

datsi

UPM

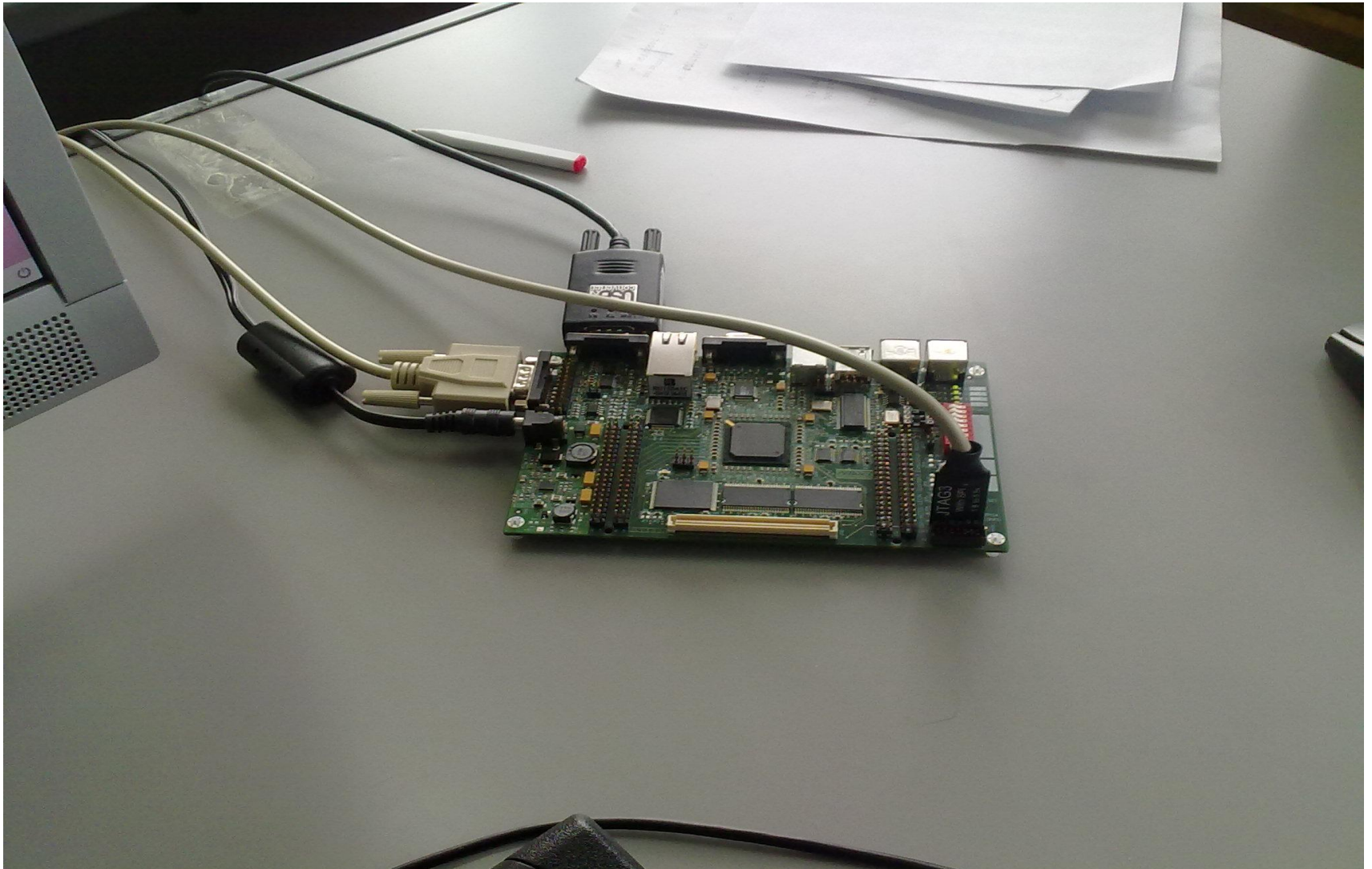
Desarrollo de sistemas empotrados

Juan Zamorano

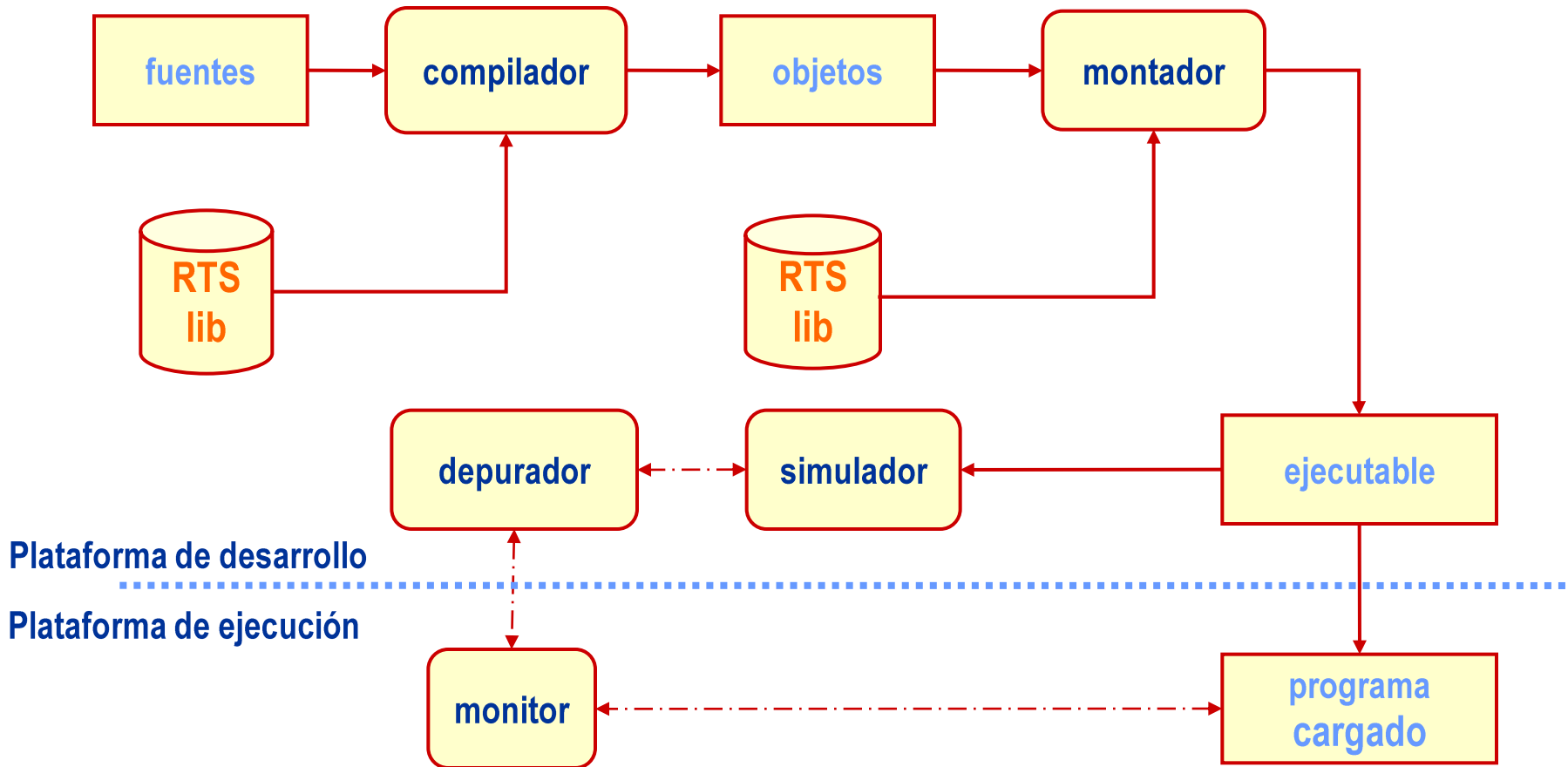
Motivación

- Los métodos, las herramientas y la tecnología que se usan para construir otros tipos de sistemas no sirven en general para los sistemas empotrados
 - no son suficientemente fiables
 - sólo contemplan el tiempo de respuesta medio, no el peor
 - no garantizan los requisitos temporales
- Las plataformas de desarrollo y ejecución suelen ser diferentes
 - es difícil hacer pruebas en la plataforma de ejecución
 - es difícil medir los tiempos con precisión

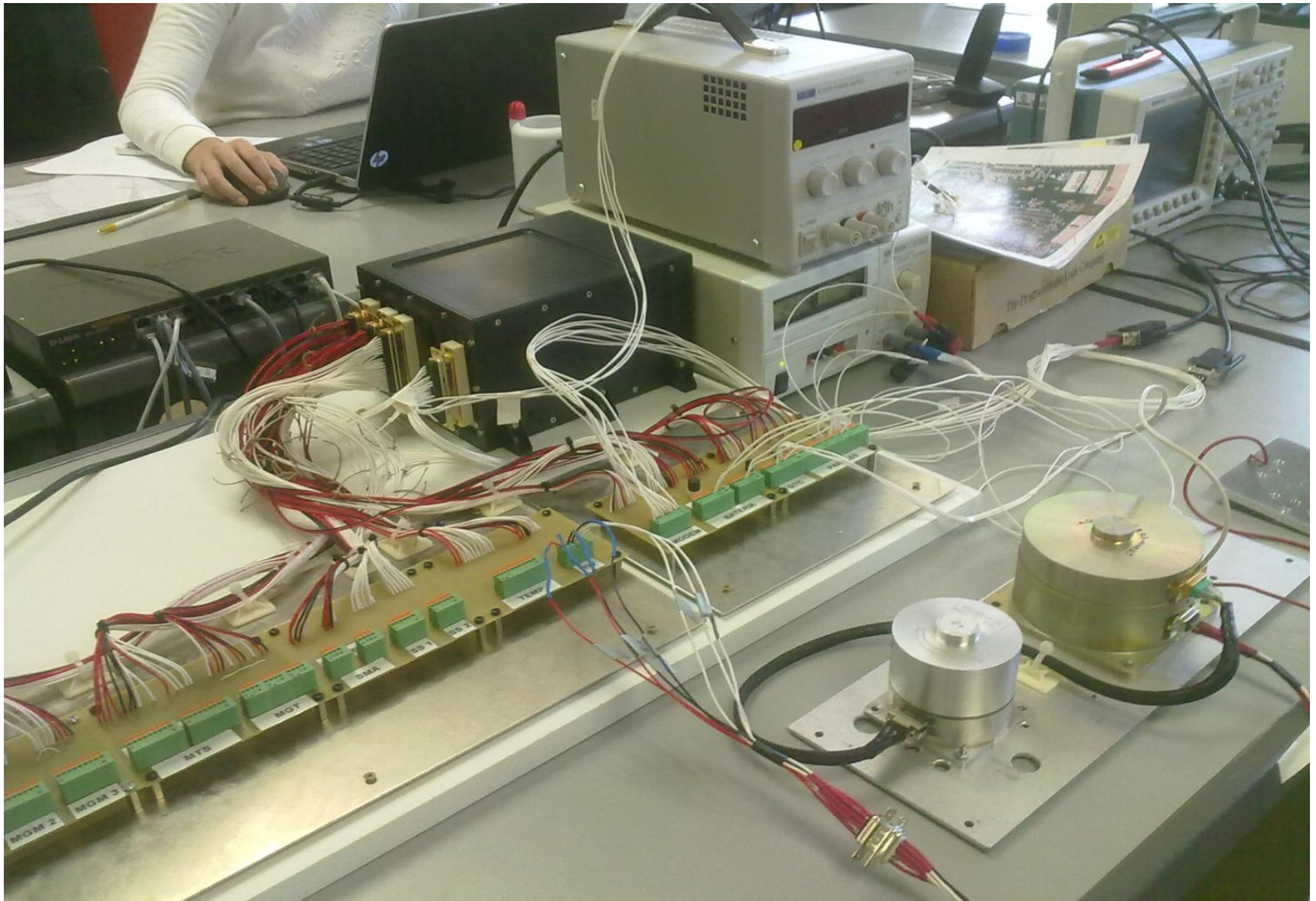
Desarrollo cruzado



Desarrollo cruzado



Desarrollo cruzado



Lenguajes de programación

- Un lenguaje de programación de sistemas empotrados debe facilitar la realización de sistemas
 - concurrentes,
 - fiables,
 - con un comportamiento temporal analizable
- Hay varias clases de lenguajes de interés para SSEE:
 - **Lenguajes ensambladores**
 - flexibles y eficientes, pero costosos y poco fiables
 - **Lenguajes secuenciales** (Fortran, C, C++)
 - necesitan un SO para concurrencia y tiempo real
 - **Lenguajes concurrentes** (Ada, Java, ...)
 - concurrencia y tiempo real incluidos en el lenguaje
 - **Lenguajes de descripción de hardware** (VHDL, Verilog, SystemC, ...)
 - incluyen además propagación de señales y simulación

C

- Es un lenguaje muy utilizado para programación de sistemas
- Es un lenguaje
 - estructurado, con bloques
 - sin tipado fuerte
 - muy flexible (pero a veces poco seguro)
- No tiene integrada la concurrencia ni el tiempo real
 - se consigue invocando servicios del sistema operativo de forma explícita
- No facilita la descomposición en módulos ni la programación con objetos
 - se puede hacer con C++
 - extensión de C para programar con objetos
 - no se suele usar en SSEE críticos por problemas de fiabilidad

Ejemplo: tarea periódica

```
void periodic () {
    struct timespec next, period;

    if (clock_gettime (CLOCK_MONOTONIC, &next) != 0) error();
    period.tv_sec = 0;
    period.tv_nsec = 10.0E6; /* 10 ms */

    while (1) {
        if (clock_nanosleep (CLOCK_MONOTONIC, TIMER_ABSTIME,
            &next, 0) != 0) error();
        acción periódica
        next = next + period;
    }
}
```


Ada

- Es un lenguaje diseñado específicamente para sistemas de tiempo real empotrados
 - concurrencia
 - tiempo real
 - acceso al hardware e interrupciones
- Es un lenguaje descendiente de Pascal
 - estructura en bloques
 - fuertemente tipado
- Está pensado para construir sistemas grandes y cambiantes
 - paquetes (módulos) y esquemas genéricos
 - extensión de tipos con herencia
 - biblioteca jerárquica
 - interfaces normalizadas con otros lenguajes (C, Fortran)

Ada 2005

- La versión actual de Ada es Ada 2012
 - La versión Ada 2005 mejoró el soporte para sistemas empujados.
- La norma define
 - un **núcleo** común para todas las implementaciones (*core language*)
 - unos **anexos** especializados para
 - programación de sistemas
 - sistemas de tiempo real
 - sistemas de alta integridad
 - sistemas distribuidos
 - sistemas de información
 - cálculo numérico
 - Los anexos definen
 - paquetes de biblioteca
 - mecanismos de implementación
- **No añaden sintaxis ni vocabulario al lenguaje**

Ejemplo: tarea periódica

```
use Ada.Real_Time;
task body Periodic is
  Period      : constant Time_Span := Milliseconds(10);
  Next_Time   : Time := Clock;
begin
  -- iniciación
  loop
    delay until Next_Time;
    acción periódica
    Next_Time := Next_Time + Period;
  end loop;
end Periodic;
```

Novedades en Ada 2005

- Mejor soporte para sistemas de tiempo real
 - perfil de Ravenscar
 - relojes y temporizadores de tiempo de ejecución
 - nuevos métodos de planificación del procesador
- Mejoras en la programación mediante objetos
 - interfaces
- Mejoras en la estructura de los programas y en las reglas de visibilidad de las declaraciones
- Mejoras en la biblioteca estándar
- Otras mejoras en el lenguaje en Ada 2012
 - Soporte para multicore
 - Programación basada en contratos

Perfiles para sistemas críticos

- El documento *Guide for the use of the Ada programming language in high-integrity systems* define subconjuntos seguros de Ada para aplicaciones críticas
- **SPARK** es un lenguaje que permite el uso de técnicas de análisis estático
 - subconjunto de Ada + anotaciones
- El perfil de **Ravenscar** define un subconjunto seguro de la parte concurrente de Ada, y los correspondientes servicios de sistema operativo

Java

- Es un lenguaje pensado para construir sistemas distribuidos
 - basado en objetos dinámicos
 - con concurrencia integrada en el lenguaje
 - bibliotecas de clases (APIs) muy útiles
 - pensado para que el código objeto sea portátil
 - interpretado por una *máquina virtual (JVM)*
 - *“write once, run everywhere”*
- La definición original no es adecuada para sistemas empotrados de tiempo real
 - la planificación de actividades concurrentes no está bien definida
 - los mecanismos de sincronización son inadecuados
 - la gestión dinámica de memoria introduce indeterminismo
 - la medida del tiempo no es suficientemente precisa
 - otros problemas con excepciones y concurrencia

Java para tiempo real

- *Real-Time Specification for Java (RTSJ)*
 - basada en un máquina virtual extendida para STR
 - hay una implementación de referencia (*TimeSys*)
 - y otras comerciales (por ejemplo, *Jamaica*)
 - investigación: Java para sistemas de alta integridad (HIJA)
- Los compiladores y las máquinas virtuales para Java de tiempo real no están todavía completamente maduros
 - lo más complicado es la gestión de memoria y la recogida de basura

Ejemplo: tarea periódica

```
public class Periodic extends RealTimeThread {  
  public Periodic() {  
    super();  
    setReleaseParameters (  
      new PeriodicParameters (  
        new AbsoluteTime ( 0,0),      /* start */  
        new RelativeTime (10,0),    /* period */  
        ...)  
      );  
    }  
  
  public void run() {  
    while (true) {  
      actividad periódica  
      waitForNextPeriod ();  
    }  
  }  
}
```


Lenguajes síncronos

- Se basan en un modelo matemático sencillo
 - los sucesos son instantáneos
 - las acciones también
 - puede haber sucesos y reacciones simultáneos
 - se puede efectuar un análisis formal del comportamiento temporal
- Ejemplos
 - Esterel
 - Lustre, Signal
 - Statecharts
- Se compilan a autómatas realizados en lenguajes de programación secuenciales (C, Ada sin concurrencia)

Ejemplo en Esterel

```
module periodic;  
  input Millisecond;  
  every 10 Millisecond do  
    acción periodica  
  end every;  
end module;
```

Lenguajes de descripción de hardware

- Permiten describir circuitos electrónicos y lógica digital
 - describen tanto la operación como su diseño y organización
 - permiten simular el circuito descrito
 - incluyen concurrencia y tiempo real
 - Incluyen propagación de señales eléctricas
- Ejemplos
 - VHDL sintaxis similar a Ada
 - Verilog sintaxis similar a C
 - SystemC sintaxis similar a C++
- Tienen simuladores incluidos para comprobar los circuitos descritos

Ejemplo en VHDL

```
process Periodic is  
  constant Period      : Time := 10 ms;  
begin  
  
  acción periódica  
  
  wait for Period;  
end process Periodic;
```

Sistemas operativos

- Los sistemas operativos convencionales no son adecuados para realizar sistemas empotrados de tiempo real
 - no tienen un comportamiento determinista
 - no permiten garantizar los tiempos de respuesta
 - algunos de ellos son poco fiables
- Un sistema operativo de tiempo real (SOTR) debe soportar
 - concurrencia: procesos ligeros (hebras o *threads*)
 - temporización: medida de tiempos y ejecución periódica
 - planificación determinista: gestión del procesador y otros recursos
 - dispositivos de E/S: acceso a recursos de hardware e interrupciones

POSIX

- Es un conjunto de normas IEEE/ISO que definen interfaces de sistemas operativos
- Permiten desarrollar software portátil y reutilizable (*Portable Operating System Interface*)
- Las normas definen servicios que se pueden incluir o no en un sistema operativo particular
- Además se definen *perfiles de aplicación* con conjuntos de servicios estándar
- Hay interfaces para C, Ada, y otros lenguajes

Normas POSIX

- **IEEE 1003.1 (2004)**
 - interfaz básica para C similar a UNIX™
 - incluye extensiones de tiempo real y hebras (*threads*)
- **IEEE 1003.5 (1999)**
 - interfaces para Ada
- **IEEE 1003.13 (1998)**
 - Perfiles para sistemas de tiempo real

POSIX para sistemas de tiempo real

Servicios de tiempo real

- Relojes precisos y temporizadores
- Señales de tiempo real
- Planificación por prioridades

Servicios de hebras (*threads*)

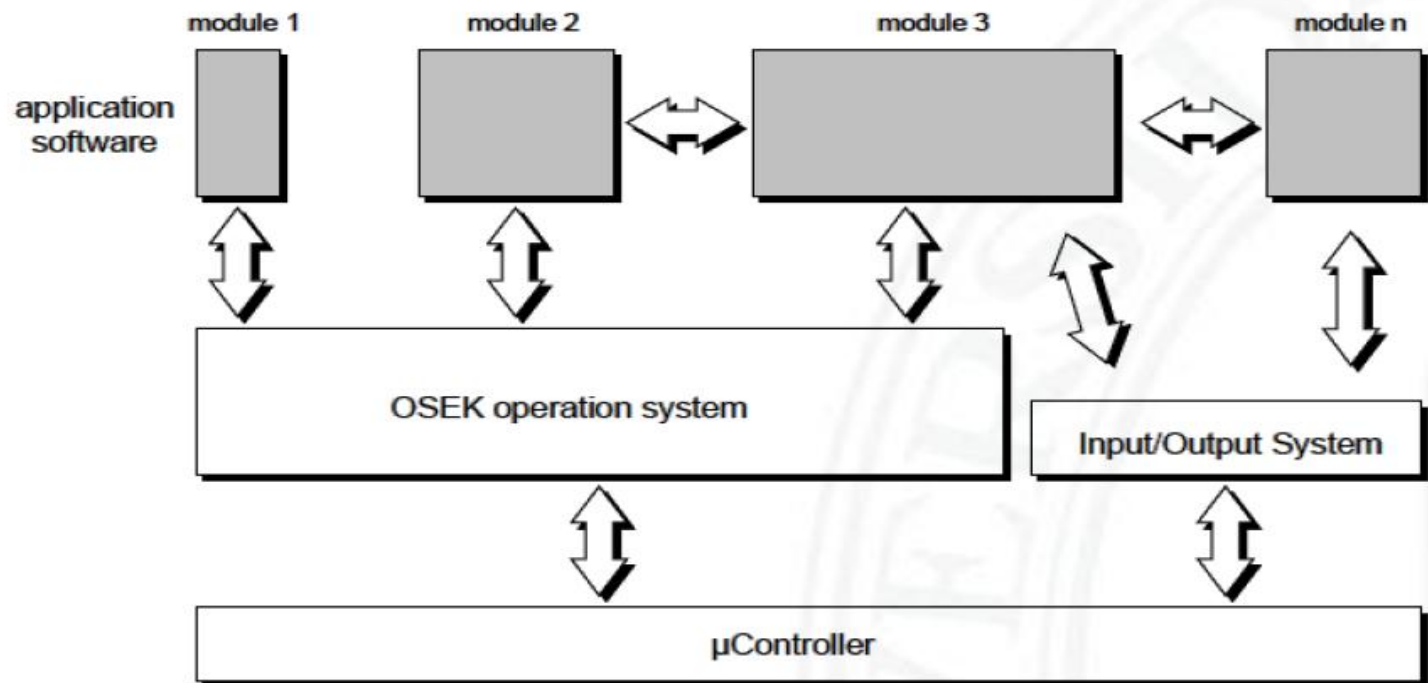
- Hebras
- *Mutex* y variables de condición

Perfiles de aplicación

- Definen subconjuntos de servicios para distintos tipos de aplicaciones
- **POSIX 13** : Perfiles para sistemas de tiempo real
 - **PSE50 : sistema de tiempo real mínimo**
 - sin gestión de memoria, ficheros ni terminal
 - sólo *threads* (no procesos pesados)
 - **PSE51 : controlador de tiempo real**
 - tiene sistema de ficheros y terminal
 - **PSE52 : sistema de tiempo real dedicado**
 - tiene gestión de memoria y procesos pesados
 - **PSE53 : sistema de tiempo real generalizado**
 - sistema completo con todo tipo de servicios

Otros estándares: OSEK

- OSEK (parcialmente estandarizado ISO 17356)
 - Offene Systeme und deren Schnittstellen für die Elektronik in Kraftfahrzeugen: Sistema Abierto y sus Interfaces para Electrónica en Vehículos a motor.
 - Similar a POSIX 13 PSE50



Otros estándares: ARINC 653

- ARINC 653 (Avionics Application Standard Software Interface)
 - estándar para IMA (Integrated Modular Avionics)
 - ejecución de distintas aplicaciones en un único computador
 - proporciona segregación temporal y espacial

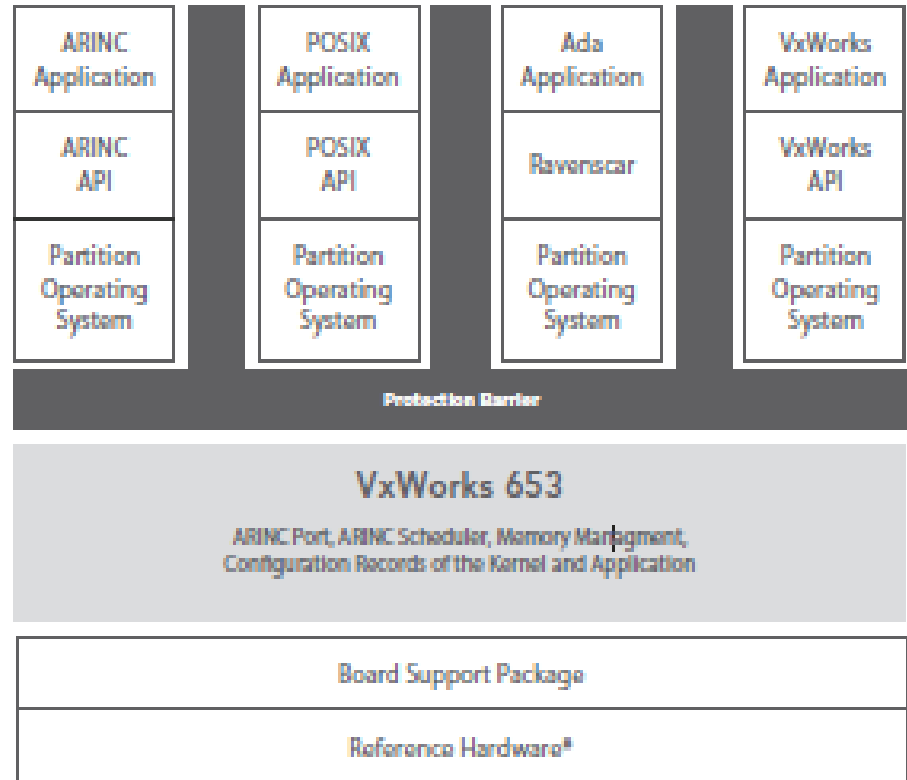


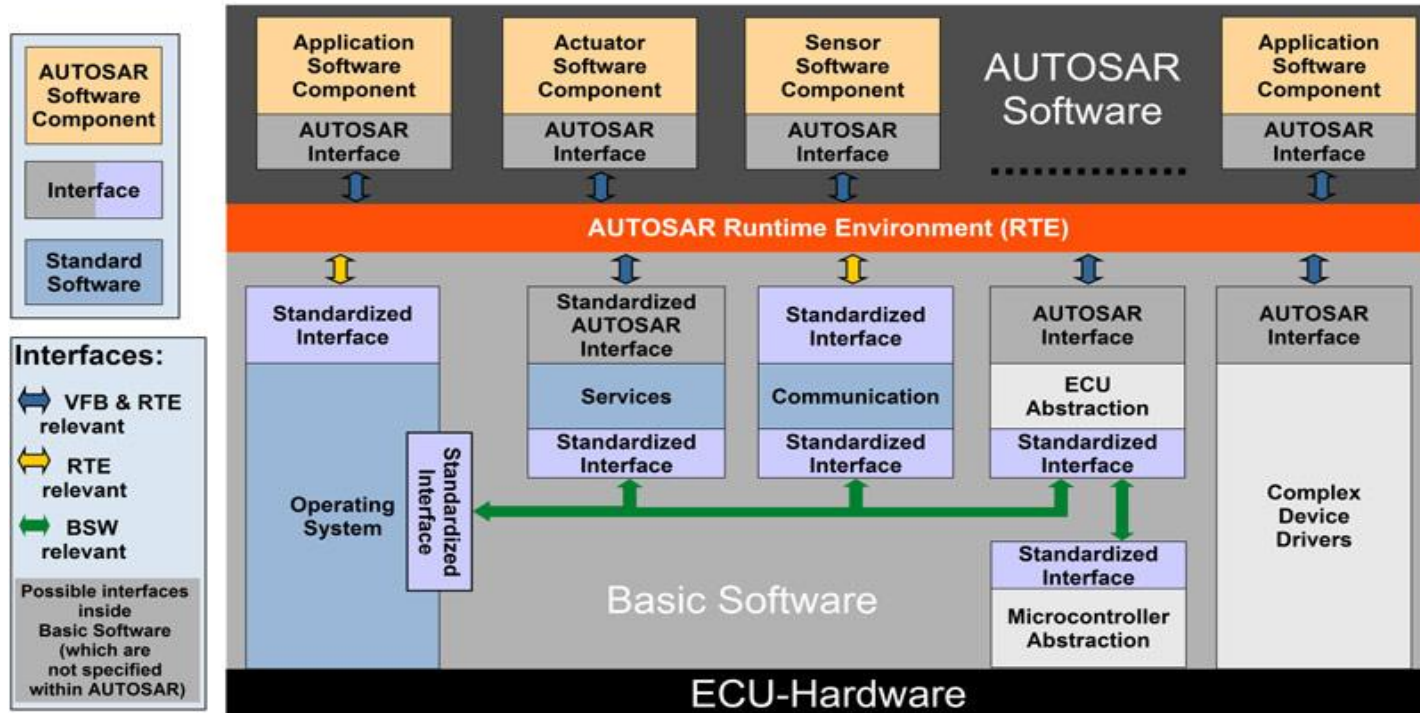
Figure 2: IMA design with VxWorks 653 Platform

* Optional

Otros estándares: AUTOSAR

- AUTOSAR (AUTomotive Open System ARchitecture)
 - Bus software: abstracción de los servicios del sistema y del hardware

Interfaces
Components and interfaces view (simplified)



Ejemplos de SOTR

- LynxOS, pSOS, QNX, ...
- VxWorks
- RTEMS
- TOPPERS/OSEK
 - nxtOSEK
 - para Lego Mindstorms (ARM7)
- **RT-Linux**
- **XtratuM** — Universidad Politécnica de Valencia
 - Hipervisor para procesadores SPARC, ARM y PCx86
- **MaRTE OS** — Universidad de Cantabria
 - perfil POSIX PSE50
 - para sistemas empotrados PCx86
- **Open Ravenscar Kernel (ORK)** — UPM
 - núcleo de SOTR para el lenguaje Ada y procesadores SPARC y PCx86

Diseño de sistemas empotrados

- El diseño de un sistema tiene varios aspectos
 - **funcional**: relación entre valores de entrada y de salida
 - **concurrente**: actividades concurrentes, sincronización, comunicación
 - **temporal**: requisitos temporales
 - **arquitectónico**: componentes, relaciones entre ellos
- Cada aspecto se expresa mejor con un tipo de notación.
Por ejemplo:
 - **Simulink** para el aspecto funcional
 - **UML** (con perfiles específicos) para el diseño detallado de componentes
 - **AADL** (*Analysable Architecture Description Language*) para los aspectos de concurrencia y arquitectura

Ejemplo: Simulink

The screenshot displays the MATLAB Simulink environment. The workspace window shows a list of variables:

Name	Size	Bytes	Class
LWall	1x1	8	double array
LWindow	1x1	8	double array
M	1x1	8	double array
Mdot	1x1	8	double array
RWall	1x1	8	double array
RWindow	1x1	8	double array
Req	1x1	8	double array
THeater	1x1	8	double array
TinIC	1x1	8	double array
c	1x1	8	double array
cost	1x1	8	double array
densAir	1x1	8	double array
ha	1x1	8	double array
htHouse	1x1	8	double array
htWindows	1x1	8	double array

The Command Window shows the command `>> thermo` and `>>`.

The Simulink model, titled "thermo", illustrates a house heating system. It starts with a "Set Point" of 70 Fahrenheit, which is converted to Celsius (F2C) and compared with the "Avg Outdoor Temp" (50 Fahrenheit, converted to Celsius) and "Daily Temp Variation". The resulting error signal (T_err) is processed by a "Thermostat" block, which outputs a "blower cmd" to a "Heater Blower" block. The blower command is multiplied by a gain of 1/s to produce "heater QDot". This heat input, along with heat loss from walls and windows (represented by LWall, LWindow, RWall, RWindow), is used to calculate the indoor temperature (Tin). The indoor temperature is converted back to Fahrenheit (C2F) and compared with the set point to determine the "Heat Cost (\$)". The heat cost is then multiplied by a gain of 1/s to produce the "Dollar Gain". The indoor temperature is also plotted in the "Thermo Plots" window.

The "Thermo Plots" window displays two graphs:

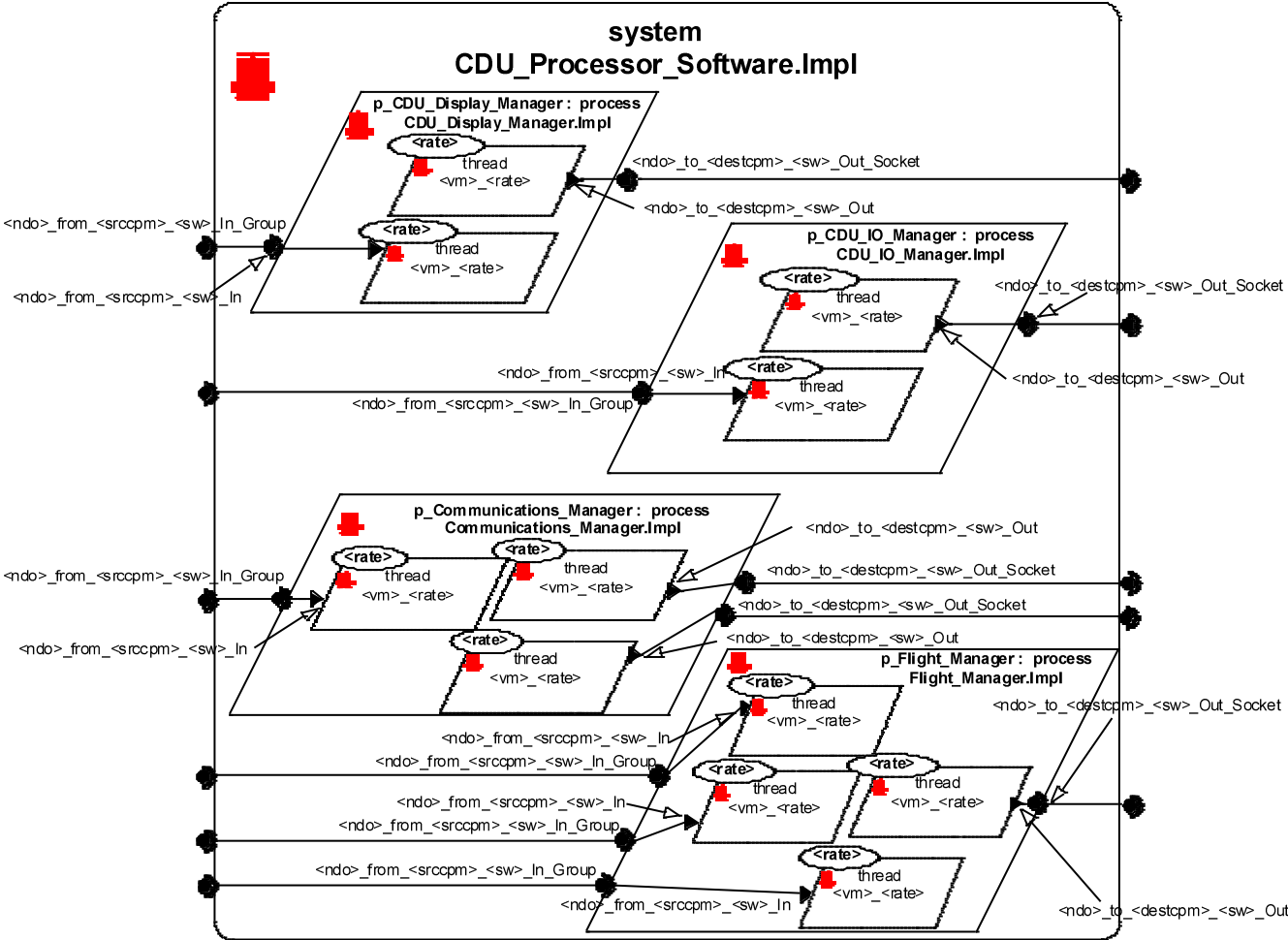
- Indoor vs. Outdoor Temp.:** A plot showing the indoor temperature (yellow line) and outdoor temperature (purple line) over time. The outdoor temperature oscillates between approximately 30 and 70 degrees Fahrenheit, while the indoor temperature remains relatively stable around 70 degrees Fahrenheit.
- Heat Cost (\$):** A plot showing the heat cost over time. The cost starts at 0 and increases to approximately 30 x 10^4 dollars by the end of the simulation.

Instructions for running the simulation are provided at the bottom of the Simulink window:

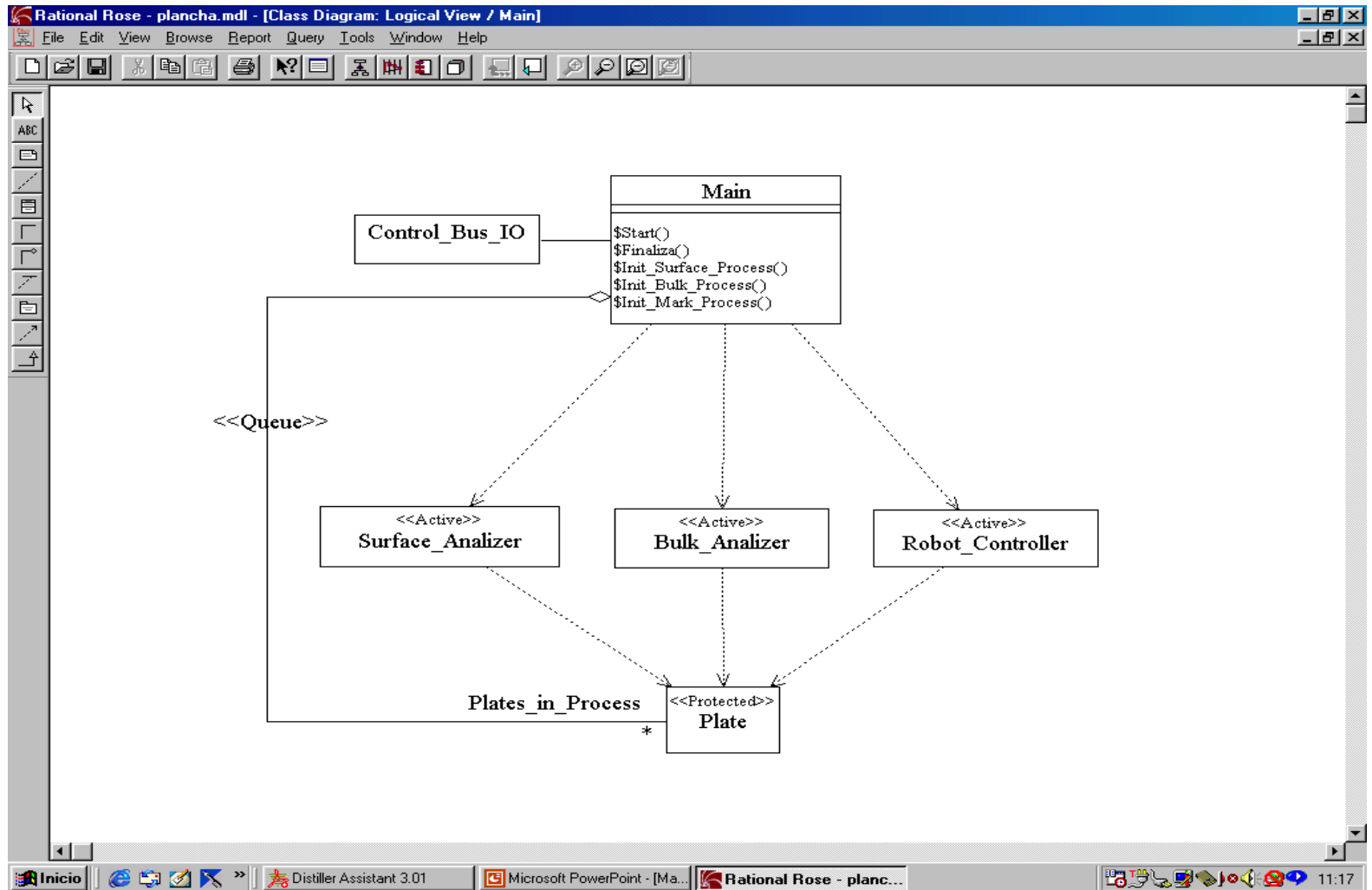
House Thermodynamics (Double click on the "?" for more info)

To start and stop the simulation, use the "Start" selection in the "Simulation" pull-down menu

Ejemplo: AADL

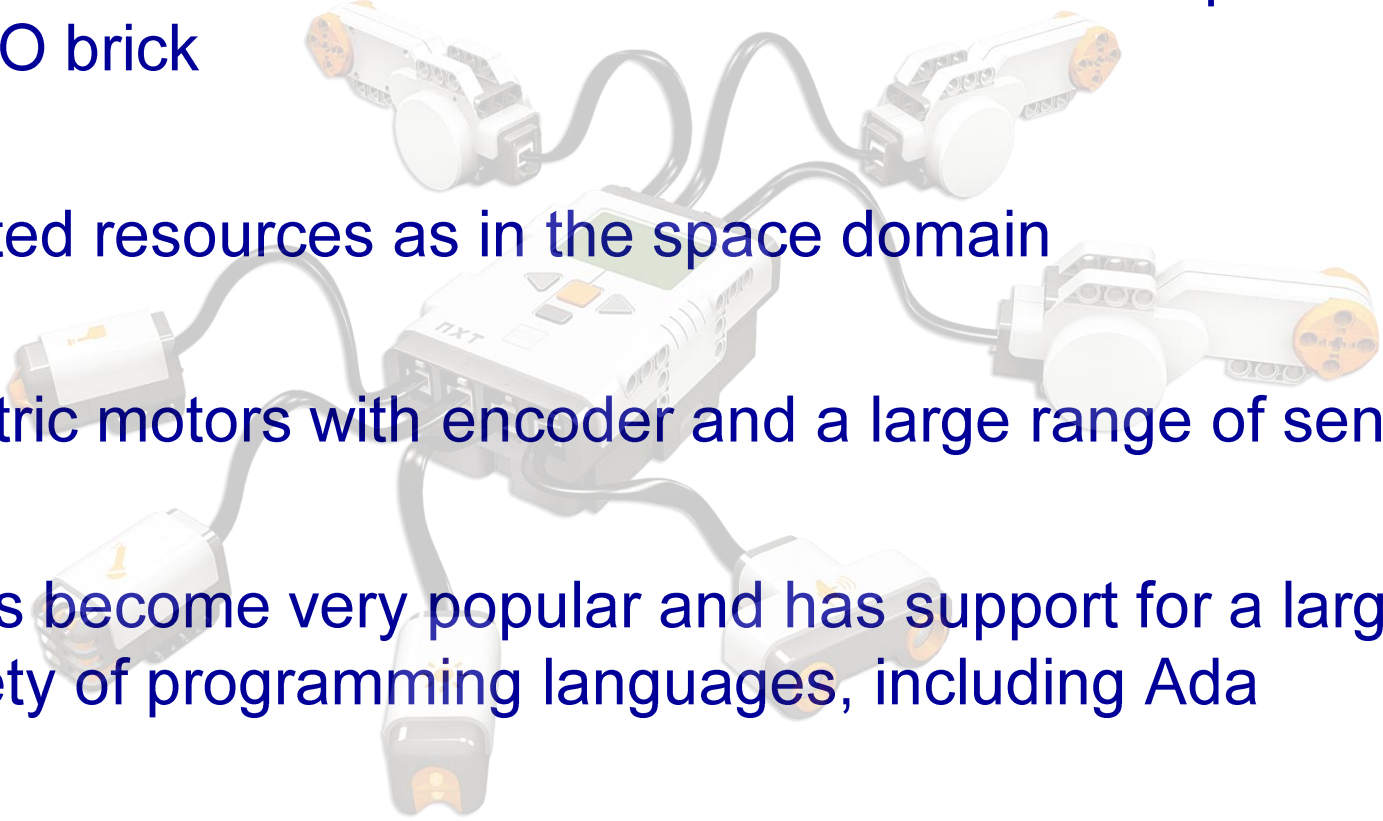


Ejemplo: UML

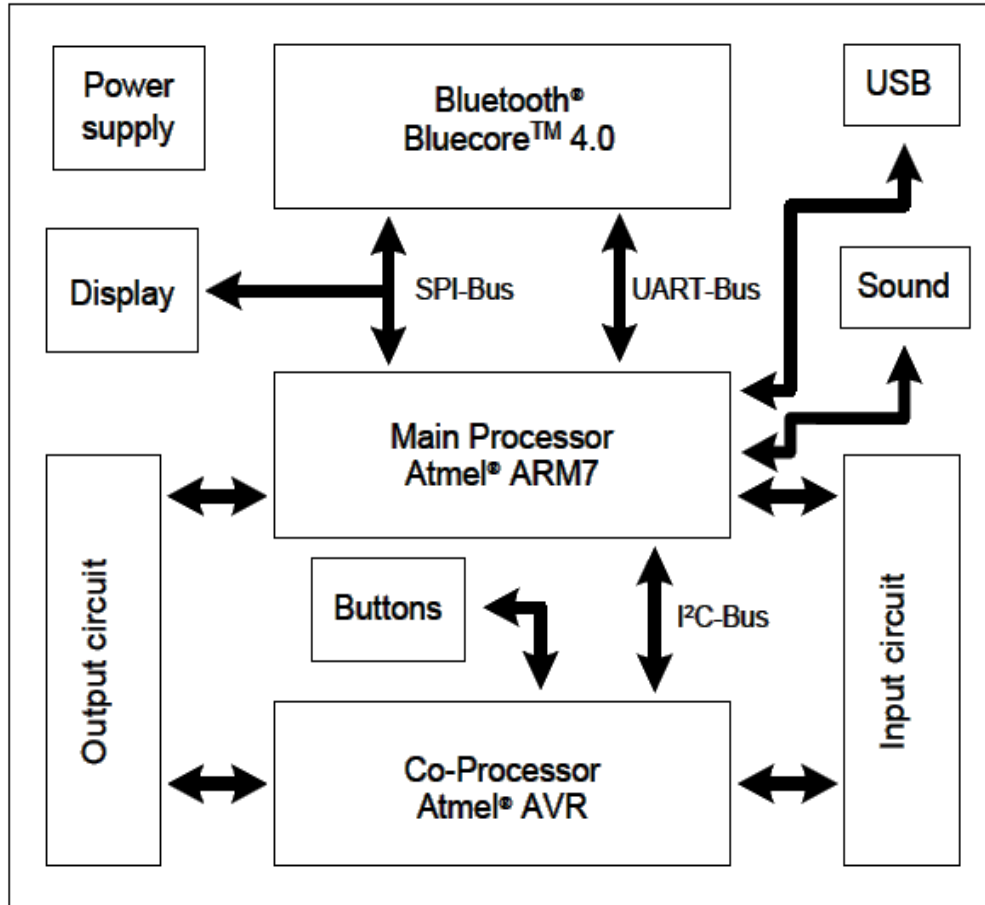


Example: LEGO Mindstorms

- Economic device that offers an embedded computer: the LEGO brick
- Limited resources as in the space domain
- Electric motors with encoder and a large range of sensors
- It has become very popular and has support for a large variety of programming languages, including Ada



LEGO brick



- 32-bit processor Atmel ARM7 at 48 Mhz
- 64 KB RAM and 256 KB Flash memory
- Atmel AVR co-processor for interacting with sensors and motors

The GNAT GPL for LEGO MINDSTORMS NXT compilation system

- Ada compiler & linker
 - Full support of Annex D – Real-time systems
- Real-time kernel
 - Implements the Ravenscar tasking model
- Ada run-time system
 - Implements the Ada tasking model on top of the kernel

GNAT for LEGO MINDSTORMS NXT

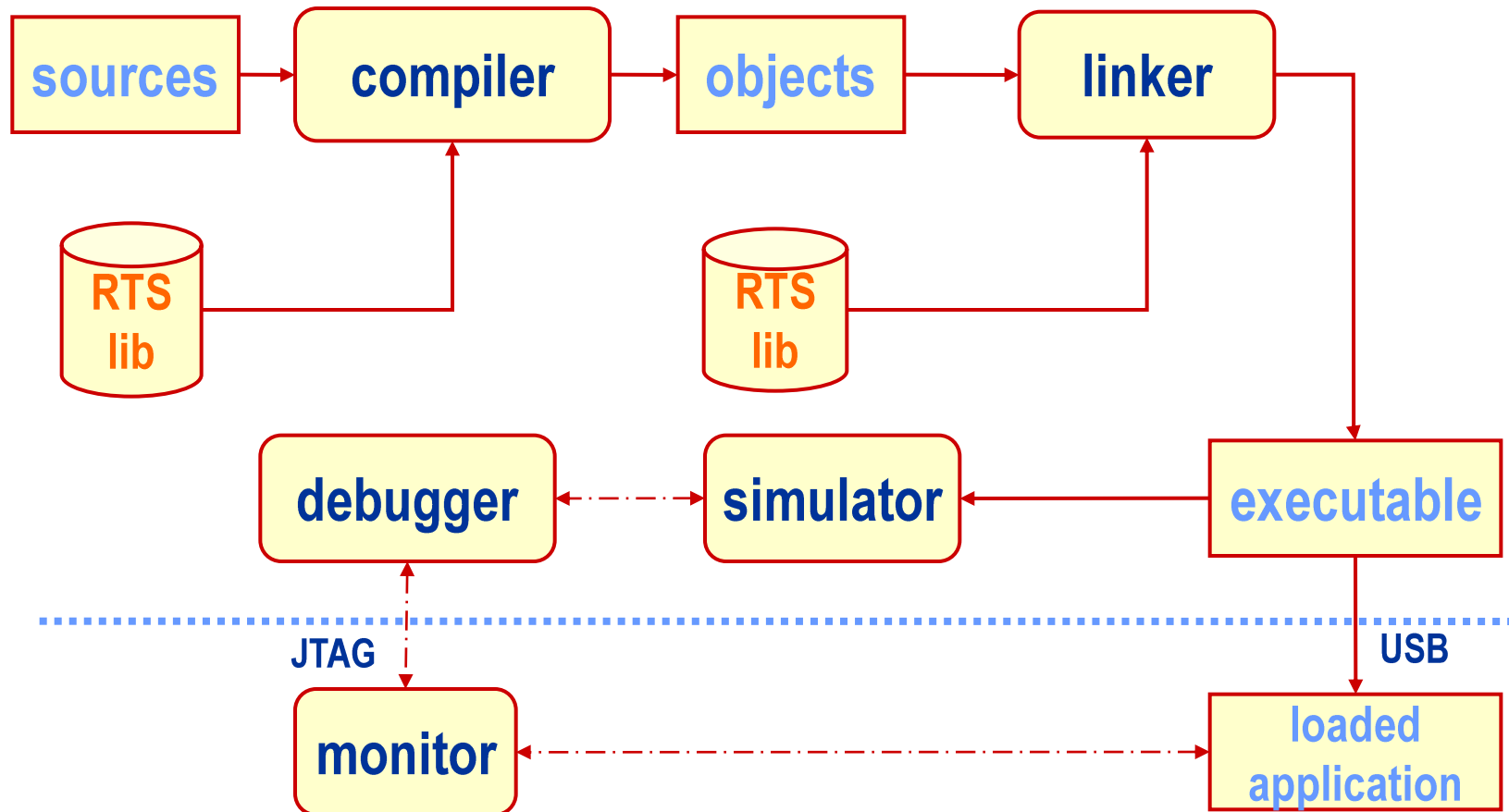
- Cross-compilation system targeted to LEGO MINDSTORMS computers
 - ARM7 processor
 - Includes drivers for MINDSTORMS devices
- Components
 - GNAT Ada compiler (Ada Core)
 - GNARL run-time system (Ada Core)
 - Adapted version of the ORK kernel (UPM)

NXT Ada drivers

- Developed by AdaCore
 - Completely coded in Ada
 - Major updates do not provide backward compatibility
- API not documented
 - read the code at
(`$installation_dir`)/lib/gcc/arm-eabi/4.5.3/rts-ravenscar-sfp/drivers
- The `NXT.AVR` package must be withed in every NXT program
 - It includes a periodic task with the highest priority that handles co-processor communications every 20 ms
 - `NTX.Display` (or `NTX.Display.Concurrent`) and `NXT.Last_Chance` are also recommended
 - High-level access to motors, sensors and buttons are provided by: `NXT.Motors`, `NXT.I2C_Sensors` and `NXT.Filtering`

Cross-development

Development platform: Linux



Execution platform: LEGO brick

Development tools

- Cross compilation system based on gnu tools
 - Ada compiler, assembler, linker, etc...
- Drivers to interact with LEGO Mindstorms hardware
- Custom firmware for LEGO Mindstorms
- Communication tool between the target and the host to download firmware and executable files
- Debugging with GDB through JTAG hardware for LEGO Mindstorms
- WCET tool to procure a timing analysis of the system
- Real-time modelling tool to evaluate the feasibility of the system

A simple example

- The directory that contains the binary tools must be in the PATH
 - `/usr/local/gnatmindstorms2011/bin/`
- Examples with Makefiles are provided with distribution
 - `/usr/local/gnatmindstorms2011/share/examples/mindstorms`
- The `fwexec` application, from the LibNXT library, can be used to download images to the LEGO brick
 - The LEGO brick must be in SAM-BA mode
- The proper image to be downloaded is the binary one (`hello.bin`)
 - `sudo ./fwexec hello.bin`
 - `sudo chmod -R 777 /dev/bus/usb`
 - Or include in `/etc/udev/rules.d/`
 - `fwexec hello.bin`
- The program starts automatically after downloading

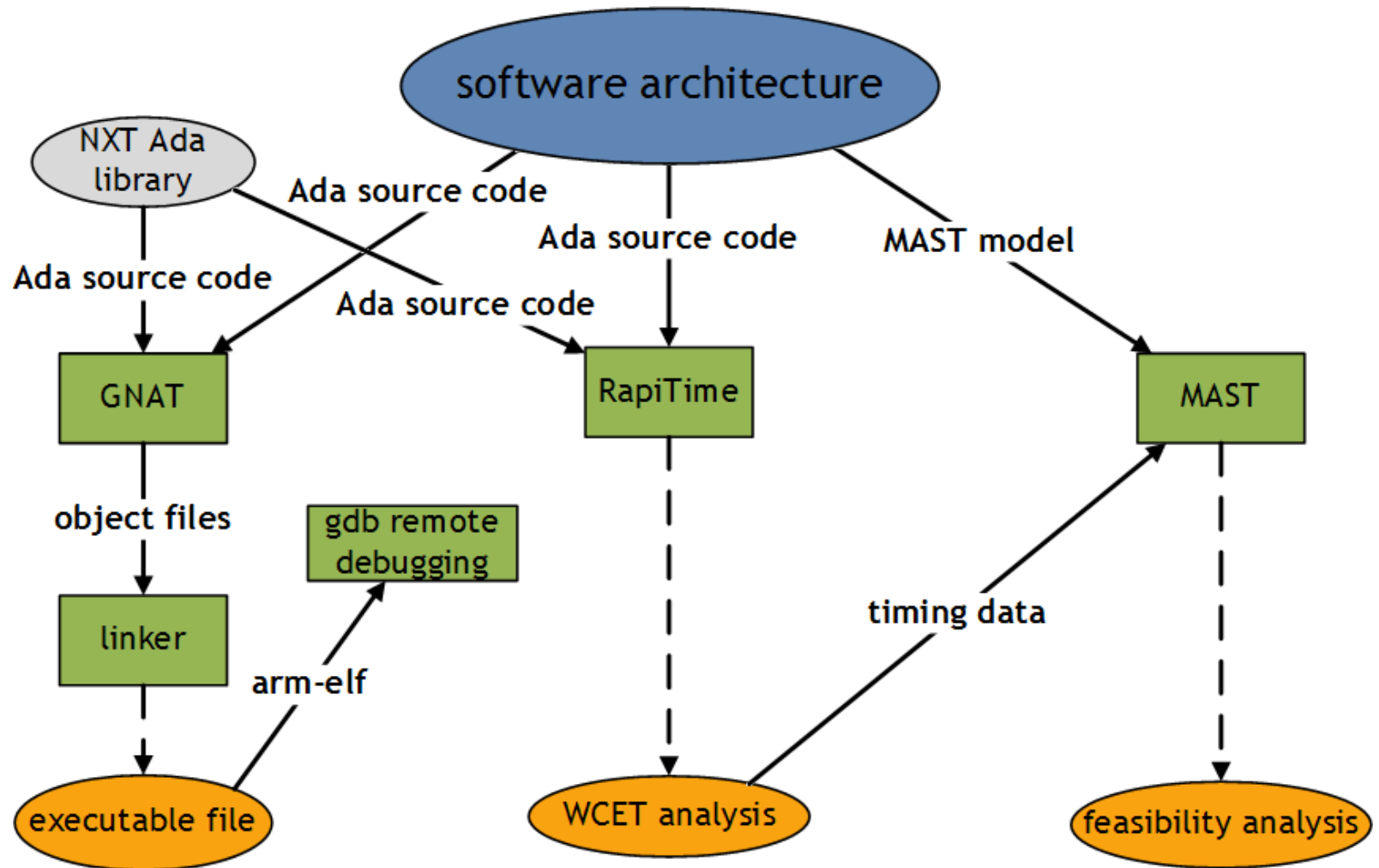
Code of simple example

The screenshot shows an IDE window with the following components:

- Menu Bar:** File, Edit, Navigate, VCS, Project, Build, Debug, Tools, Window, Help.
- Scenario Panel:** The project contains no scenario variables.
- Project Browser:** Shows a project named 'Hello' with sub-items: hello.adb, tasking.adb (expanded to show entry, package, pragma, protected, subprogram, task, type), and tasking.ads.
- Code Editor:** Displays the following Ada code:

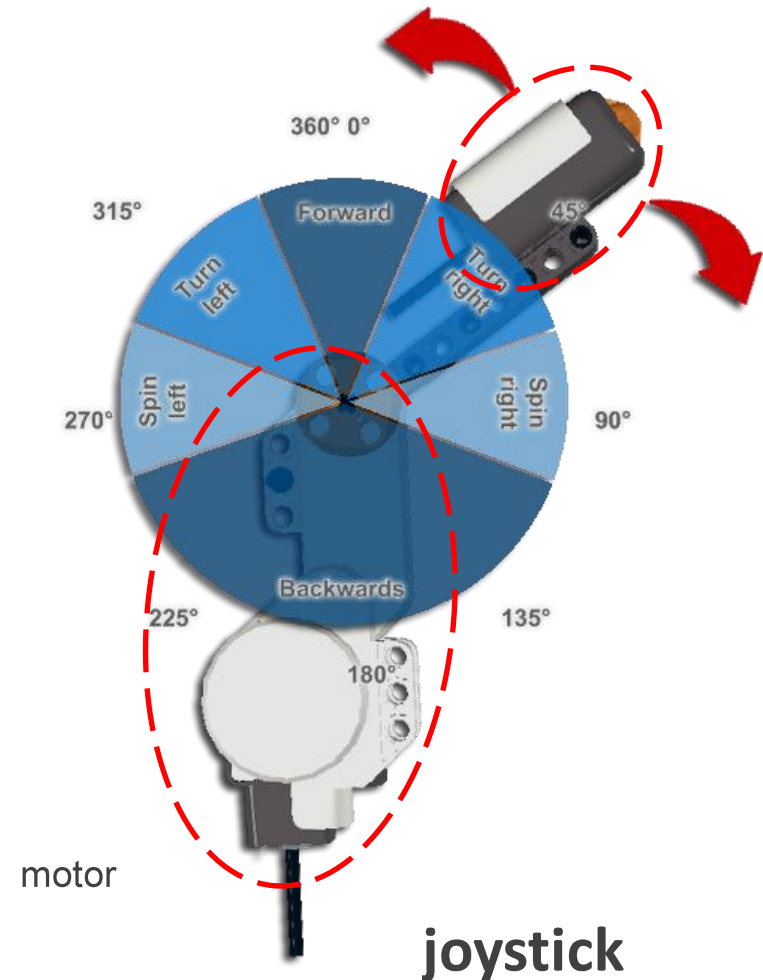
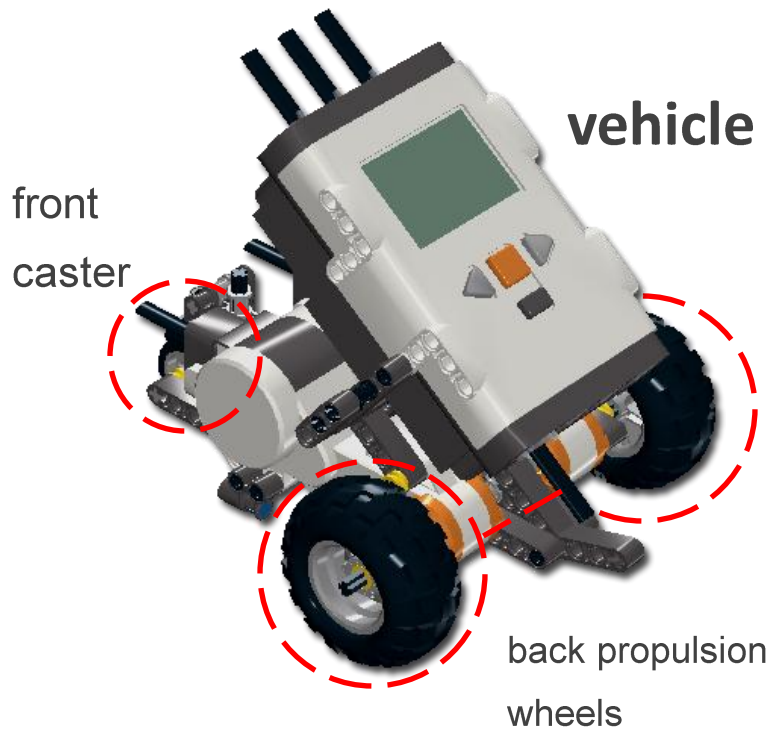
```
1 -- tasking.adb - application tasks for the 'hello' example
2 with Ada.Real_Time;
3 -- used for Clock, Time_Span, Milliseconds
4 with NXT.Display; use NXT.Display;
5 -- used for Put_Line;
6 with NXT.AVR;      use NXT.AVR;
7
8 package body Tasking is
9   use Ada.Real_Time;
10
11   -- Task and protected object declarations --
12   type Cycle_Count is mod 10;
13
14   task Periodic is
15     pragma Priority (1);
16     pragma Storage_Size (4096);
17   end Periodic;
18
19   task Sporadic is
20     pragma Priority (2);
```
- Tasking Panel:** Shows 'tasking.adb' and 'hello.gpr'.
- Messages Window:** Contains compilation output for 'Tasking' and 'hello.gpr', including compiler flags and a successful termination message: [2012-05-22 10:19:36] process terminated successfully (elapsed time: 03.67s).

Development process

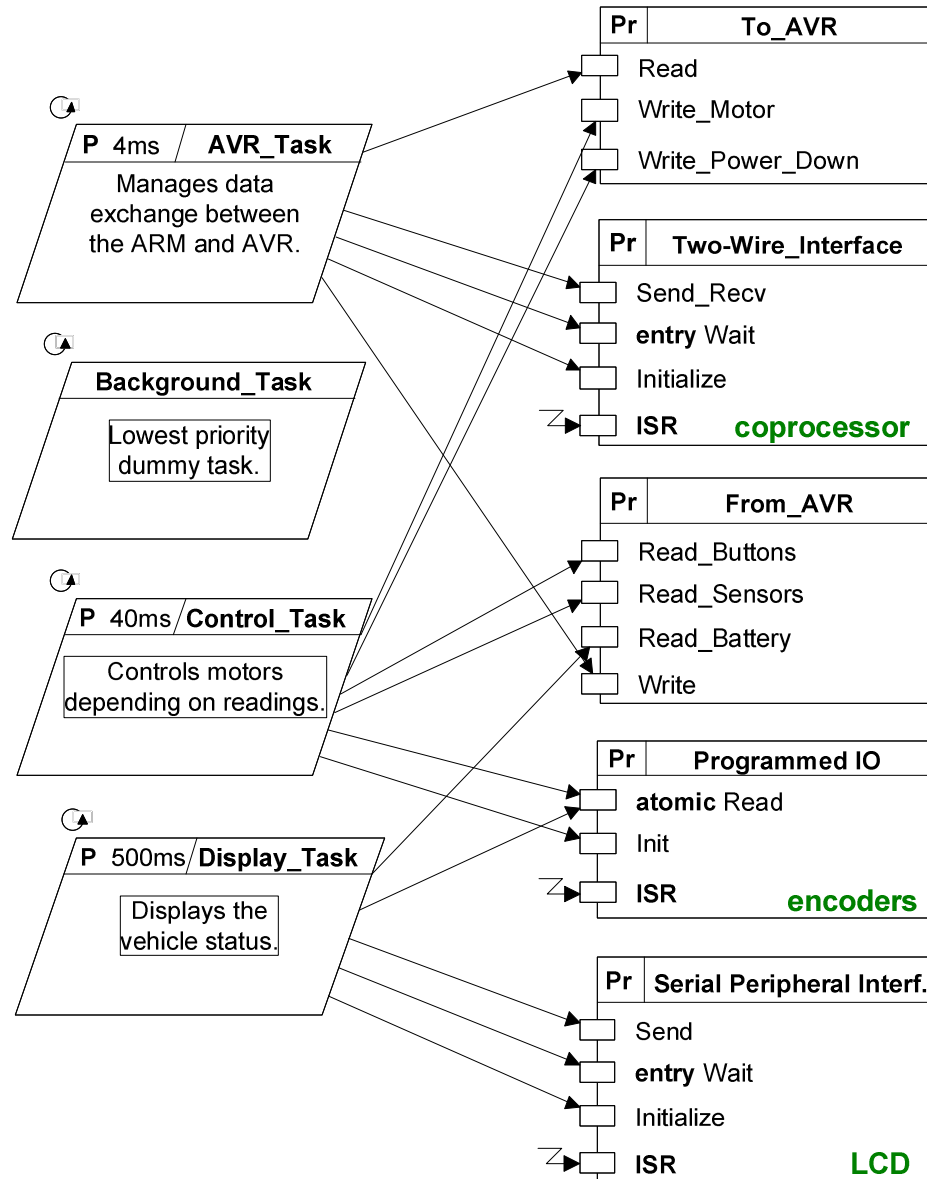


Vehicle example

- The vehicle has a **front caster** wheel used to turn and **two back wheels**, each with an independent motor



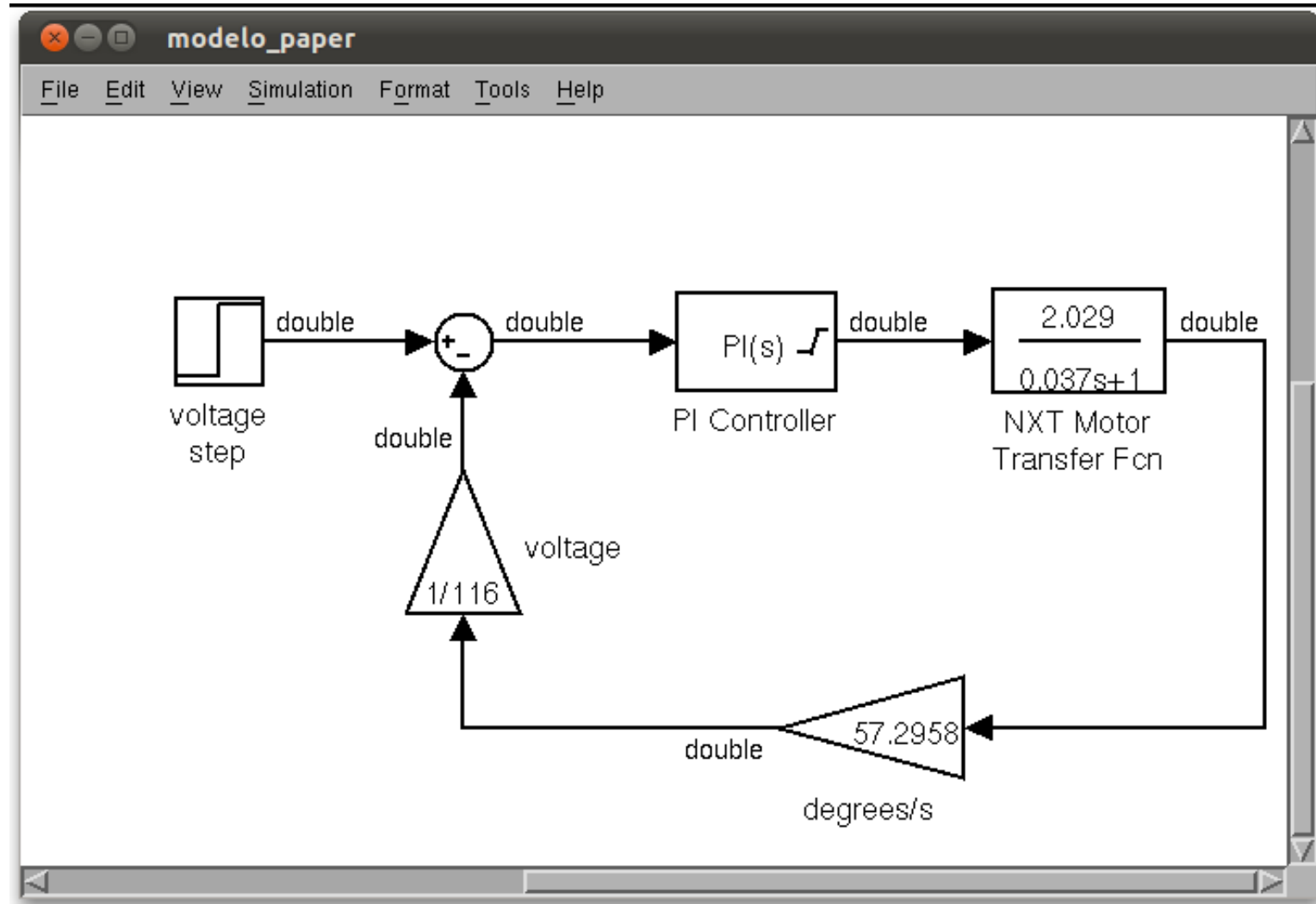
Software architecture



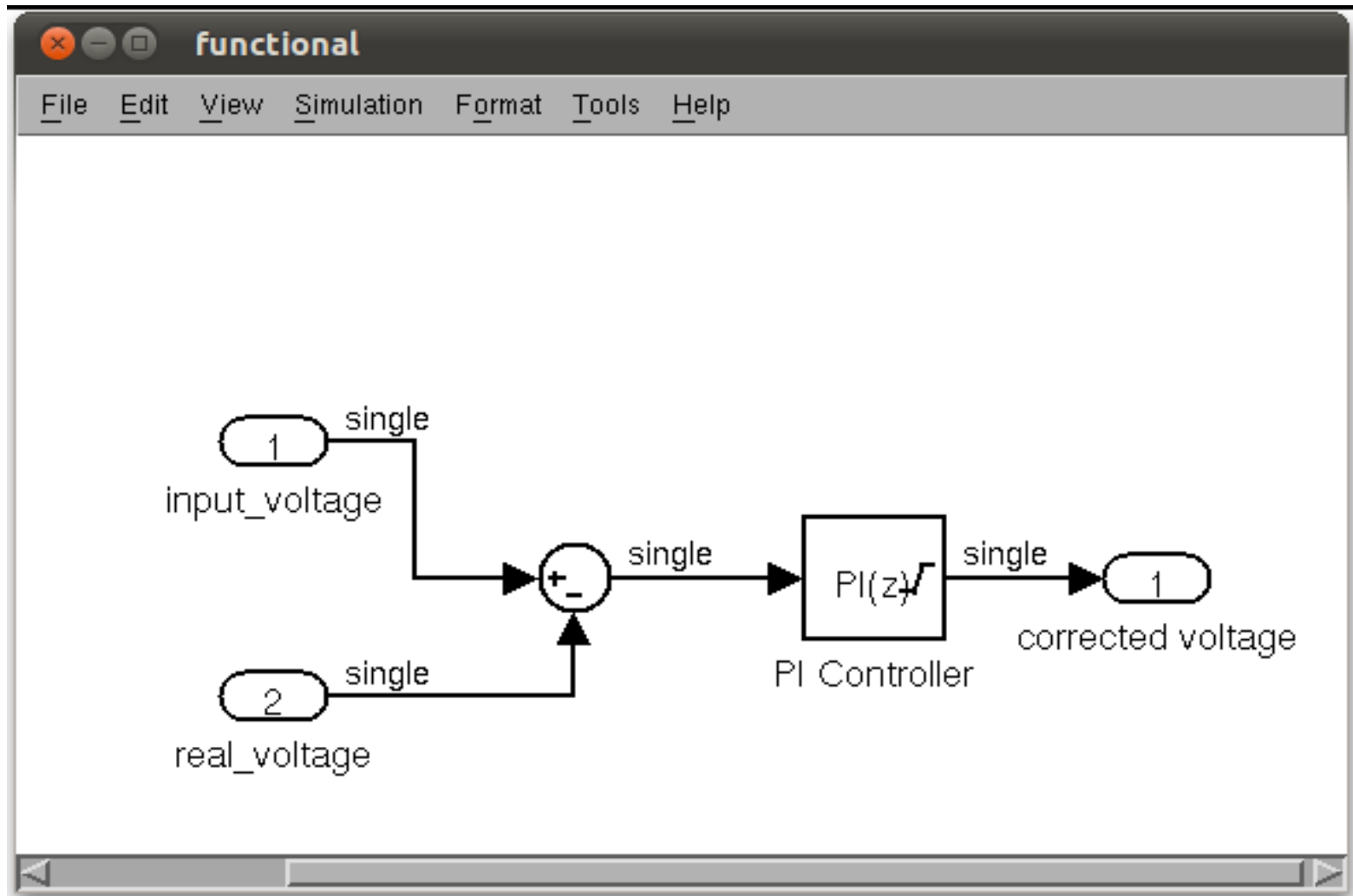
Demonstration

- Compile and download the code to the vehicle example.
- Run it and show that vehicle does not go straight.

Simulink control model



Simulink model for code generation



Including functional code

The screenshot displays the GNAT Programming Studio (GPS) interface for the 'Vehicle_V17' project. The main editor window shows the file `tasks.adb` with the following code:

```
8
9
10  -----
11  -- Importing the main of the C code
12  -----
13  procedure PID;
14  pragma Import (C, PID, "functional_step");
15  -----
16  -- Exporting functional_U module for
17  -----
18  -- Struct of functional_U module
19  type ExternalInputs_functional is
20     record
21       Input_Voltage : Float;
22       Real_Voltage  : Float;
23     end record;
24
25  -- functional_U object
26  Functional_U : ExternalInputs_functional;
27
28  pragma Export (C, Functional_U, "functional_U");
```

The left sidebar shows the project structure:

- Vehicle_V17
 - tasks.adb
 - package
 - Tasks
 - pragma
 - subprogram
 - task
 - type
 - variable

The bottom status bar indicates the current file is `tasks.adb` and provides information such as 'Subversion: 189 (Up to date)', 'Insert', 'Writable', 'Unmodified', and the time '10:39'. The Messages window at the bottom displays the following text:

```
Messages
Welcome to GPS 5.0.1 (20110113) hosted on i686-pc-linux-gnu
the GNAT Programming Studio

This is the GPS GPL Edition.

For professional needs, use the GPS version supplied with GNAT Pro.
For details contact us at sales@adacore.com
(c) 2001-2011 AdaCore
```

Demonstration

- Compile and download the code to the vehicle example
- Vehicle does go straight

Resumen

- La tecnología de software convencional no es adecuada, en general, para desarrollar sistemas empotrados de tiempo real.
- El desarrollo de sistemas empotrados presenta una complejidad añadida debido a la utilización de entornos diferentes para desarrollo y ejecución.
- Existe gran variedad de sistemas operativos para sistemas empotrados.
 - La elección de uno u otro depende de las necesidades de cada aplicación (tamaño del sistema, certificación, tiempo real estricto, etc.).
 - Existen versiones “empotrables” de sistemas operativos convencionales.