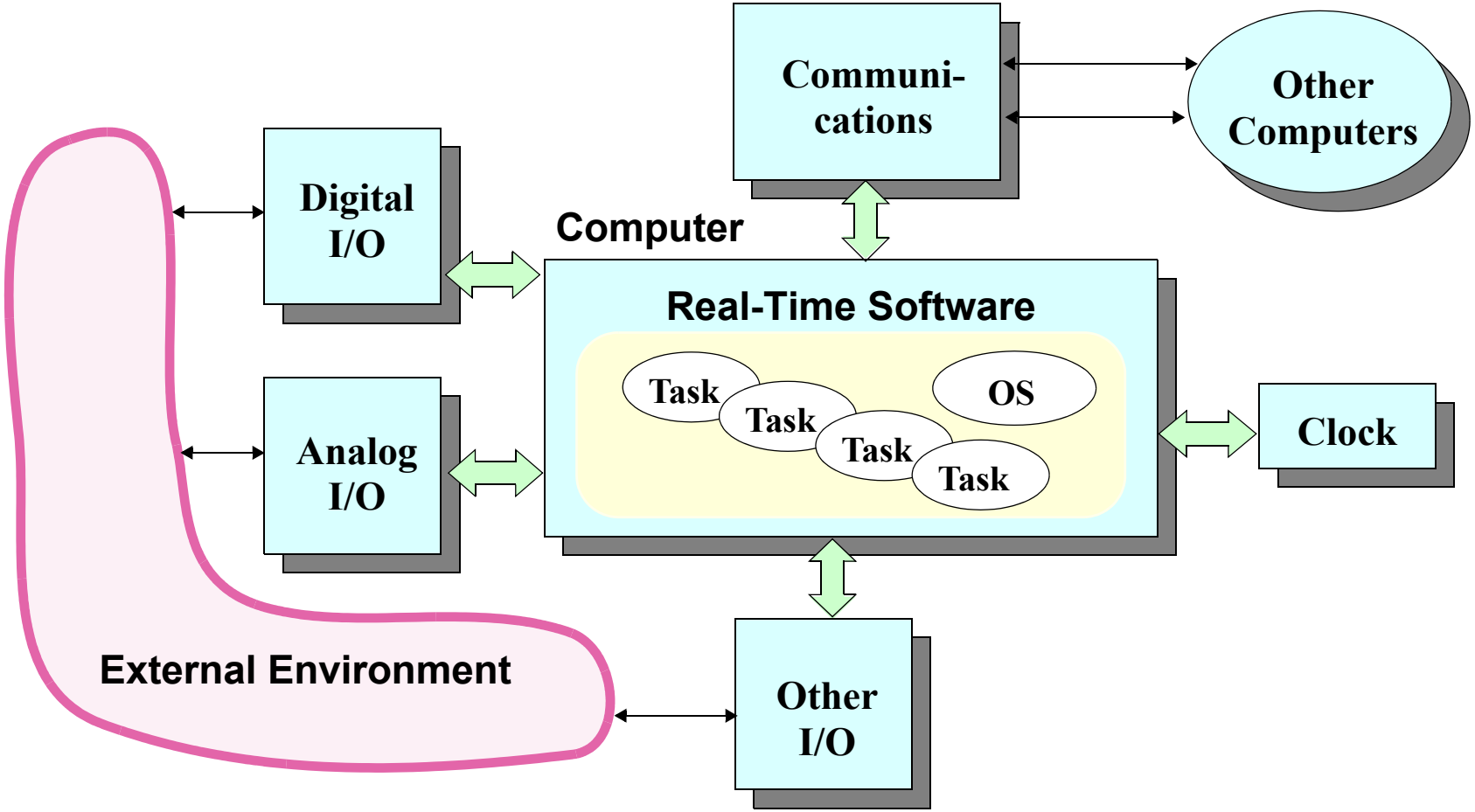
Sistemas ciberfísicos

- Es la terminología "de moda" para designar a los sist. empotrados
- Wikipedia (Extraído de Sep 2022):
  - A cyberphysical system (CPS) is a computer system in which a mechanism is controlled or monitored by computer-based algorithms. Physical and software components are deeply intertwined
- La principal diferencia entre los dos términos:



Ejercicio 1.1. Escribir nuestra propia definición

Interacción con el entorno



Interacción con el entorno

Una característica común a todos los sistemas empotrados es que tienen una fuerte interacción con el entorno

- mediante sensores y actuadores: *sistema reactivo*
- el entorno cambia con el tiempo
- el sistema controla simultáneamente diversos aspectos del entorno

Como resultado:

- el software tiene que acompasar su temporización a la del entorno
  - en muchos casos esto se traduce en requisitos temporales
  - que convierten el sistema empotrado en un *sistema de tiempo real*
  - el comportamiento temporal tiene que ser *predecible*
- el software es *concurrente*

Software reactivo: aquel con interacción entre el software y el entorno

Cuándo es un sistema informático empotrado y de tiempo real

Ejercicio 1.2. Escribir ejemplos de:

- sistemas empotrados
- sistemas que *no* son empotrados

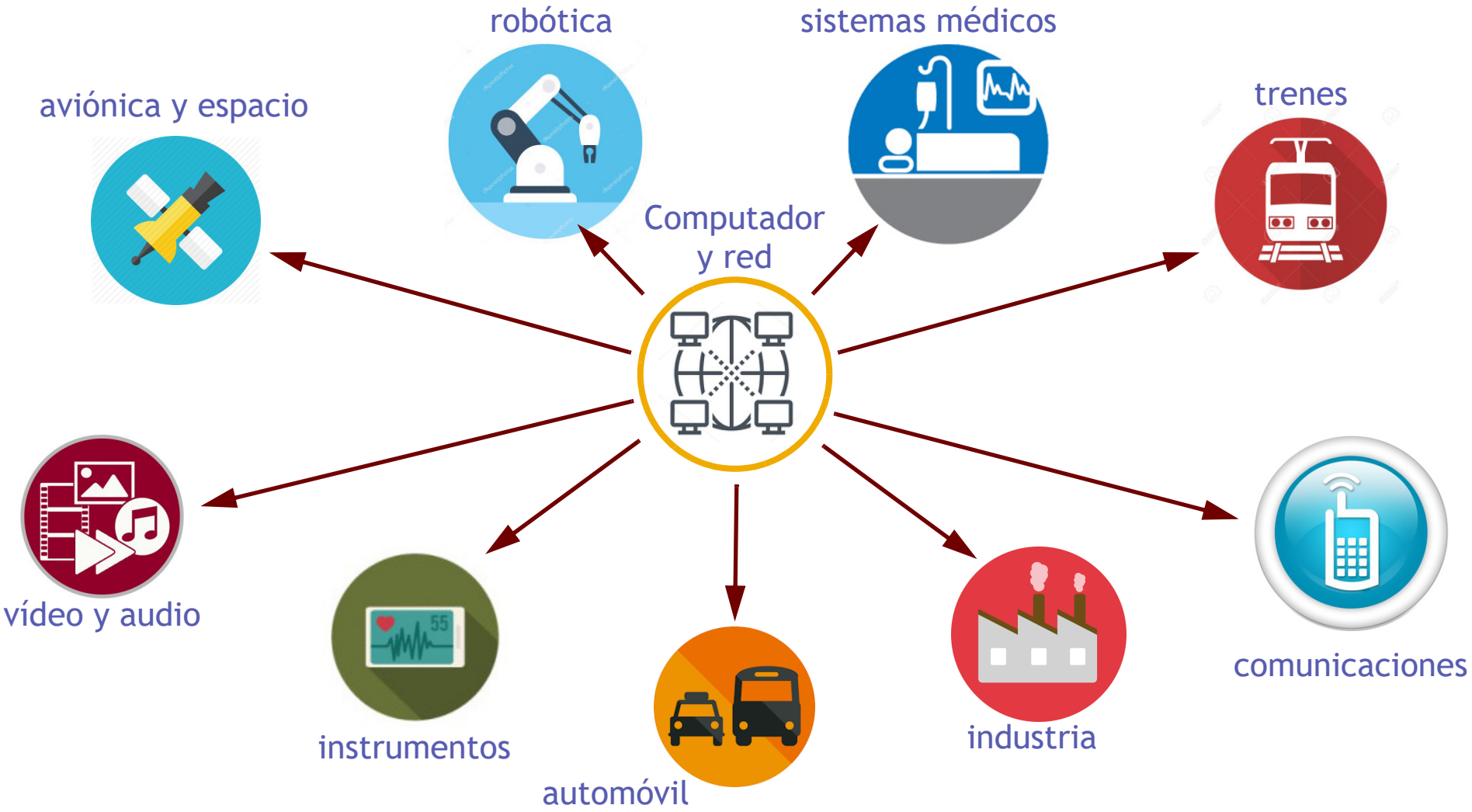
Indicar cuáles de ellos son sistemas de tiempo real

La importancia de los sistemas empotrados

La creciente capacidad de los procesadores (Ley de Moore) ha hecho que los sistemas empotrados reemplacen sistemas puramente mecánicos o electrónicos

Al no cambiar la interfaz humana muchas veces los sistemas empotrados pasan desapercibidos

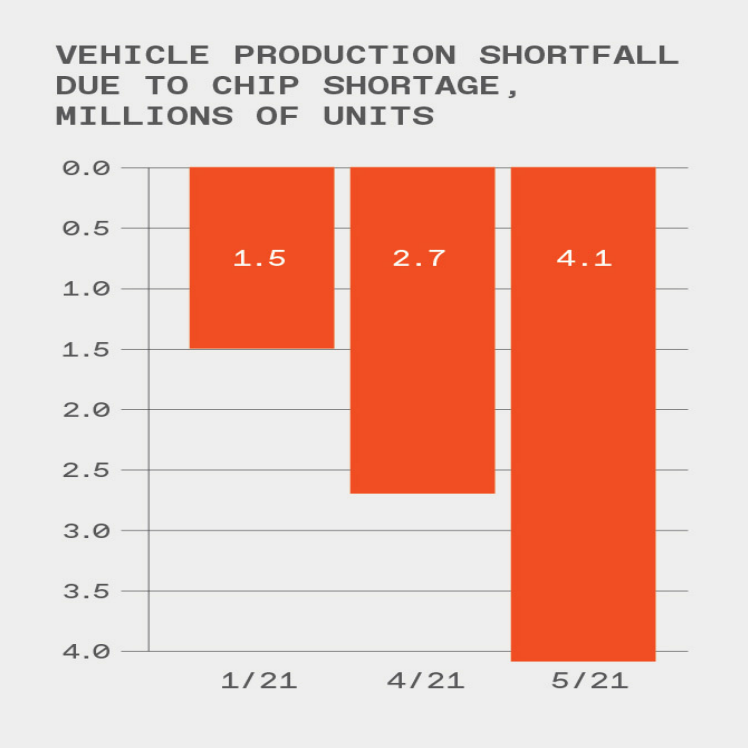
Algunas aplicaciones de los sistemas empotrados



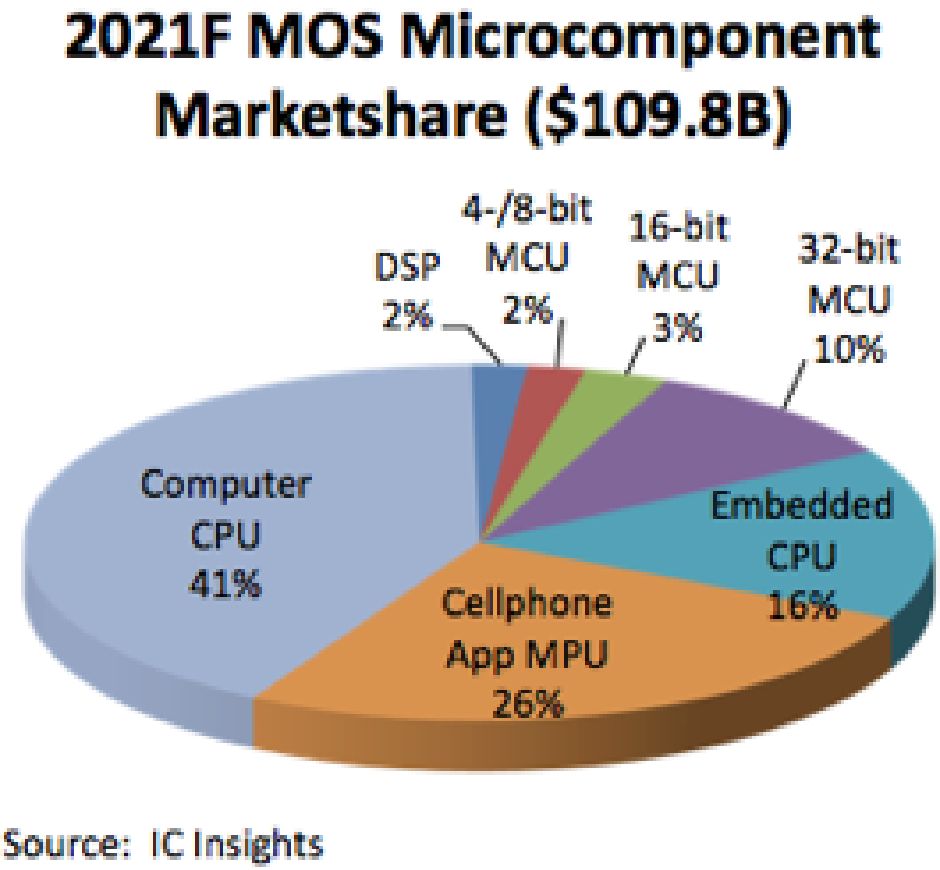
La crisis de los “chips”, 2021-23

IEEE Spectrum:

- 40% of the cost of a new car can be attributed to semiconductor-based electronic systems. This total will approach 50% by 2030. Each new car today has about $600 worth of semiconductors packed into it, consisting of up to 3,000 chips of all types.
- Even low-end vehicles are quickly approaching 100 ECUs and 100 million of lines of code.
- More features such as adaptive cruise control and automatic emergency braking, are becoming standard.

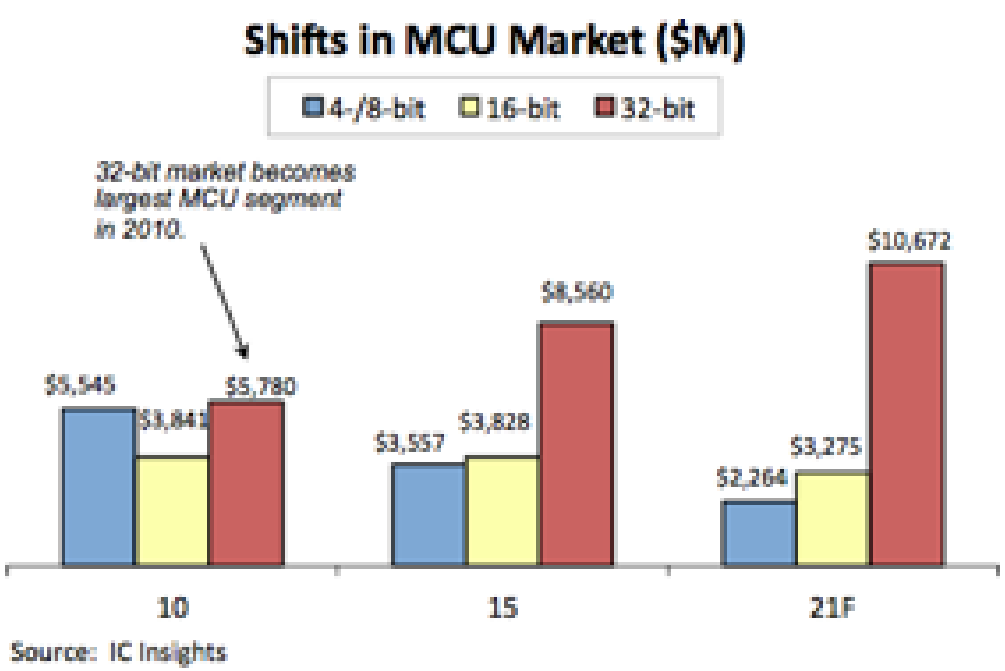


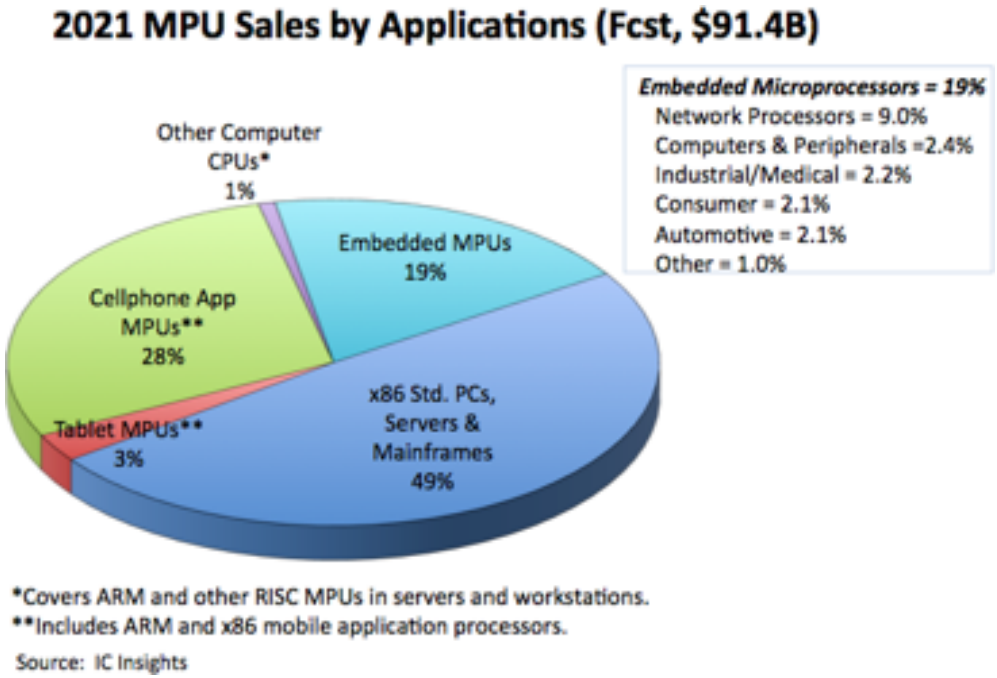
Mercado de los sistemas empotrados



Fuente: http://www.icinsights.com/services/mcclean-report/

Mercado de los microprocesadores





Fuente: http://www.icinsights.com/services/mcclean-report/:

2019: 34 giga unidades de microcontroladores empotrados, frente a 391 M unidades de procesadores para PCs (1.15%)

Internet-of-things

Wikipedia (Extraído de Sep 2022):

- The Internet of things (IoT) describes physical objects (or groups of such objects) with sensors, processing ability, software, and other technologies that connect and exchange data with other devices and systems over the Internet or other communications networks.
- Traditional fields of embedded systems, wireless sensor networks, control systems, automation (including home and building automation), enable the Internet of things.
- In the consumer market, IoT technology is most synonymous with products pertaining to the concept of the "smart home", including devices and appliances (such as lighting fixtures, thermostats, home security systems and cameras, and other home appliances).

IoT= sistemas empotrados + comunicación + gestión integrada

Internet-of-things

Ejercicio 1.3: describir un par de ejemplos de sistemas empotrados:

- uno que se puede considerar un dispositivo IoT
- otro que no

Requisitos habituales de los sistemas empotrados

a) Tiempo real

- debe acompasar su ejecución al funcionamiento del entorno

b) Gestión de recursos escasos

- por motivos de coste, espacio, peso, energía
- recursos como CPU, memoria, energía requieren una gestión eficiente

c) Confiabilidad: fiabilidad, seguridad, ...

- puede haber diversos niveles de criticidad según la gravedad de un fallo de cada componente software del sistema

d) Tolerancia a fallos

- tolerancia a fallos software y/o hardware: robustez y redundancia

1.1.a) Tiempo real

Definiciones (Kopetz):

- *Kopetz*: A real-time computer system is a computer system where the correctness of the system behavior depends not only on the logical results of the computations, but also on the physical time when these results are produced
- *James Martin*: A real-time system "controls an environment by receiving data, processing them, and returning the results sufficiently quickly to affect the environment at that time"

Control del tiempo mediante dos *mecanismos* principales

- *retrasos*: cuando el sistema informático es más rápido que el entorno
- *planificación*: cuando partes del sistema son más lentas que el entorno organizamos cuándo vamos a hacer cada cosa para que dé tiempo de todo

Requisitos temporales

Plazos: tiempo máximo entre la llegada de un evento y el final de su procesamiento

- *llegada*: lectura de una entrada, interrupción o recepción de un mensaje
- *final*: escritura de resultados, envío de mensaje o actuación

Separación mínima o máxima entre eventos: *Jitter* o variabilidad acotados

- existen eventos periódicos cuya llegada puede sufrir una variación temporal acotada pero arbitraria
- ejemplo: un autobús pasa en promedio cada 10 minutos pero puede adelantarse o retrasarse por efectos del tráfico
  - el requisito temporal podría ser un jitter máximo de 2 minutos de atraso o adelanto

Tipos de requisitos de tiempo real

Estricto (Hard)

- El requisito debe cumplirse en todo caso, incluso el más desfavorable (peor caso)

No Estricto (Soft)

- El requisito debe cumplirse con una garantía probabilística
  - por ejemplo, el 99% de las veces en promedio

Firme (Firm)

- El requisito se puede incumplir de forma esporádica
  - por ejemplo una de cada 100 veces
  - en una ventana temporal los incumplimientos deben estar acotados

Si el sistema tiene al menos un requisito estricto se considera que es un sistema estricto

Terminología de los sistemas de tiempo real

Ejercicio 1.4. Proponer definiciones para los siguientes conceptos:

- Paso del tiempo
- Evento
- Tiempo absoluto (instante)
- Tiempo relativo (duración)
- Instante actual, pasado y futuro
- Granularidad del tiempo:
  - conceptos relacionados: resolución y tick

1.1.b) Gestión de recursos escasos

Por motivos de coste, espacio, peso, energía

- CPU
  - *planificación*, para gestionar las actividades de la CPU de la forma más eficiente posible
  - *coprocesadores*, para descargar a la CPU de trabajos que se pueden hacer mediante hardware dedicado
- Memoria
  - planificación de la memoria
  - programación ahorrativa: tamaño de código y de datos
- Energía
  - mecanismos de reducción de potencia
  - planificación consciente de la energía

1.1.c) Confiabilidad

Puede haber diversos niveles de criticidad según la gravedad de un fallo de cada componente software del sistema

Críticos

- El incumplimiento tiene efectos graves
  - para la vida de muchas personas (catastrófico)
  - para la vida de algunas personas (peligroso)
  - efectos económicos de importancia (importante)
  - efectos económicos de menor importancia (menor)

No críticos

- Incumplimiento con efectos leves: degradación de la funcionalidad

Los diferentes niveles de criticidad dan lugar a distintos requisitos de certificación o validación de los sistemas

Niveles de certificación DO-178B

Level	Failure condition	Failure Rate
A	Catastrophic	1.0E-9/hour
B	Hazardous	1.0E-7/hour
C	Major	1.0E-5/hour
D	Minor	1.0E-3/hour
E	No Effect	n/a

Niveles de integridad SIL IEC 61508, para operación continua

SIL: (Safety Integrity Level)

SIL	PF por hora (Probabilidad de fallo)	
1	0.00001-0.000001	10^{-5}-10^{-6}
2	0.000001-0.0000001	10^{-6}-10^{-7}
3	0.0000001-0.00000001	10^{-7}-10^{-8}
4	0.00000001-0.000000001	10^{-8}-10^{-9}

Ejemplo de operación continua: control del vuelo de un avión

Niveles de integridad SIL IEC 61508, para operación de baja demanda

SIL: (Safety Integrity Level)

SIL	PFD (Probabilidad de fallo bajo demanda)	RRF (Factor de reducción del riesgo)
1	0.1-0.01	10-100
2	0.01-0.0001	100-1000
3	0.0001-0.00001	1000-10000
4	0.00001-0.000001	10000-100000

Ejemplo de operación de baja demanda: desfibrilador cardíaco

Factor de riesgo: RRF = Riesgo (sist. base) / Riesgo (sist. modificado)

En determinadas circunstancias, RRF = 1/ PFD

Requisitos de integridad SIL

Ejercicio 1.5. Hacer una breve descripción de las adaptaciones de las normas SIL a las siguientes áreas:

- Automóviles
- Ferrocarriles
- Industria de producción
- Centrales nucleares
- Maquinaria

Hacerlo para una de las áreas, en formato de presentación breve (máximo 4 transparencias)

Aspectos relacionados con la confiabilidad

- *fiabilidad*: probabilidad de que no falle
- *seguridad para las personas* (safety): cualidad de que el fallo del sistema no cause daño
  - evitar el accidente o protegerse de él (un casco)
- *seguridad frente al delito* (security): protección de la información y de la operación frente a intrusiones
  - evitar el delito o protegerse de él (una cerradura)
- *mantenibilidad*: facilidad para que el sistema sea reparable tras un fallo
- *disponibilidad*: probabilidad de que el sistema esté disponible cuando hace falta

1.1.d) Tolerancia a fallos

Solo los sistemas empotrados más críticos tienen requisitos de tolerancia a fallos

- En especial si no tienen un modo seguro ante el fallo ('fail-safe')

Fallos temporales o permanentes

- *Temporales*: robustez para darse cuenta de que ha ocurrido un error y tomar medidas correctoras
  - suelen ser fallos hardware
- *Permanentes*: se resuelven con redundancia
  - puede ser de software o de hardware

1.2 Planificación de las aplicaciones software en sistemas reactivos

Dos paradigmas principales para la planificación

- disparada por el tiempo (*time triggered*)
  - predecible a través de un plan estático
  - poco flexible
  - difícil gestionar actividades aperiódicas y cambios dinámicos
  - más difícil de mantener
  - más fácil de certificar
- conducida por eventos (*event driven*)
  - planificación por prioridades
  - planificación expulsora o no expulsora
  - se necesitan métodos analíticos para garantizar la predictibilidad
  - más flexible frente a cambios y gestión de eventos aperiódicos
  - más difícil de certificar

Planificación

Influye en la arquitectura interna de la aplicación, no en su carácter reactivo de interacción con el entorno

En sistemas conducidos por eventos

- La ejecución de una actividad se inicia con la llegada de un evento que proviene de una interrupción hardware o software, de la llegada de un mensaje o de un cambio de estado significativo
- Se requiere una planificación dinámica que decide qué proceso o tarea servirá al evento y cuándo

En sistemas disparados por el tiempo

- Todas las actividades se inician por el progreso del tiempo
- Solo hay una interrupción en cada nudo: la del reloj
- En sistemas distribuidos debe haber sincronización de relojes

Ejemplos con un software de control de un sistema de ascensores

Conducido por eventos:

- cuando el usuario pulsa el botón de llamada se produce una interrupción que es atendida inmediatamente por el sistema

Ejemplo (cont.)

Disparado por tiempo:

- Diseño 1:
  - cada cierto tiempo (por ejemplo cada 50ms) el sistema comprueba si el usuario está pulsando el botón de llamada o no
  - Si detecta que lo está pulsando inicia las acciones necesarias
- Diseño 2:
  - cuando el usuario pulsa el botón un circuito de memoria almacena este hecho
  - cada cierto tiempo (por ejemplo cada 0.5s) el sistema comprueba si se pulsó el botón de llamada o no
  - si detecta que fue pulsado resetea el circuito de memoria e inicia las acciones necesarias

1.3 Planificación de recursos que no son de procesamiento

a) Energía

b) Redes de comunicación

c) Memoria

a) Planificación del consumo energético

Muchos sistemas se basan en baterías, donde la energía es limitada e interesa operar a baja potencia para maximizar la vida de la batería

Los sistemas espaciales tienen su potencia limitada por los sistemas de generación disponibles:

- paneles solares, pilas de radioisótopos, ...

El consumo energético se puede controlar en muchos sistemas

- control basado en voltaje de la CPU
- control basado en la velocidad de la CPU
- apagado independiente de dispositivos que no estén en uso
- apagado independiente de algunos núcleos en sistemas multinúcleo

Consumo energético en la CPU

El control del voltaje suele ser parejo al de la velocidad

- Es habitual una dependencia casi lineal entre la frecuencia y el voltaje
  - si la frecuencia se reduce el voltaje también se puede reducir
  - como la potencia depende linealmente de la frecuencia y del cuadrado del voltaje, el efecto combinado reduce no solo la potencia sino la energía necesaria para hacer un determinado cálculo
- Ejemplo: el procesador Intel Xscale puede operar entre 0.7 a 1.75 v y entre 150 y 800 MHz. El ahorro de energía tiene un factor de 6.3

La circuitería para controlar la frecuencia y el voltaje es compleja y suelen darse solo dos posibilidades, potencia alta o baja, seleccionables por software

Consumo energético en la CPU

El sistema operativo puede integrar la gestión de energía con la planificación de tareas

- *Planificación consciente de la energía*: Si se conoce el tiempo de ejecución de peor caso de una tarea a frecuencia alta y sobra tiempo para finalizarla dentro de su plazo se puede reducir la frecuencia hasta que sea necesario aumentarla para finalizar justo a tiempo

b) Planificación en redes de comunicación

En muchos sistemas empotrados se usan redes convencionales (Internet sobre cable o Wifi) pero estas redes generalmente no ofrecen garantías de tiempo real

Problemas con la resolución de colisiones

Con determinadas restricciones se puede garantizar tiempo real

- UDP vs TCP
- Ethernet punto a punto
- Ethernet con tráfico regulado
  - RT-EP, FTT-Ethernet, ...
- Switches Ethernet con prioridades
- WiFi sin interferencias y manteniendo cobertura

Redes de campo

Existen redes de campo (*field buses*) para sistemas industriales, con comportamiento de tiempo real

Ejemplos:

- *CAN Bus*: prioridades fijas
- *EtherCAT*: cada nudo inserta su información en un frame enviado desde el máster
- *Profibus*: paso de testigo temporizado (timed token)
- *AFDX*: Ethernet punto a punto, switch con dos prioridades, round-robin
- *TTP-Ethernet*; basada en disparo por tiempo con un plan preestablecido
- *FlexRay*: ciclo temporal con una parte estática (disparada por el tiempo) y una dinámica (al estilo CAN)

Ejercicio 1.6

Ejercicio 1.6. Hacer una breve presentación de las características de una de las siguientes redes de campo:

- *EtherCAT*
- *Profibus*
- *AFDX*
- *TTP-Ethernet*
- *FlexRay*
- *Local Interconnect Network (LIN Bus)*
- *MOST communication protocols*

Máximo 4 transparencias

Entregar: el material de presentación

Middleware de comunicaciones

Necesidad del software de intermediación para evitar el paso de mensajes explícito

- facilita la reconfiguración del código
- escalabilidad

Necesidad de adaptar el middleware para que opere con características de tiempo real

- RT-CORBA
- Java RMI-QoS
- Ada DSA
- DDS
- MQTT
- OPC UA

c) Planificación de la Memoria

La memoria principal suele ser limitada

En muchos casos no hay sistemas de memoria secundaria ni de memoria virtual

En otros casos la memoria secundaria solo sirve para cargar la aplicación y/o datos de configuración al inicio y no se usa durante la operación

A veces una memoria ROM guarda las instrucciones de programa y las constantes y la pequeña memoria principal se usa solo para los datos

Arquitectura de la memoria

La arquitectura de memoria es uno de los principales problemas en arquitecturas modernas por la impredecibilidad que introduce

- cachés de varios niveles
- memoria virtual
- mecanismos de coherencia de caches en sistemas multinúcleo y muchos-núcleos

Existen trabajos que proponen controladores de memoria con características de tiempo real

- sin embargo su coste es muy alto

Habitualmente hay que recurrir a HW estándar para reducir costes

Arquitectura de la memoria (cont.)

En la práctica en sistemas de alta integridad se bloquean cachés, se anula la memoria virtual y/o se usa un solo núcleo

- Hay muchos trabajos que intentan eliminar estas limitaciones

Cuando la memoria es escasa se utilizan mecanismos de planificación que reducen su uso

- reducción del número de tareas (y de stacks)
- optimizaciones en compiladores

1.4. Variaciones al proceso de desarrollo

En la especificación de requisitos

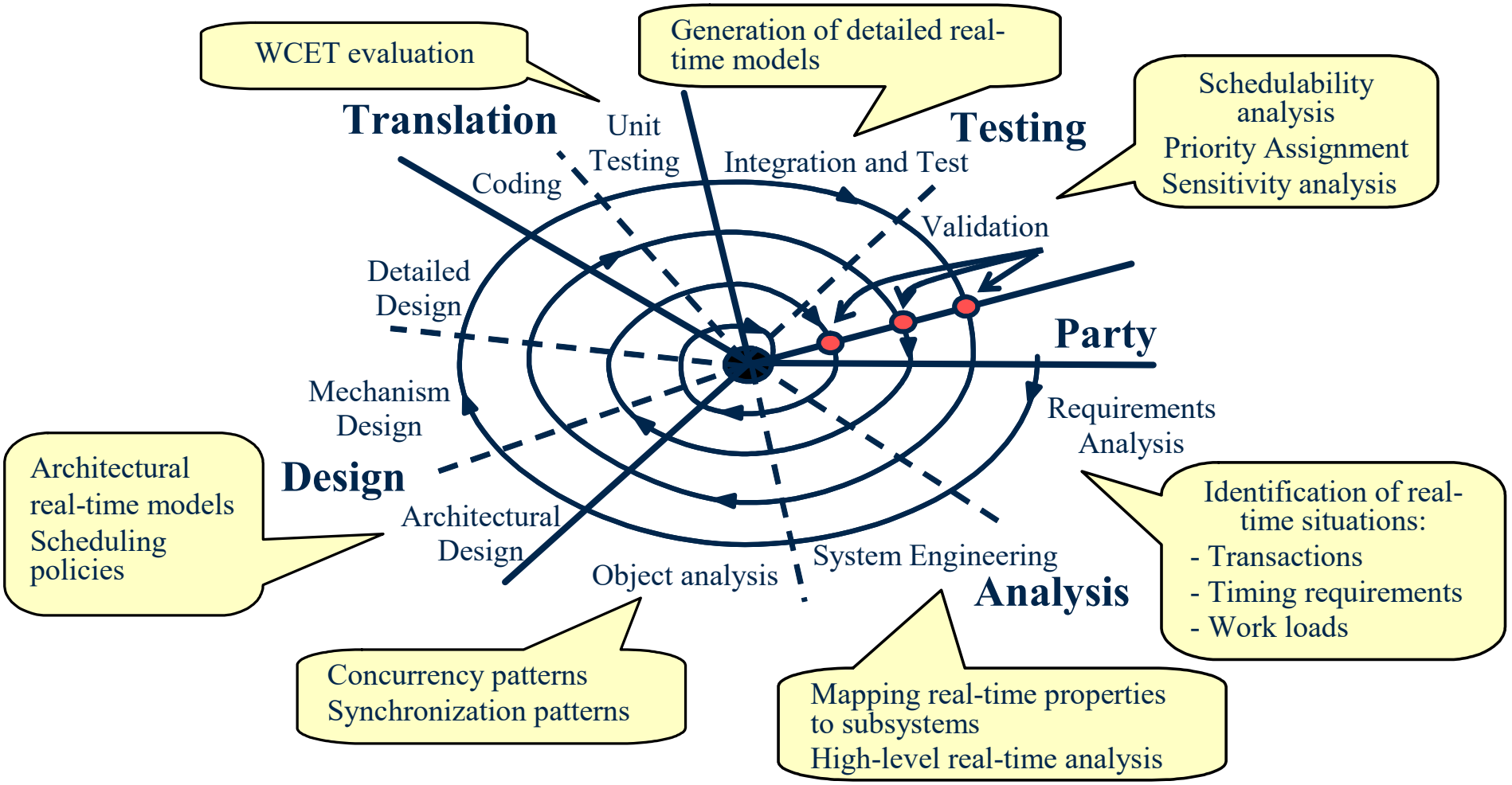
En la elección de la plataforma

En el diseño

En la implementación

En la validación

Variaciones relativas al tiempo real



Ejemplo

Ver ejemplo del sistema SCADA

1.5. El papel del desarrollo de software dirigido por modelos

Técnica de gestión de la complejidad: Abstracción por medio de *modelos*

- Un modelo es una representación abstracta de un sistema o de uno de sus aspectos, centrada únicamente en los elementos de interés
- El modelo oculta detalles innecesarios en cada nivel de diseño
- "Dame un modelo y explicaré el mundo"

Técnica para evitar errores: *Correcto-por-construcción*

- Obtención de la implementación por medios automáticos formalmente verificados a partir de un modelo de su especificación

Esto da lugar al desarrollo de software dirigido por modelos (MDSD)

Ingeniería dirigida por modelos (MDE)

Tesis de César Cuevas (UC, 2016):

- "En MDE los procesos de desarrollo se conciben como una serie de pasos mediante los que los modelos relativos a la especificación y que describen el dominio del problema, van siendo refinados hasta conducir a aquellos que constituyen el dominio de la implementación, así como hasta aquellos otros que constituyen la verificación y validación de cada uno de ellos y de la correspondencia entre ambos.
- En MDE, los pasos del proceso de desarrollo son considerados como simples transformaciones entre modelos."

El desarrollo de software dirigido por modelos (MDSD) es una instancia de MDE

- aplicada al mundo del software

Impacto del MDSD

Se está confirmando como una tecnología muy prometedora

Se han desarrollado diversos lenguajes de modelado que facilitan la adopción de estas tecnologías

- lenguajes de propósito general
- lenguajes específicos de dominio

Los propios modelos se pueden describir con otros lenguajes

- ello da lugar a los *metamodelos*

Los propios metamodelos se pueden describir con otros lenguajes

- lenguajes de metamodelado
  - usualmente basados en diagramas de clases de UML
  - permiten escribir *meta-metamodelos*

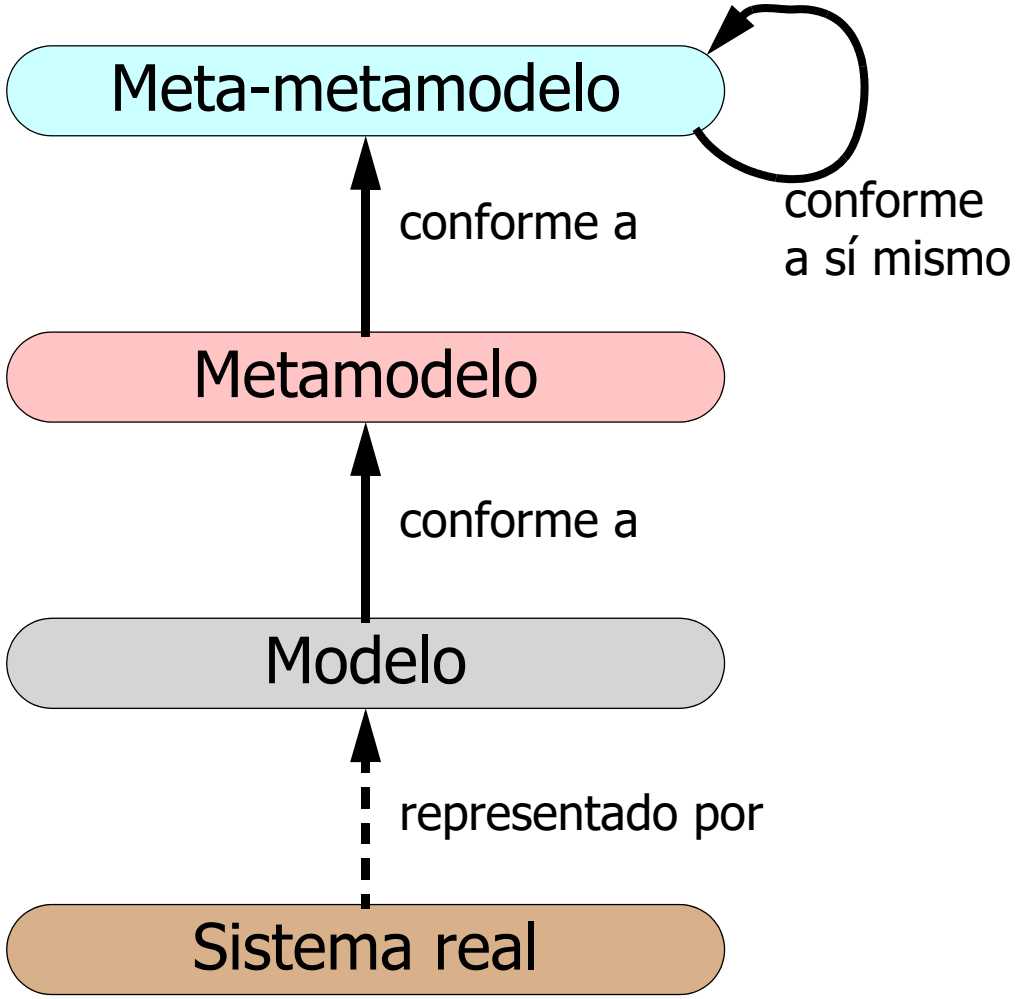
Arquitectura de metamodelado de 3+1 capas

M3: meta-metamodelos que describen los metamodelos de M2

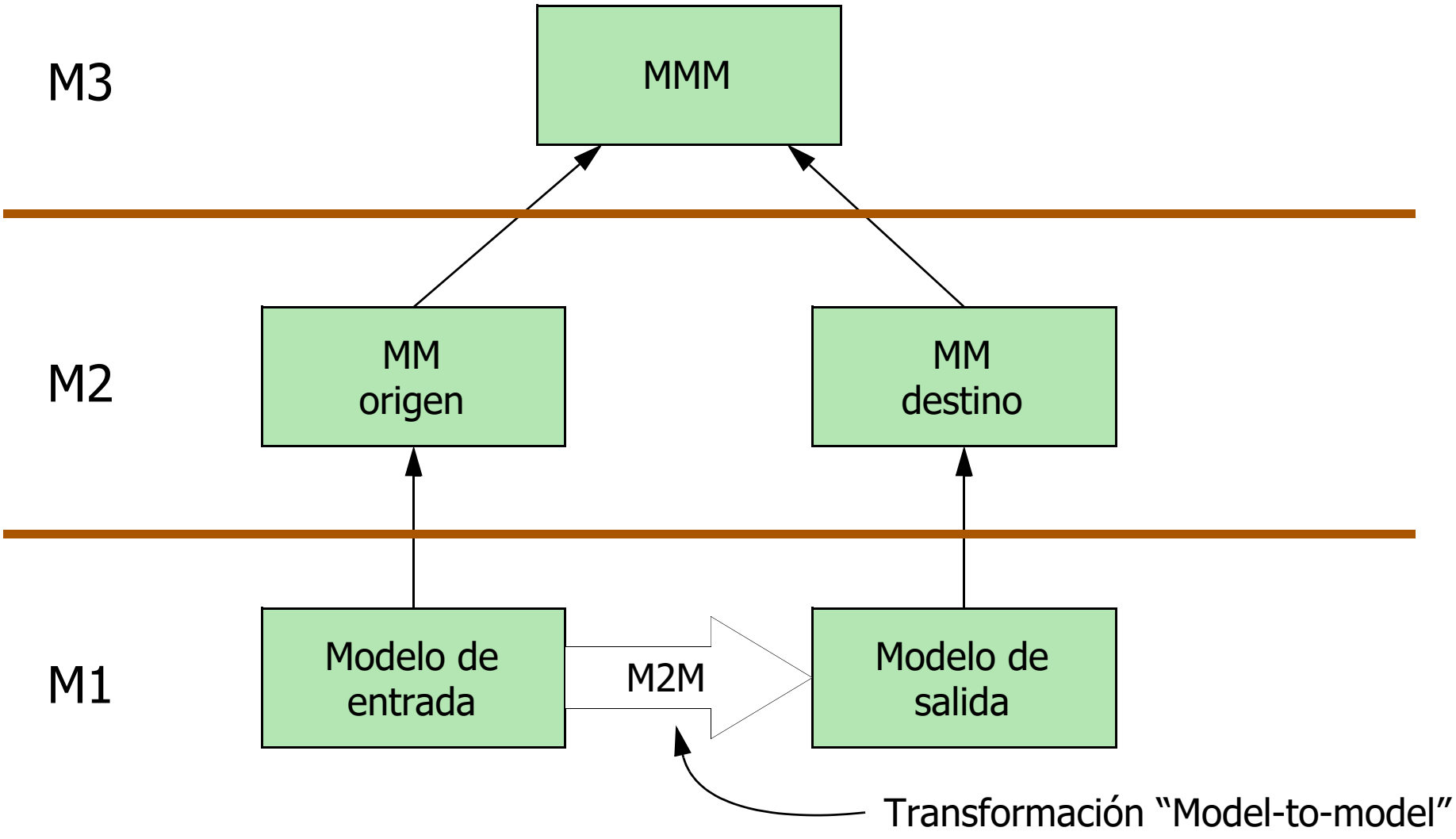
M2: metamodelos que describen los modelos de M1

M1: modelos que representan entes del mundo real

M0: entes del mundo real



Transformaciones de modelos



Lenguajes de transformación de modelos

Ejercicio 1.7: Hacer una breve presentación (5 minutos) sobre un lenguaje de transformación de modelos

Por ejemplo:

- ATL
- QVT
- AToM3
- MOF/Acceleo
- Epsilon/ETL
- Tefkat
- VIATRA

Poner un ejemplo sencillo de una de ellas en una transparencia y explicar lo que hace

Conclusiones sobre MDSD

Progresiva adopción y consolidación en la última década

Ventajas

- Eleva los niveles de abstracción del proceso de construcción del software
- Automatiza la implementación en búsqueda del concepto correcto-por-construcción
- Aumenta la productividad
- Mejora la descripción de múltiples vistas del sistema

Inconvenientes

- curva de aprendizaje
- al ser un proceso basado en herramientas éstas tienen un coste alto de desarrollo

Ejercicio 1.8: Sistema SCADA: Modelado de tiempo real

- Ver enunciado en la sección de “problemas”

Bibliografía

[1] “Embedded System Design”. Springer, 2010

[2] “Real-Time Systems: Design Principles for Distributed Embedded Applications”. Hermann Kopetz. Springer, 2011

[3] “Model-Based Engineering of Embedded Real-Time Systems”: International Dagstuhl Workshop, Dagstuhl Castle, Germany, November 4-9, 2007. Revised Selected Papers. Holger Giese, Gabor Karsai, Edward A. Lee, Bernhard Rumpe, Bernhard Schätz. Springer, 2010

[4] “Metaherramientas MDE para el diseño de entornos de desarrollo de sistemas distribuidos de tiempo real”. César Cuevas Cuesta. Tesis doctoral, Universidad de Cantabria, 2016