DOCENCIA EN INFORMÁTICA INDUSTRIAL: LENGUAJES DE PROGRAMACIÓN

Rogelio Mazaeda, Eusebio de la Fuente,
José Luis González, Eduardo Moya

Dpto. ISA, Universidad de Valladolid

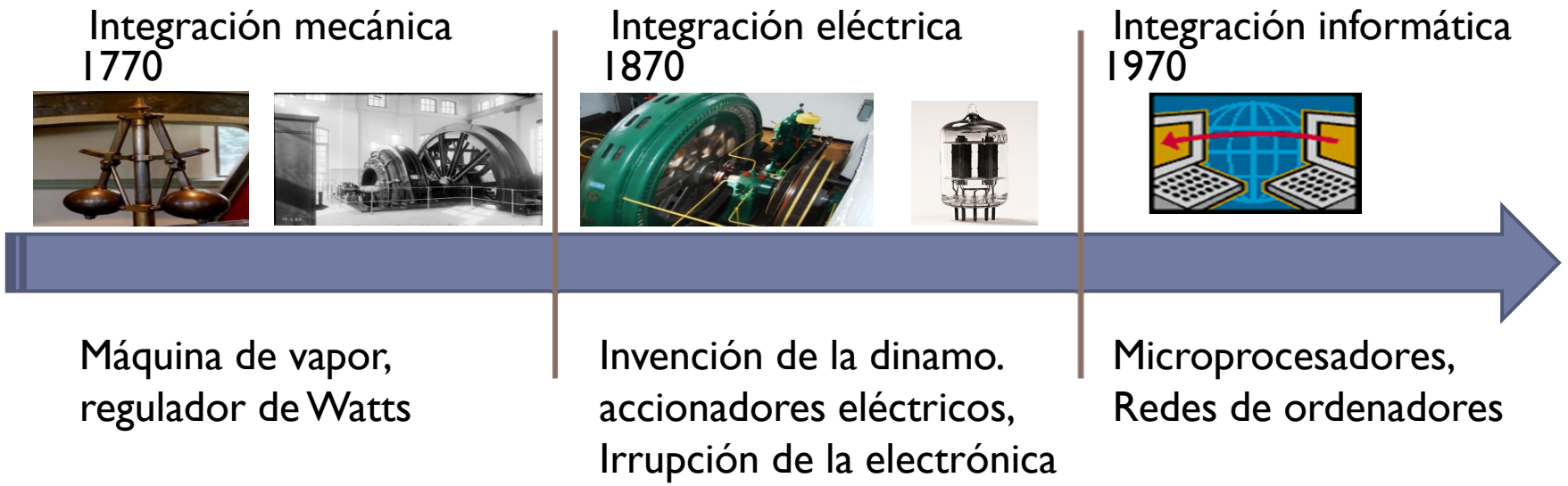
Reunión Grupo Temático Educación en Control. XXXVII Jornadas de Automática. Madrid

Antecendentes

- Los autores vienen participando en los dos últimos años en un **Proyecto de Innovación Docente** para reflexionar sobre los contenidos y la forma de impartirlos en las asignaturas relacionadas directamente con la **Informática Industrial** en el **Grado de Ingeniería en Electrónica Industrial y Automática** (GIEIA) de UVA.

Introducción

La **Informática** es imprescindible en la industria, en la que ocupa el lugar de **tecnología integradora** que antes ocuparon las tecnologías **mecánica** y la **eléctrica**.

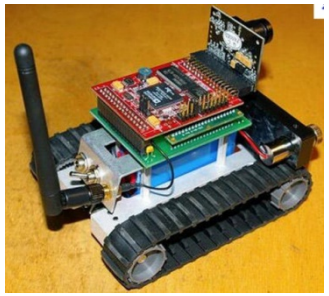


Introducción

Es inconcebible la industria contemporánea sin la informática.

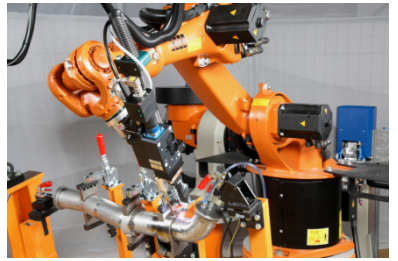
Programas a diferentes niveles en el control de la industria

Programas empotrados conformando el producto



CAD/CAM/CAE (ciclo de vida del producto, simulación)

Programación de robots



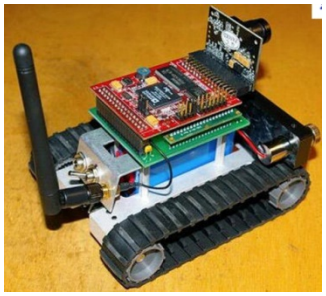
¿Cómo abordar ese estudio?

Introducción

Nos referiremos a aquellas aplicaciones de la informática que implementan **sistemas reactivos** y que interactúan directamente con el mundo físico **controlando procesos industriales** o conformando un **producto "inteligente"**.

Programas a diferentes niveles en el control de la industria

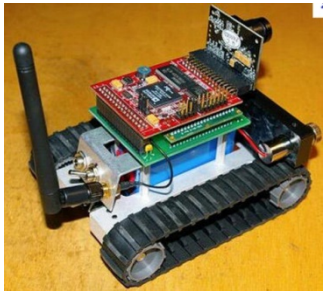
Programas empotrados conformando el producto

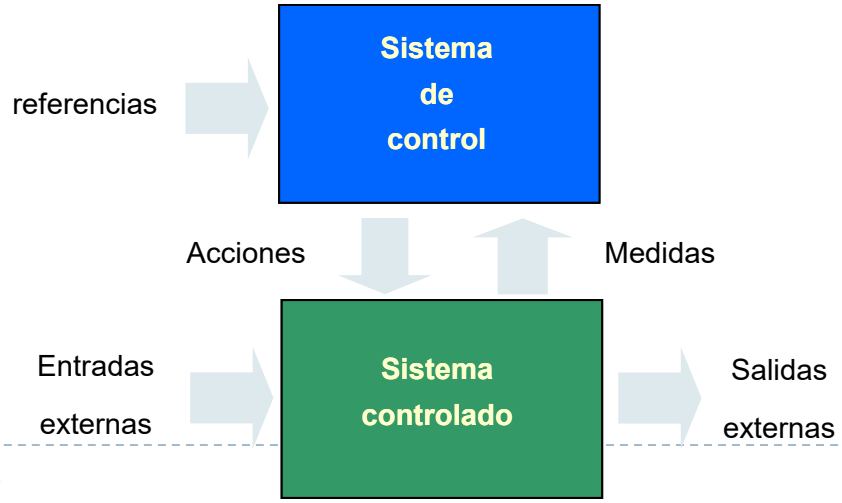


Introducción

Programas a diferentes niveles en el control de la industria

Programas empotrados conformando el producto





Sistemas reactivos:

- Lidian con el **mundo físico** directamente a través de **sensores** y **actuadores**.
- Los **eventos de interés** ocurren **simultáneamente** y en muchos casos en lugares **espacialmente separados**.
- La reacción de sistema informático debe cumplir determinados **plazos de tiempo (*deadlines*)**.
- Un error en estas aplicaciones (salida incorrecta o fuera de plazo) puede causar **perjuicios** que pueden ser **muy graves en vidas y bienes materiales**.

La Informática Industrial en GIEIA

Consideramos que el **graduado de GIEIA** debe tener las competencias necesarias para **participar activamente** y **estar en el centro** del importante momento de que vive la **informática en la industria y** de la **renovación** que se espera.

Retos de la disciplina:

- **Amplitud y diversidad**.
- Carácter **dinámico**.
- La **dificultad** intrínseca de la programación.

La Informática Industrial en GIEIA

Retos de la disciplina:

- **Amplitud y diversidad**.
- Carácter **dinámico**.
- La **dificultad** intrínseca de la programación.

Estrategias de solución:

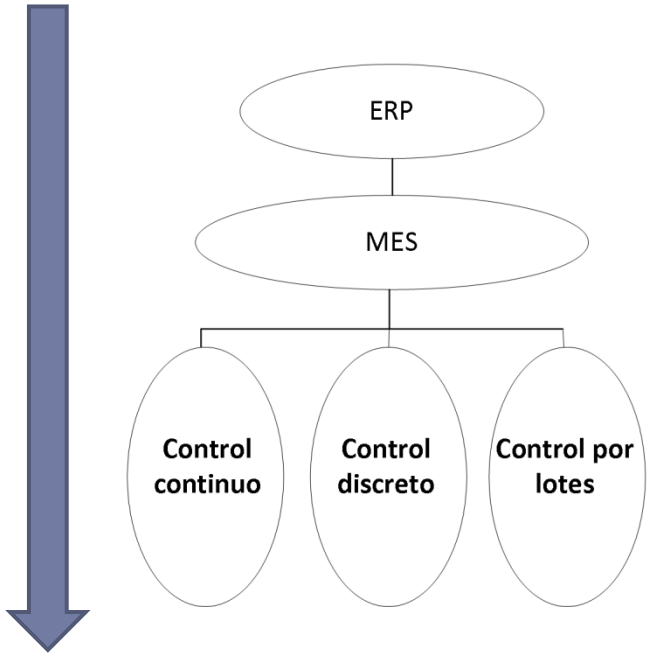
- **Acotar** el núcleo básico**,** buscar un **hilo conductor y coordinar asignaturas afines** .
- Identificar conocimiento de valor **permanente**, apoyarnos en **normas** existentes.
- Elegir el **lenguaje** y el nivel de **abstracción** adecuado.

Introducción

En la industria el modelo jerárquicos normado en ISA-95 provee una buena guía para el estudio de la informática en la fábrica.

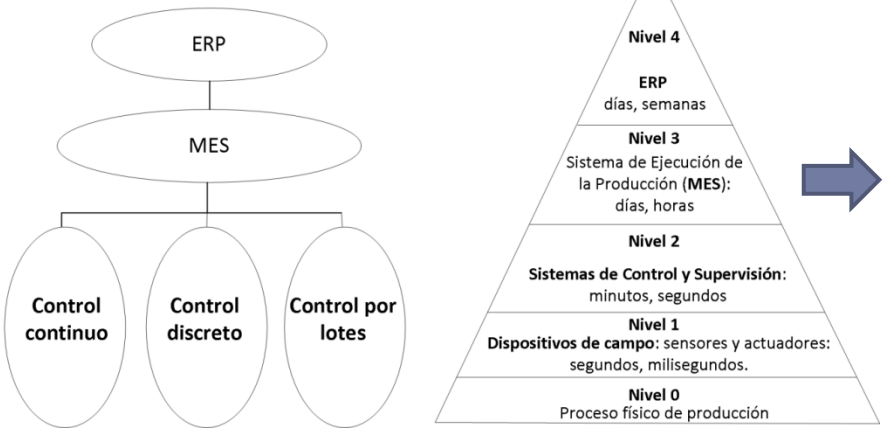
Plazos más amplios y menos estrictos, en los niveles más altos predominio de la informática convencional

Plazos más cortos y estrictos



Introducción

Se anuncia un nueva **revolución industrial** (**Industria 4.0** o *smart factory*)



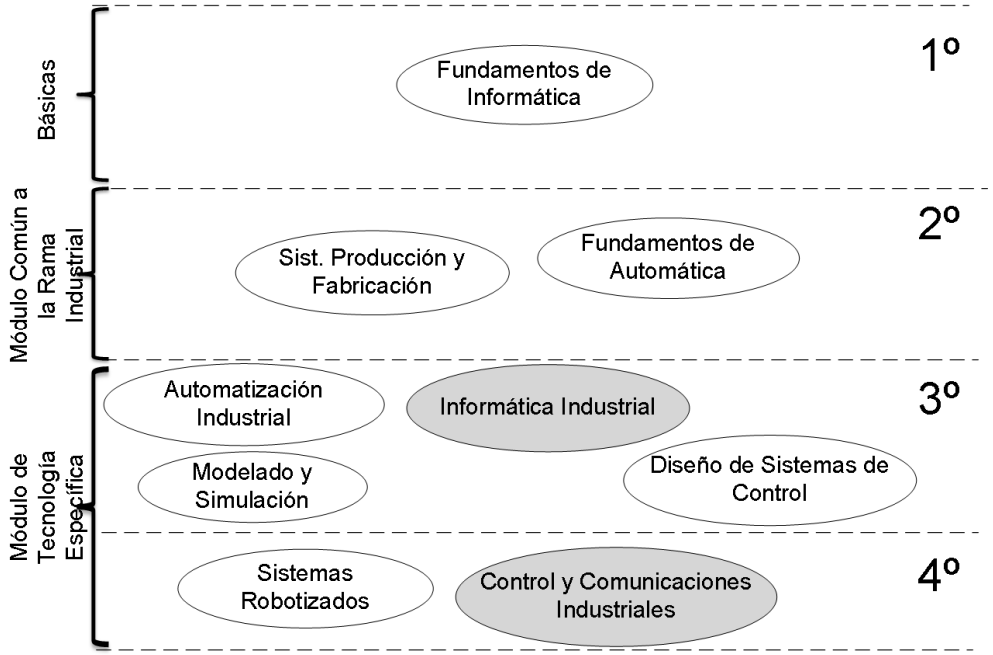
Industry 4.0:

- Una estructura menos jerarquizada, **más flexible**, reconfigurable en línea abarcando amplias áreas geográficas.
- Revalorización del **operario humano**.
- **Sistemas cyber-fisicos**: el sistema de control digital y el controlado (discreto o continuo/muestreado o híbrido) considerado como un todo.
- Internet de las cosas: **IoT**
- Preponderancia de las **redes**, especialmente las **inalámbricas**.
- Acceso a muchos datos redundantes desde múltiples fuentes (**Big Data**).
- Computación basada en la nube (**cloud computing**)

La Informática Industrial en GIEIA

¿Cuáles son las asignaturas que hemos considerado como directamente relacionadas con la Informática Industrial?

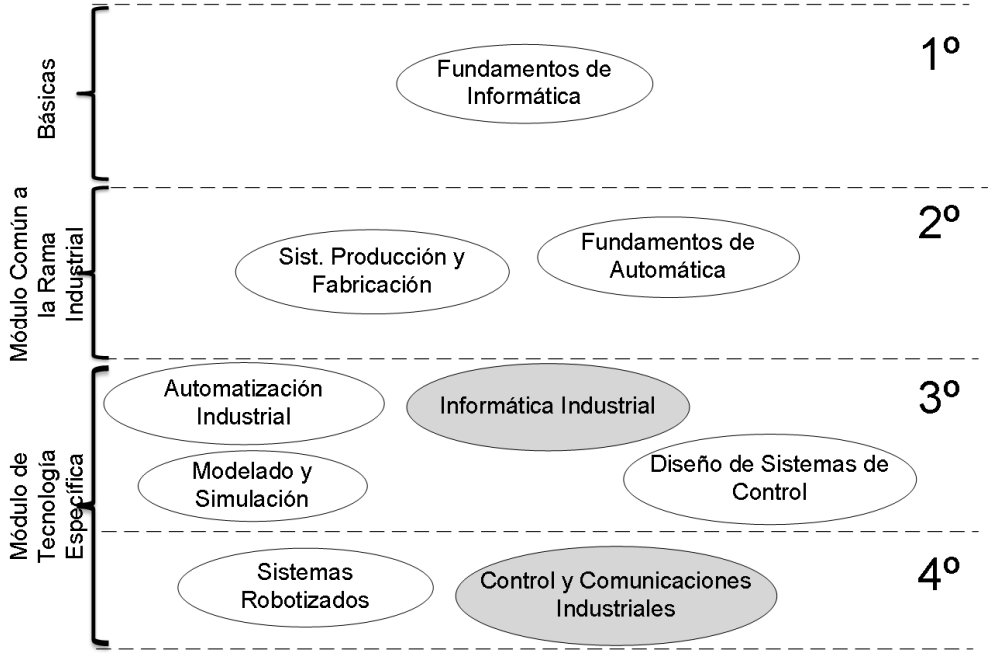
- **Informática Industrial** (II)- 6 crédito ECTS
- **Control y Comunicaciones Industriales** (CCI)- 6 créditos ECTS



Parece razonable:

- **Coordinar** los temas de la disciplina entre ambas asignaturas teniendo en cuenta el **contexto**.
- Tomar a **II** como **introductoria** de **CCI**.
- El alumno de ver se capaz de ver ambas asignatura como un todo integrado y situarse en que parte de la contenido se encuentra.

La Informática Industrial en GIEIA



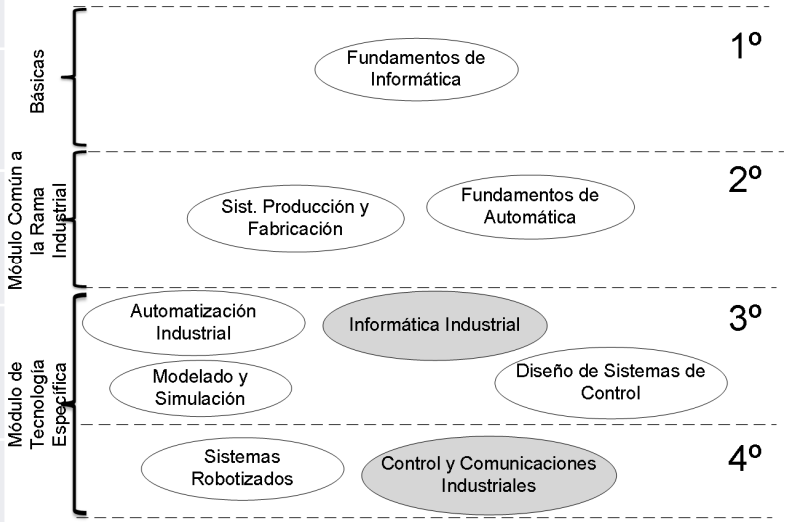
Contexto:

- **Fund. Informática** utiliza el lenguaje C. (*)
- **Fund. Automática** teoría de control en el campo **continuo**.
- **Diseño de Sistemas de Control** da elementos de control discreto (transformada z) se imparte posterior a **II** y previo a **CCI**.
- **Automatización industrial** estudia **autómatas programables**.

(*) Este año comenzaremos a dar **C++ imperativo** en un sentido similar a como se usa en término en **Fundamentos de Programación** de J.A. Cerrada y M.E. Collado

La Informática Industrial en GIEIA

T	Descripción
1	**Contexto Industrial**: El alcance de la II. La pirámide ISA-95. Industria 4.0
2	**Modelado Formal de Sistemas Discretos**: redes de Petri, máquinas de estado finitas, …a nivel muy introductorio.
3	**Sistemas muestreados**: teorema de Shannon, ecuaciones en diferencias, transformada z, filtros anti-alias, *jitter*.
4	**Concurrencia**: Arquitectura de ordenador y soporte del S.O y del lenguaje en relación a la concurrencia. Procesos e hilos. Sincronización y comunicación entre tareas.
5	**Sistemas de Tiempo Real (STR)**: Discusión de alternativas y algoritmos de planificación de tareas.
6	**Tolerancia a fallos y Seguridad**
7	**Comunicaciones Industriales y Sistemas Distribuidos**
8	**Sistemas de Control y Supervisión Industriales**: Control digital directo, autómatas programables, sistemas DCS y SCADA. Ejemplos de aplicación. Estándares IEC 61-131-3, 61499

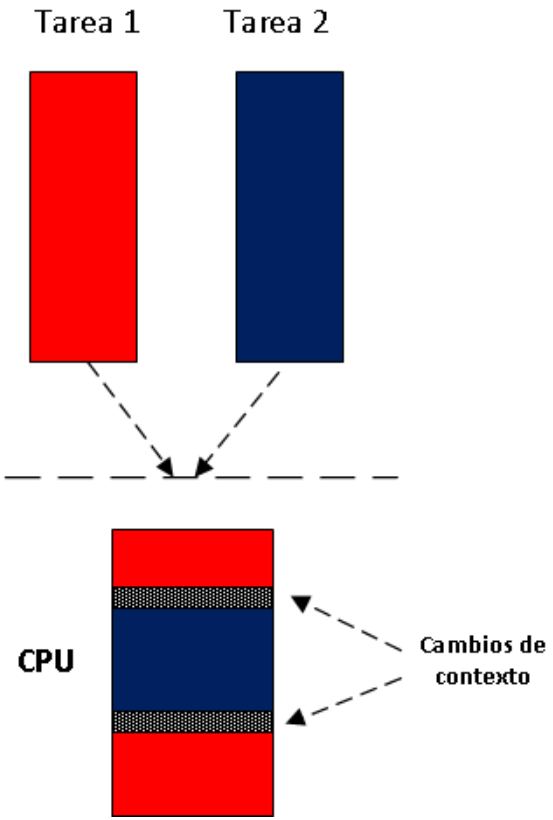


II: Se da **visión general** de la disciplina pero se hace énfasis en el estudio de la **programación concurrente** aplicados a **sistemas discretos**.

CCI: El énfasis en **STR** aplicados a sistemas **continuos** o **híbridos** y en las **comunicaciones industriales**.

Programación concurrente en II

Es común considerar la **concurrencia** como un paradigma adecuado para estos sistemas reactivos:



1 CPU

No hay simultaneidad real.

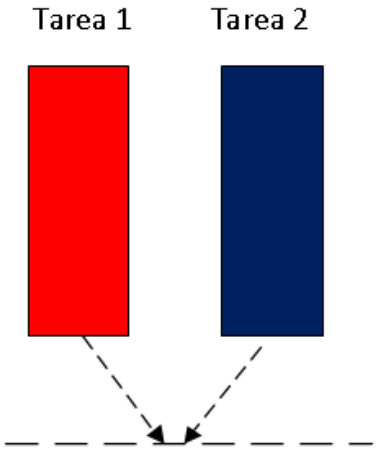
Ventajas

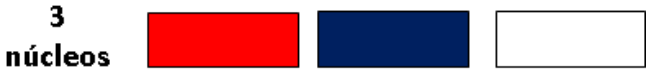
- **Independencia de cometidos.**
- **Menor latencia.**
- Brinda la granularidad para que los STR realicen su programación.

Desventajas

- **Pérdida de capacidad de cómputo** útil (cambios de contexto)
- **Mayor dificultad** de programación
- Cuando las tareas necesitan intercambiar información o sincronizarse se debe evitar:
  - Condiciones de competencia (*race conditions*)
  - Interbloqueos (*deadlocks*)

Programación concurrente en II





Varios CPU

Puede haber simultaneidad real si N tareas < # núcleos y por tanto mayor capacidad de computo (paralelismo).

En caso contrario tendremos situación similar.

La comunicación entre procesos sigue siendo un problema:

$$P = \frac{1}{f_s + (1 - f_s)/N}$$

Ley de Amdahl *

N: número de CPUs o núcleos

Fs: fracción del código compartido

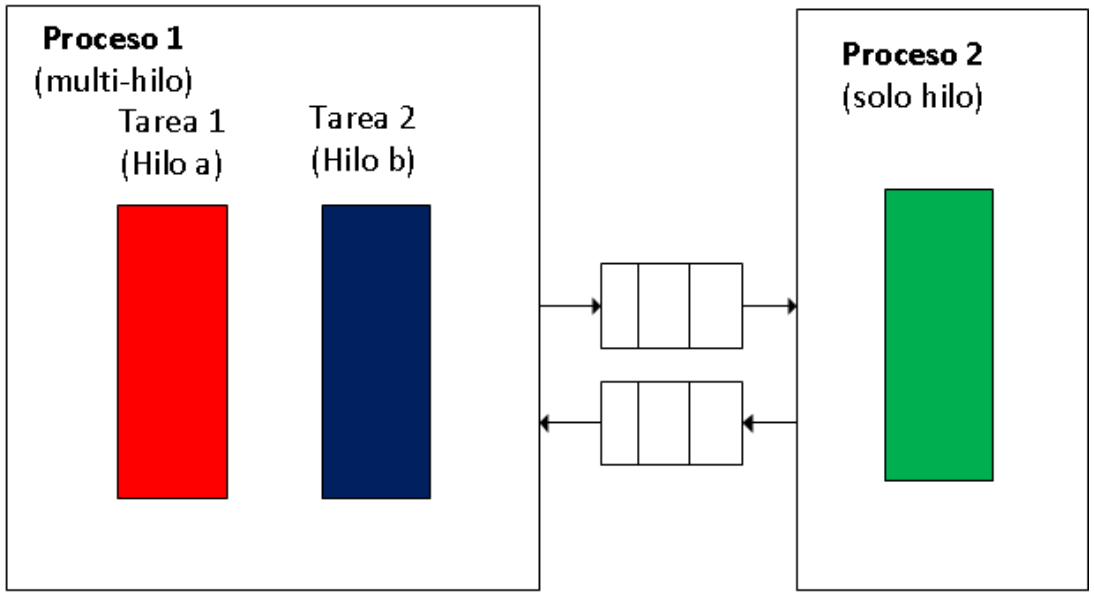
P: velocidad de ejecución

G. Amdahl, "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities", 1967

Programación concurrente en II

Entre procesos:

- Diferentes espacios de memoria:
- Cambio contexto más costoso
- Mecanismo de comunicación preferido colas de mensajes



Entre hilos:

- Único espacio memoria:
- Cambio contexto menos costoso
- Mecanismo de comunicación puede ser variables comunes

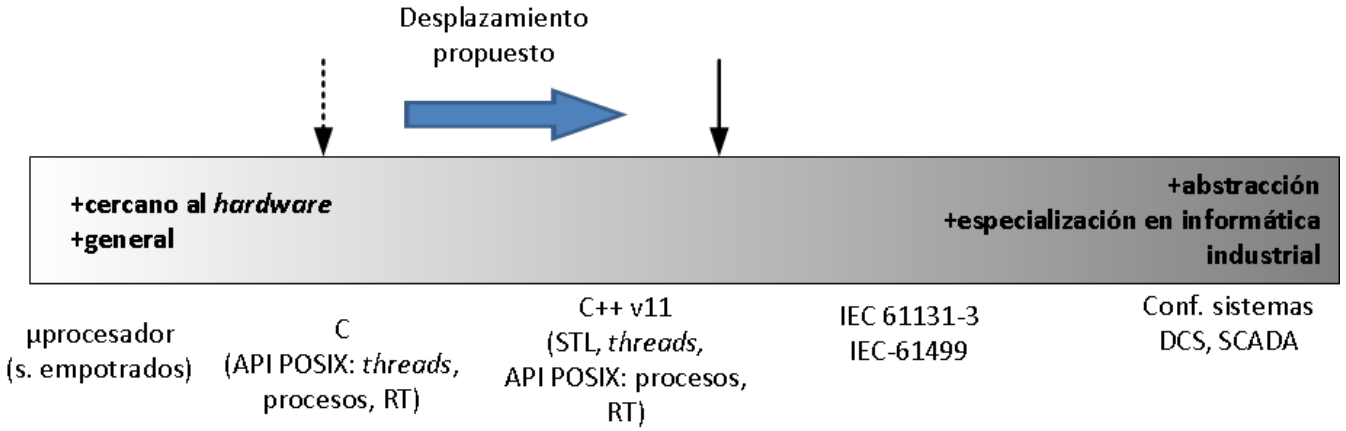
La **concurrencia** se implementa con soporte del **S.O.** y de los **lenguajes de programación**.

Lenguajes que soportan concurrencia:

- Ada, C#, Java, Erlang, Haskell, etc.
- Nativamente o a través de librerías.
- C++v11 soporta concurrencia <threads> y elementos de sincronización.

Elección del lenguaje C++

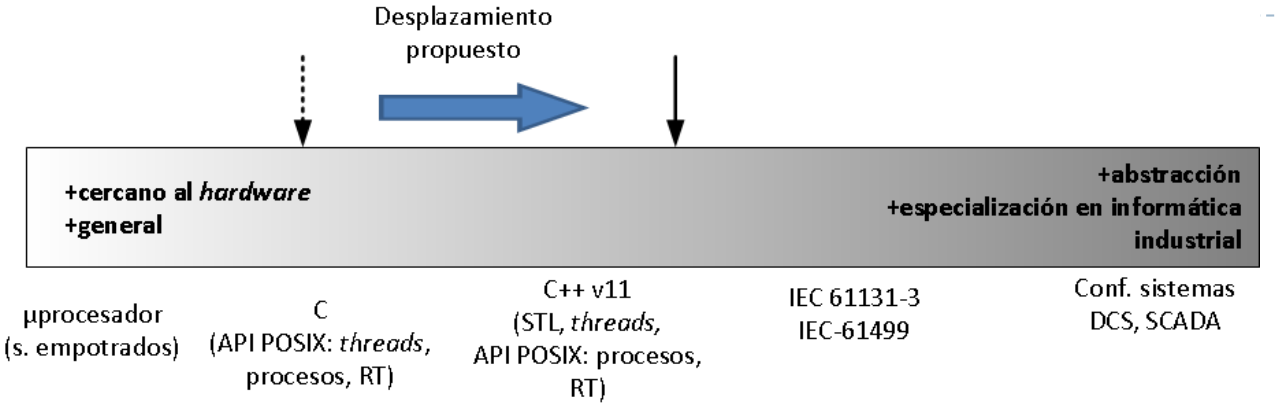
Petición de principio: El graduado de GIEIA debe ser capaz de programar aplicaciones reactivas con un **lenguaje general** aunque deben ser consciente del rango completo de posibilidades.



Situación previa:

- **Lenguaje C** (imperativo procedural)**.** Los alumnos lo han visto en **Fund. Informática**.
- Posibilidades de bajar a **nivel de hardware** (bueno para **sistemas empotrados**)
- Concurrencia con librerías externas dependientes del S.O. (ej: **POSIX**).
- Relativamente **bajo nivel de abstracción**. La creación de la infraestructura necesaria (ej: colecciones con reserva dinámica de memoria) requiere esfuerzo que sólo deja tiempo para **ejemplos muy simples**.

Elección del lenguaje C++



Lenguaje de elección C++:

- Lenguaje **multi-paradigma** (procedural, orientado a objeto, prog. genérica, funcional).
- Muy extendido con un desarrollo muy acelerado (versiones 11, 14 y 17) con mejoras importantes en pocos años.
- Librería extensa: en particular la librería genérica **STL** que permite trabajar con colecciones de objetos arbitrarios (vectores, mapas, listas, etc) a un **gran nivel de abstracción**.
- Con las prácticas adecuadas, el **coste de la abstracción** es mínimo (tiempo y huella en memoria).
- Desde C++v11: **C++ moderno**: nuevas primitivas (**bucles basados en rango**, inferencia de tipo: tipo auto, funciones anónimas, etc), mejoras en biblioteca estándar (*smart pointers*, trabajar con lapsos de tiempo, expresiones regulares, etc).
- Particularmente C++v11 **nuevo modelo de memoria** y **recursos de librería estándar** para **concurrencia** mediante hilos (threads).

Elección del lenguaje C++

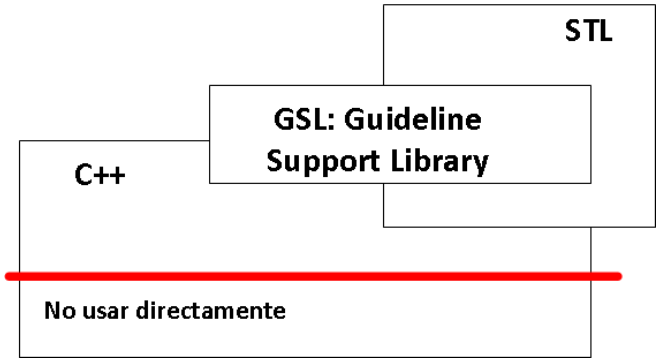
```
#include <iostream>
#include <thread>
#include <vector>

void ImprimeVec(std::vector<int> && v)
{
    for (auto elem: v )
        std::cout << elem << std::endl;
}

int main()
{
    std::vector<int> vec {1,5,7,8,9,10,11,45,

    std::thread t(ImprimeVec,std::move(vec));
    t.join();
    return 0;
}
```

Concurrencia en C++ v11 soporta*:

- **Hilos** *std::thread*.
- Objetos de **exclusión mutua** para accesos a datos compartidos (*std::mutex*).
- **Sincronización de hilos**: unión (*std::thread.join*), **espera por eventos** sin consumo de recursos (std::condition_variable).
- **Comunicación asíncrona** entre hilos (*std::promises* y *std::futures*)
- Funciones varias: adquiere varios *mutex* de forma segura (*std::lock*), interroga **grado de paralelismo** (*std::thread::harware_concurrency*) .
- Contiene operaciones atómicas para realizar programación libre de cerrojos (<*atomic*>).

A. Williams, "C++: Concurrency in action. Practical multithreading", 2012.

Elección del lenguaje C++



Fiabilidad de las aplicaciones en C++:

- Lenguajes de propósito general tiene mucha variabilidad.
- Es muy difícil garantizar seguridad en aplicaciones complejas, recurrente, aplicado al control de procesos críticos.
- Tienden a ser retro-compatible, lo que agrava el problema.
- Se proponen **buenas prácticas** para evitar errores:
  - Evitar el uso de **new** and **delete** en programas de aplicación y sustiuirlo por *smart pointers*
- Generalizar el patrón **RAII** (*resource allocation is initialization*) a todo tipo de circunstancias que implican adquisión, uso y liberación de un recurso (*mutex*)
- Esfuerzo por dictar **GSL**: guía de nuevas practicas que puedan ser garantizadas por compiladores.

CppCon 2015: Bjarne Stroustrup "Writing Good C++14"

Elección del lenguaje C++

```cpp
#include <iostream>
#include <thread>
#include <mutex>
#include <vector>
#include <chrono>

class Recurso
{
public:
    bool salir=false;

    void mult(float factor)
    {
        std::lock_guard<std::mutex> guard(m);
        for (auto &elem: vec )
            elem*=factor;
    }
    void imprime()
    {
        std::lock_guard<std::mutex> guard(m);
        for (auto elem: vec )
            std::cout << elem << " ";
        std::cout << std::endl;
    }
private:
    std::mutex m;
    std::vector<float> vec {1,5,7,8,9,10,11,45,-12};
};
```

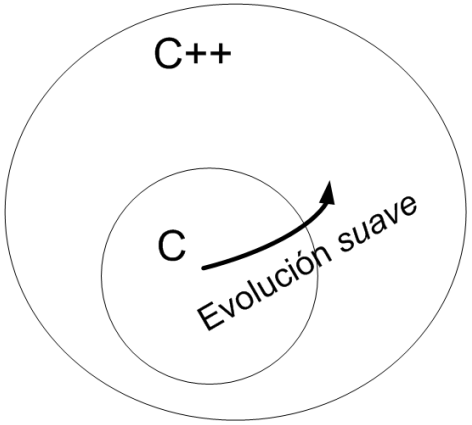
```cpp
void ImprimeVec(Recurso &r)
{
    while (!r.salir)
    {
        std::this_thread::sleep_for(std::chrono::seconds(5));
        r.imprime();
    }
}
int main()
{
    Recurso rec;
    std::thread t(ImprimeVec,std::ref(rec));
    do
    {
        float f;

        std::cout << "Entre factor (negativo salir)" << std::endl;
        std::cin >>  f;
        if (f>=0)
            rec.mult(f);
        else
            rec.salir=true;
    }
    while (!rec.salir);
    t.join();
    return 0;
}
```

Soporte del patrón RAII para trabajar con mutex (*std::lock_guard*).

Estrategia docente actual y futura

Estrategia seguida en asignatura II en el año previo:

- Laboratorio frente al ordenador (38 horas):
  - Primero utilizar las clases de la biblioteca (*string*, *cin*, *cout*, *vector,* etc).
  - **Nuevos tipos de datos**, **sobrecarga de funciones**, paso por referencia, etc.
  - Introducir elementos de creación de **nuevas clases**.
  - Fomentar en los alumnos la investigación y uso de las funcionalidades de la **biblioteca estándar**.
  - Realizar **proyectos** de mayor entidad y más cercanos al tema industrial.
- Clases de pizarra (22 horas):
  - Adelantar en el estudio teórico de los temas relevantes.



Planes para curso actual:

- Introducir los temas de **concurrencia** dentro del lenguaje C++ v11.
- Avanzar hacia el uso de un subconjunto fiable del lenguaje y **buenas prácticas** de programación.

Resultados

Resultados en la asignatura II:

- Se "pierde" tiempo en describir conceptos básicos del C++. Se espera que la situación mejore si de da un subconjunto del lenguaje en C++ en **Fund. Informática**.
- Los alumnos no terminan dominando el lenguaje C++, tampoco era la intención.
- La respuesta del alumno es muy positiva.
  - La experiencia de **buscar en la biblioteca estándar** por soluciones ya existentes les resulta muy satisfactoria.
  - Agradecen el uso de **ejemplos más realistas** y no puramente académicos.

Posibilidades de generalización:

- El uso del C++ puede resultar **polémico**. En cualquier caso, la experiencia descrita es generalizable para alumnos de titulaciones similares.
- Para alumnos de otras titulaciones (menos predispuestos a la programación) probablemente se requiera un **nivel de abstracción mayor** (ej: configuración de DCS y SCADAS)

Práctica entregable curso 2015-2016