datsi

UPM

# Desarrollo de sistemas empotrados

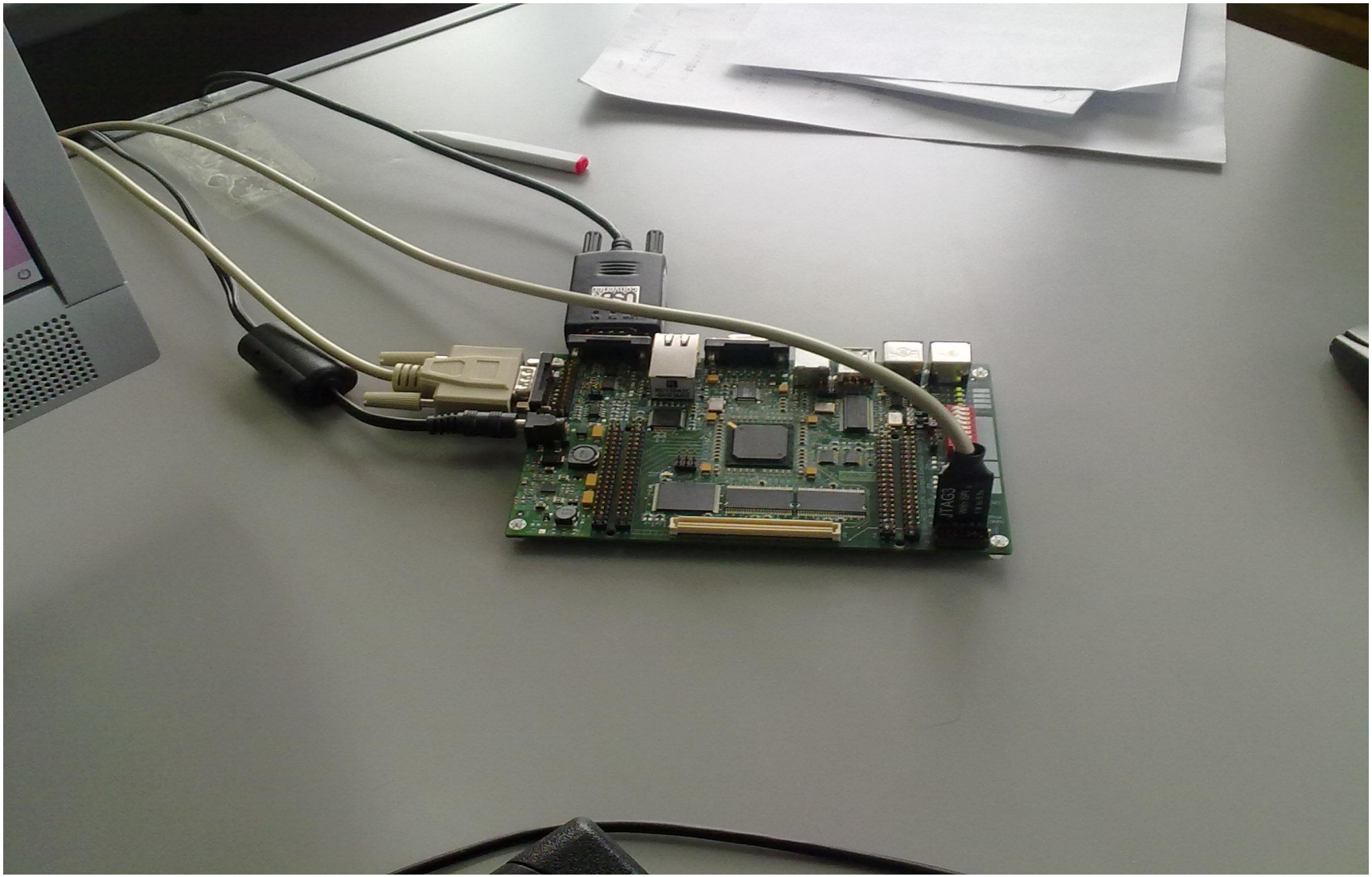
Juan Zamorano

# Motivación

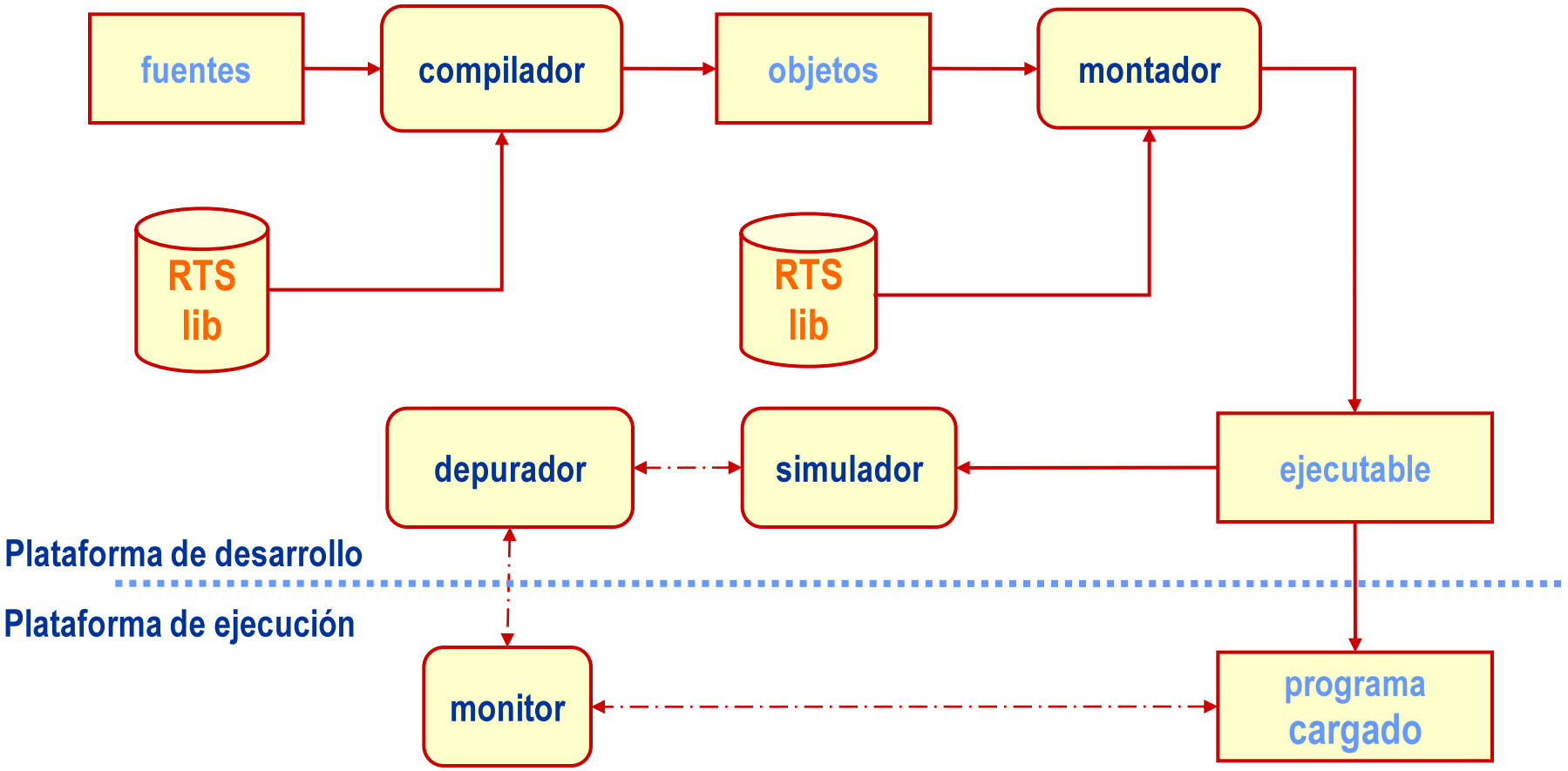
- Los métodos, las herramientas y la tecnología que se usan para construir otros tipos de sistemas no sirven en general para los sistemas empotrados
  - no son suficientemente fiables
  - sólo contemplan el tiempo de respuesta medio, no el peor
  - no garantizan los requisitos temporales

- 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

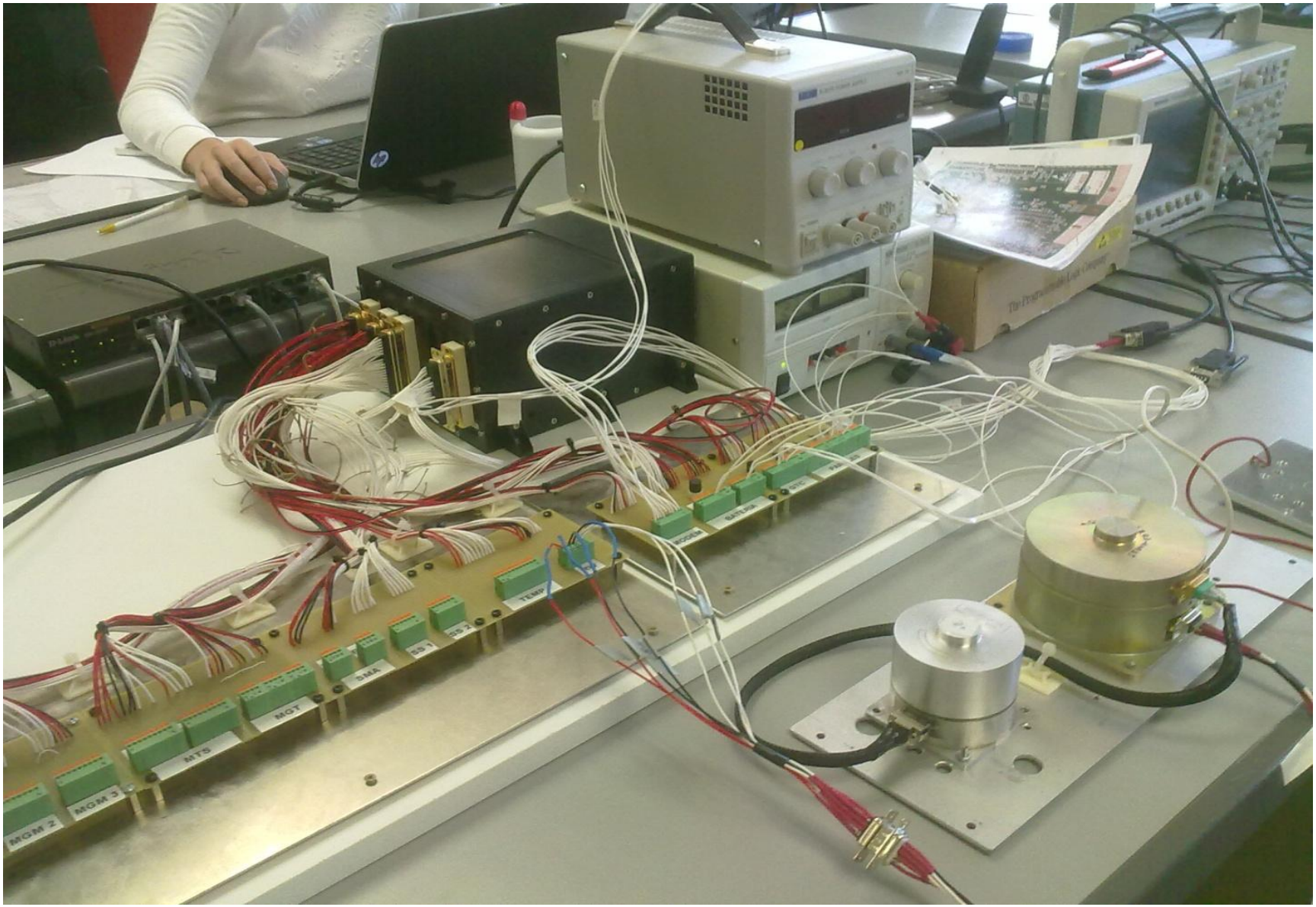
# Desarrollo cruzado



# 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 empotrados.
- 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

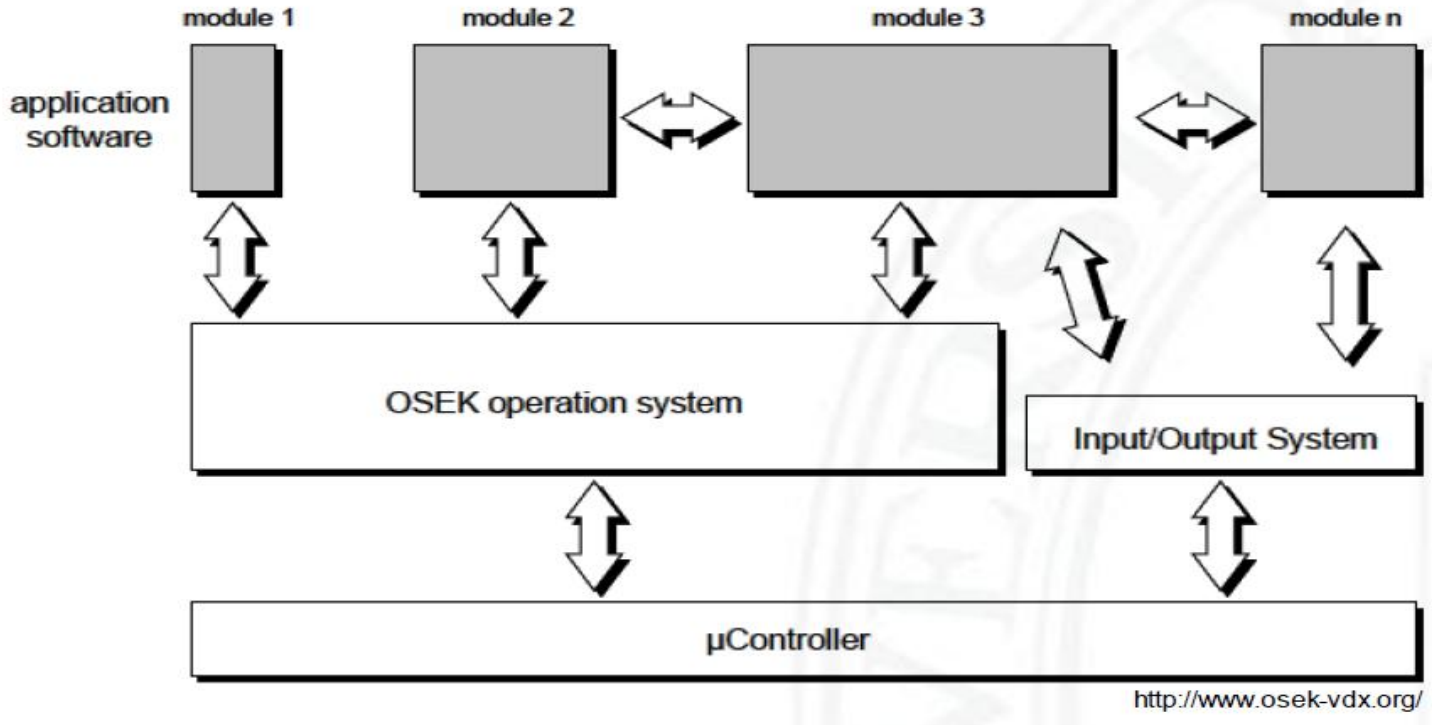
# Perfiles de aplicación

- 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

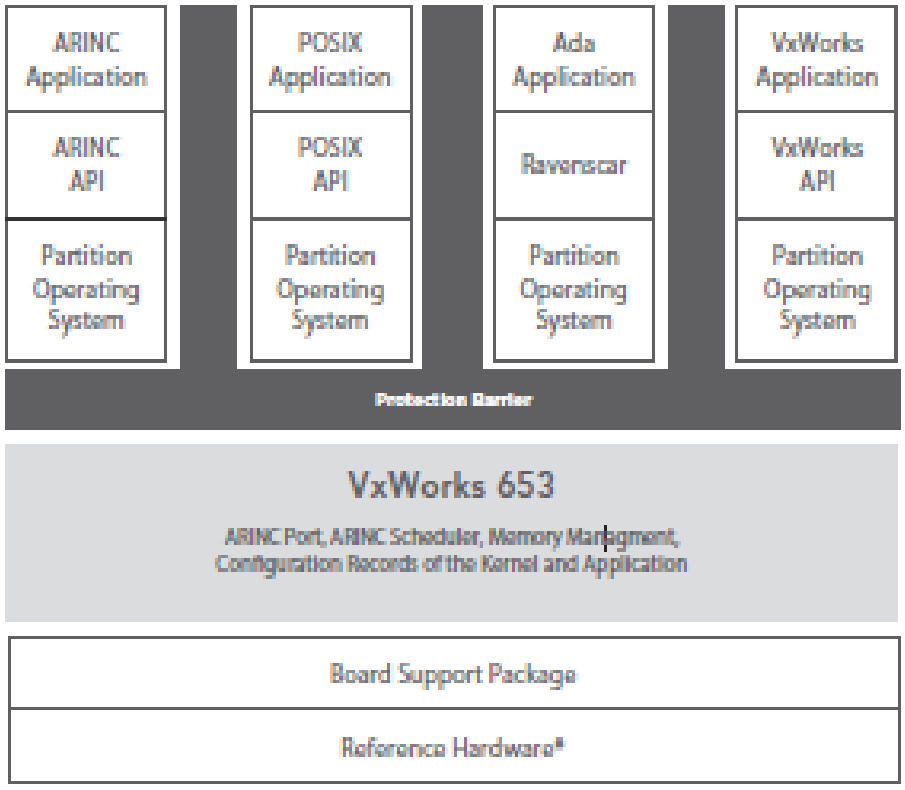
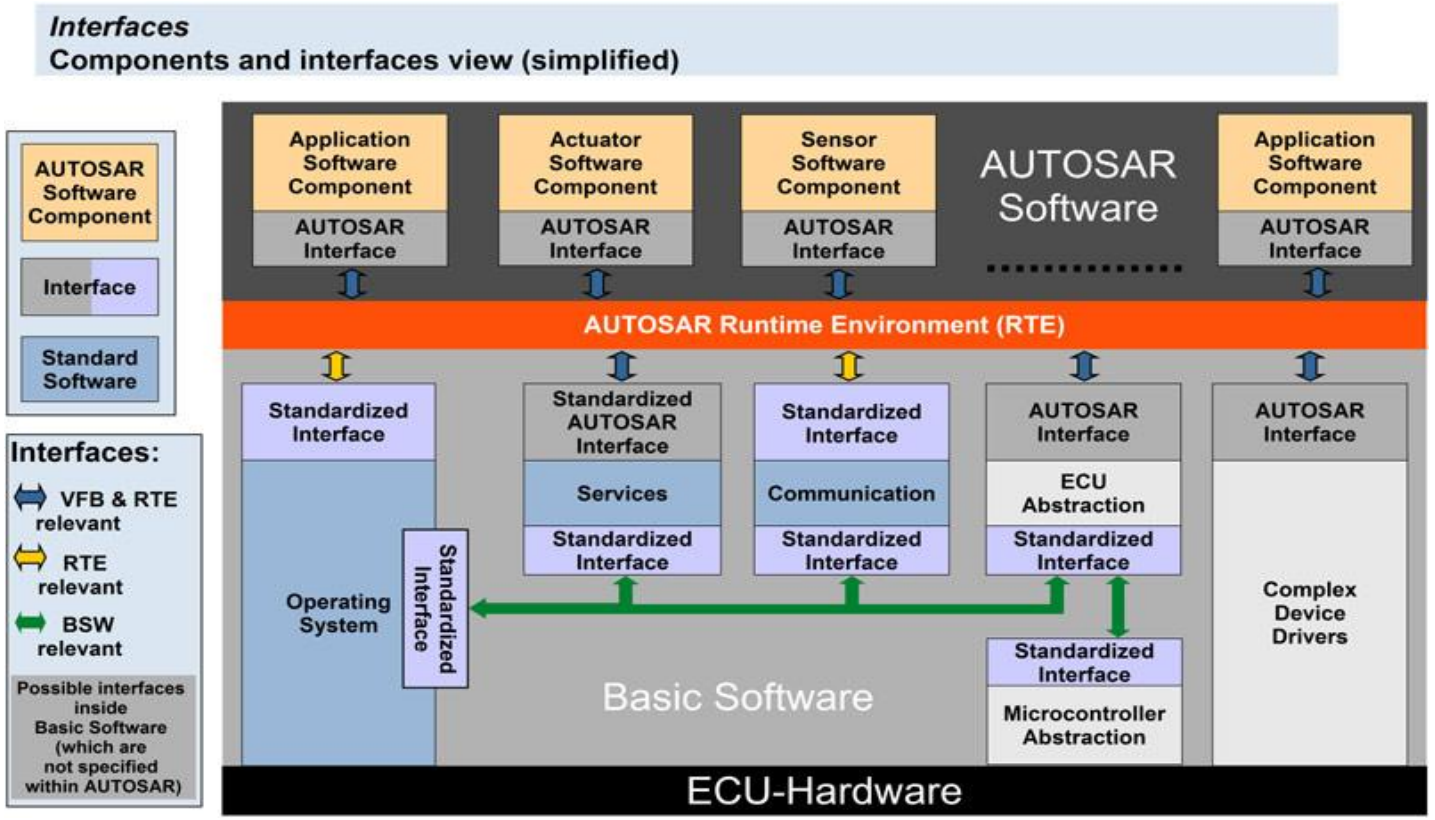



*Figure 2: IMA design with VxWorks 653 Platform*

# Otros estándares: AUTOSAR

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



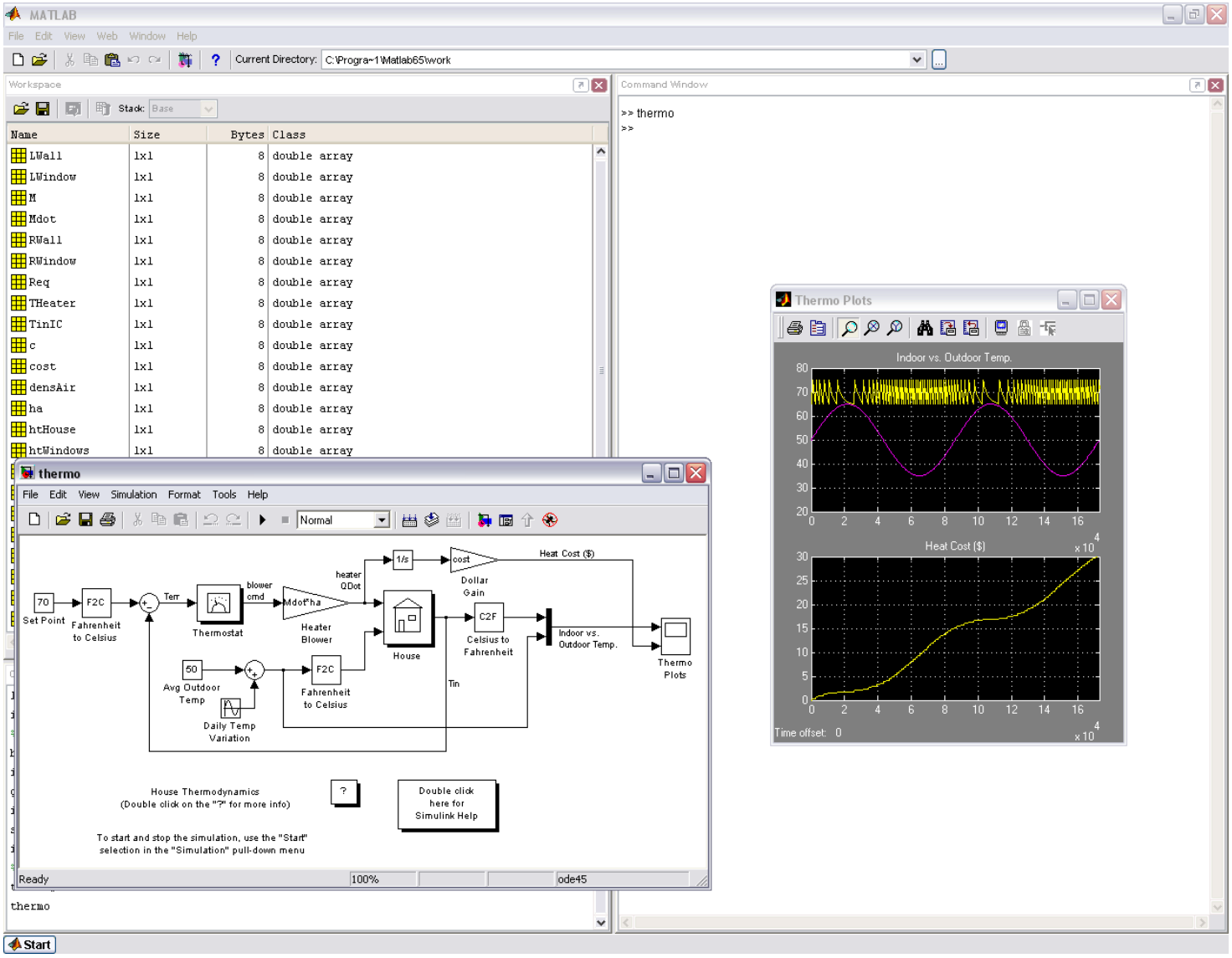
# Ejemplos de SOTR

- 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

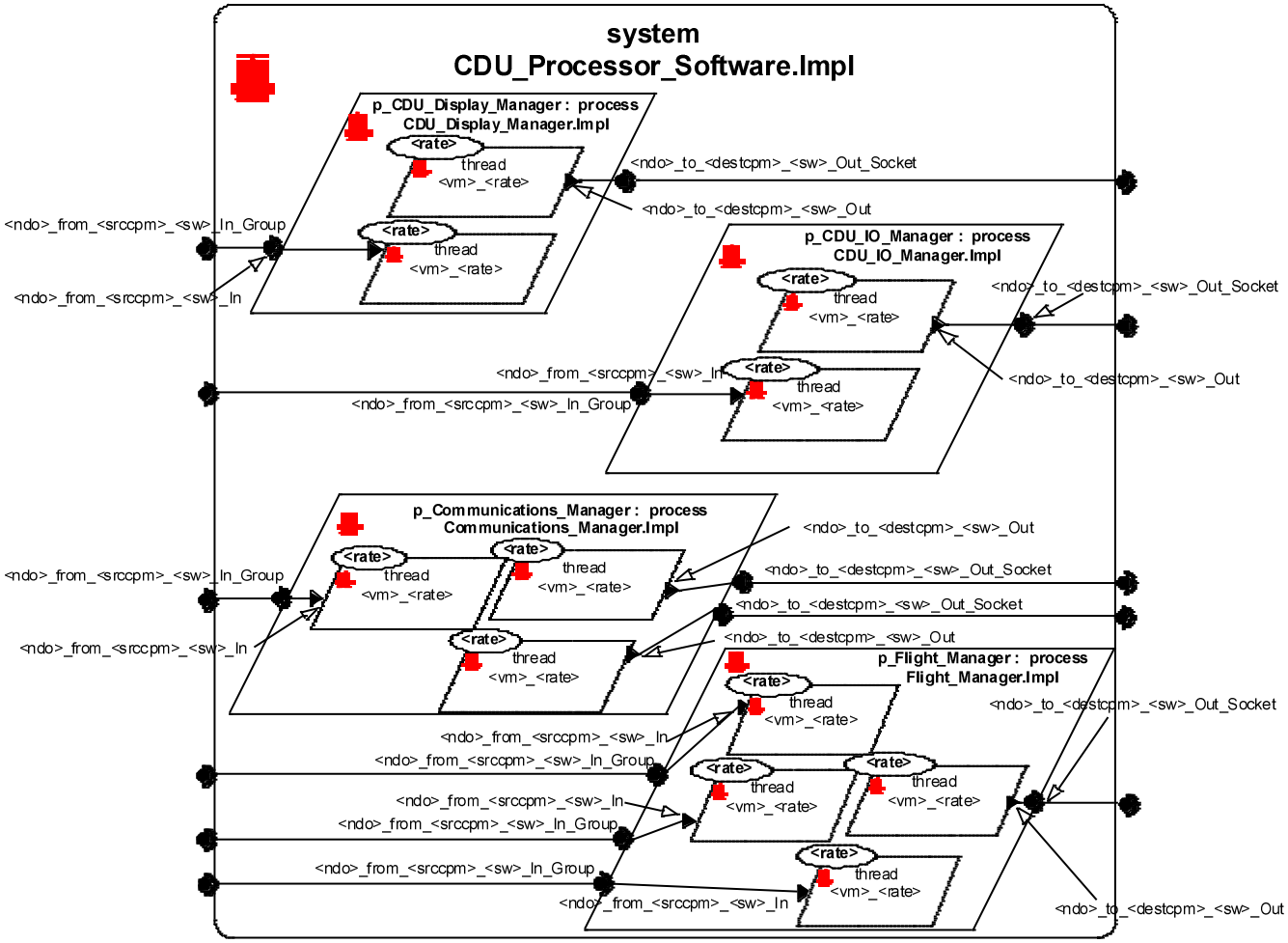
# Diseño de sistemas empotrados

- 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

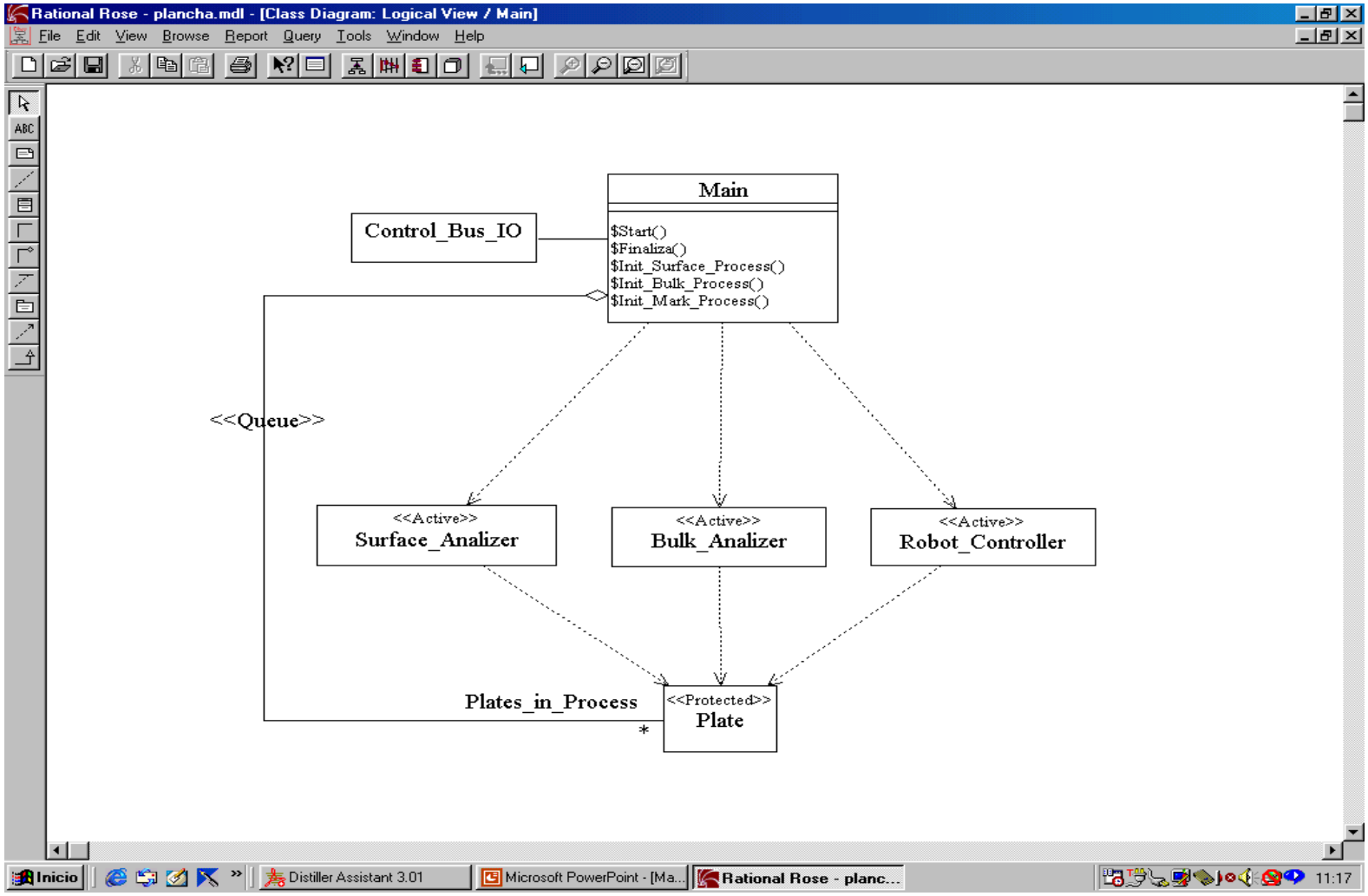
# Ejemplo: Simulink



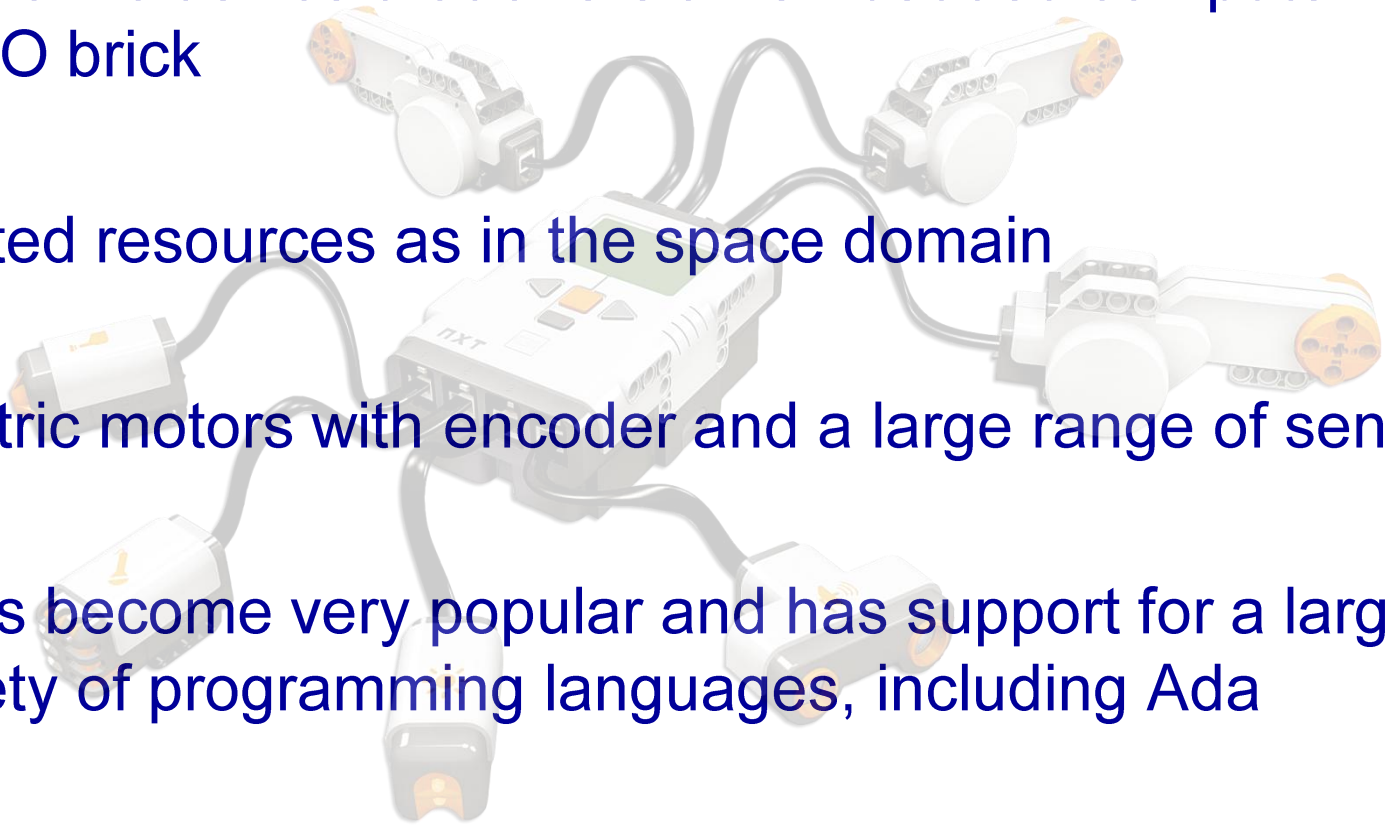
# Ejemplo: AADL



# Ejemplo: UML

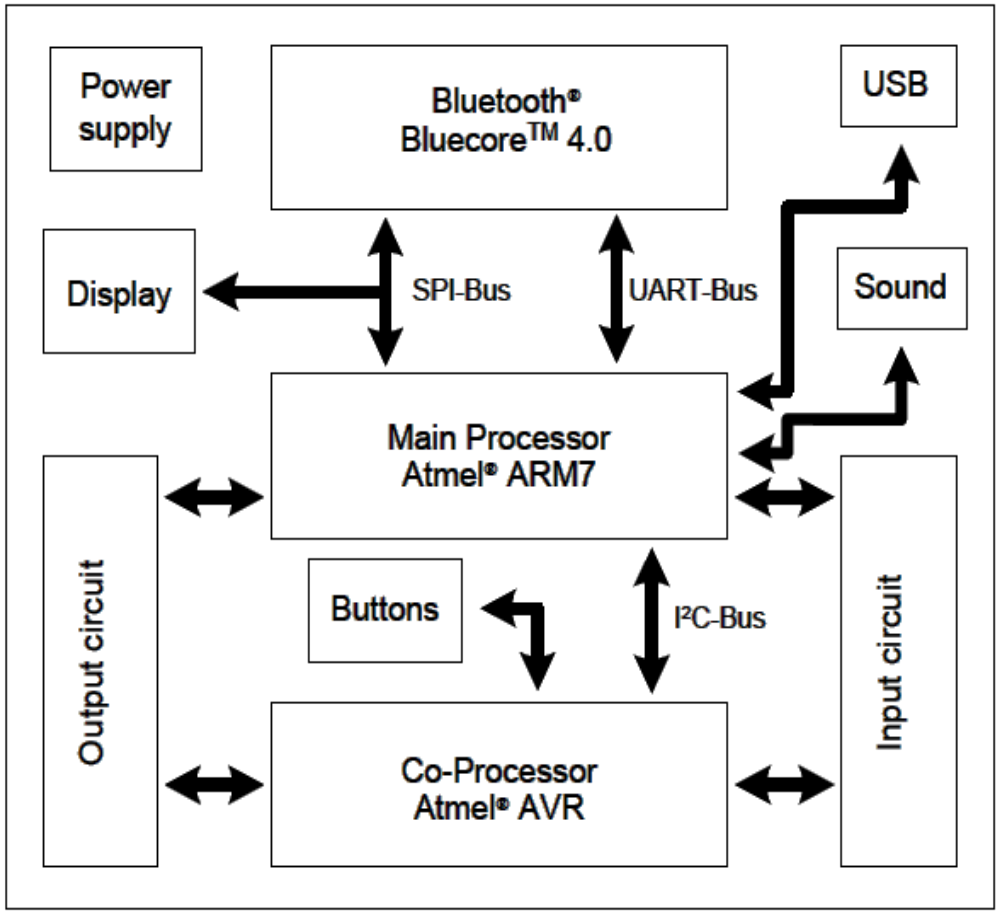

Rational Rose - plancha.mdl - [Class Diagram: Logical View / Main]
File Edit View Browse Report Query Tools Window Help
Main
$Start()
$Finaliza()
$Init_Surface_Process()
$Init_Bulk_Process()
$Init_Mark_Process()
Control_Bus_IO
<<Queue>>
<<Active>>
Surface_Analizer
<<Active>>
Bulk_Analizer
<<Active>>
Robot_Controller
Plates_in_Process
*
<<Protected>>
Plate
Inicio
Distiller Assistant 3.01
Microsoft PowerPoint - [Ma...
Rational Rose - planc...
11:17

# 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

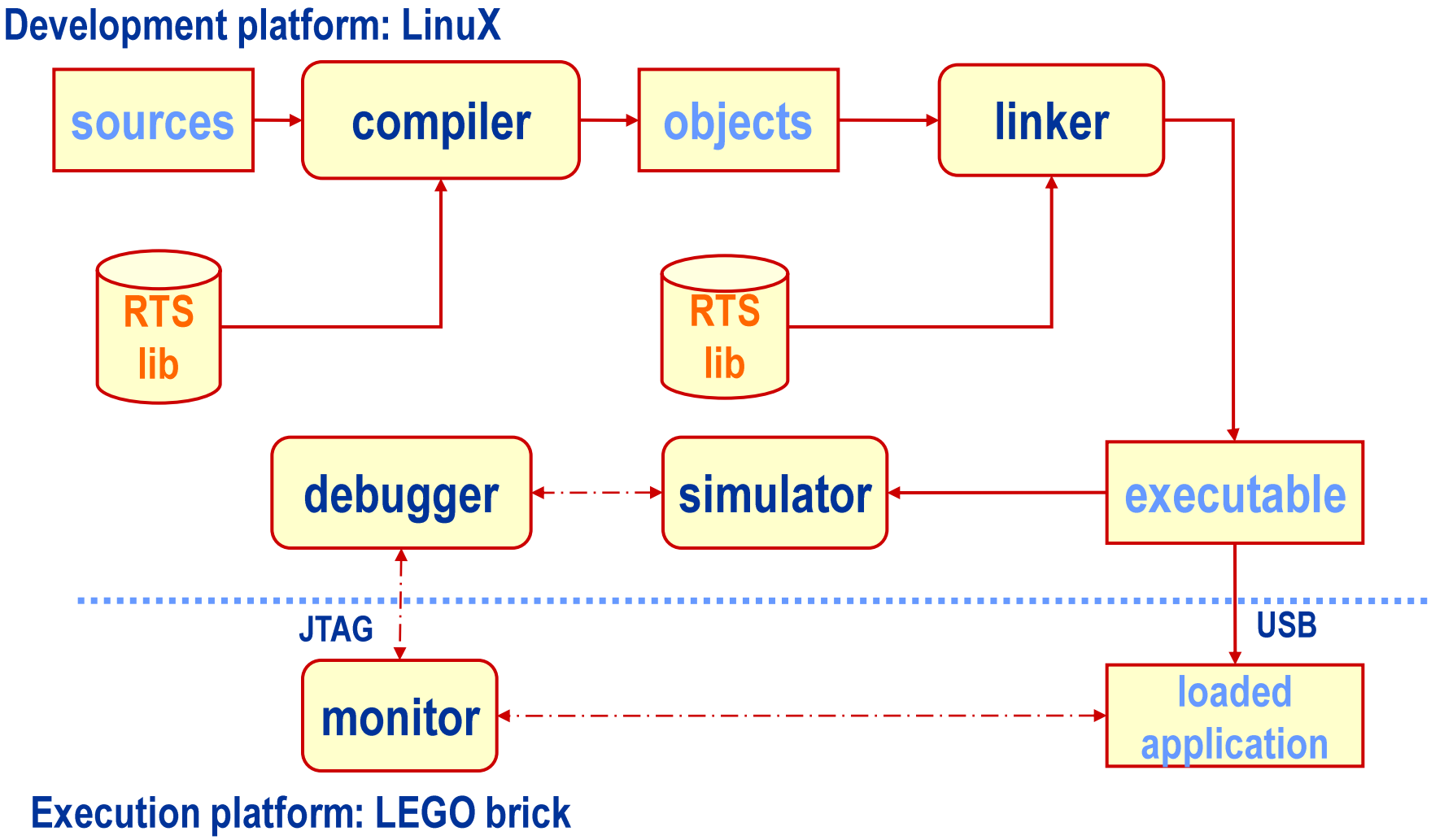
- Cross-compilation system targeted to LEGO MINDSTORMS computers
  - ARM7 processor
  - Includes drivers for MINDSTORMS devices
- Components
  - GNAT Ada compiler (Ada Core)
  - GNARL run-time system (Ada Core)
  - Adapted version of the ORK kernel (UPM)

# 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



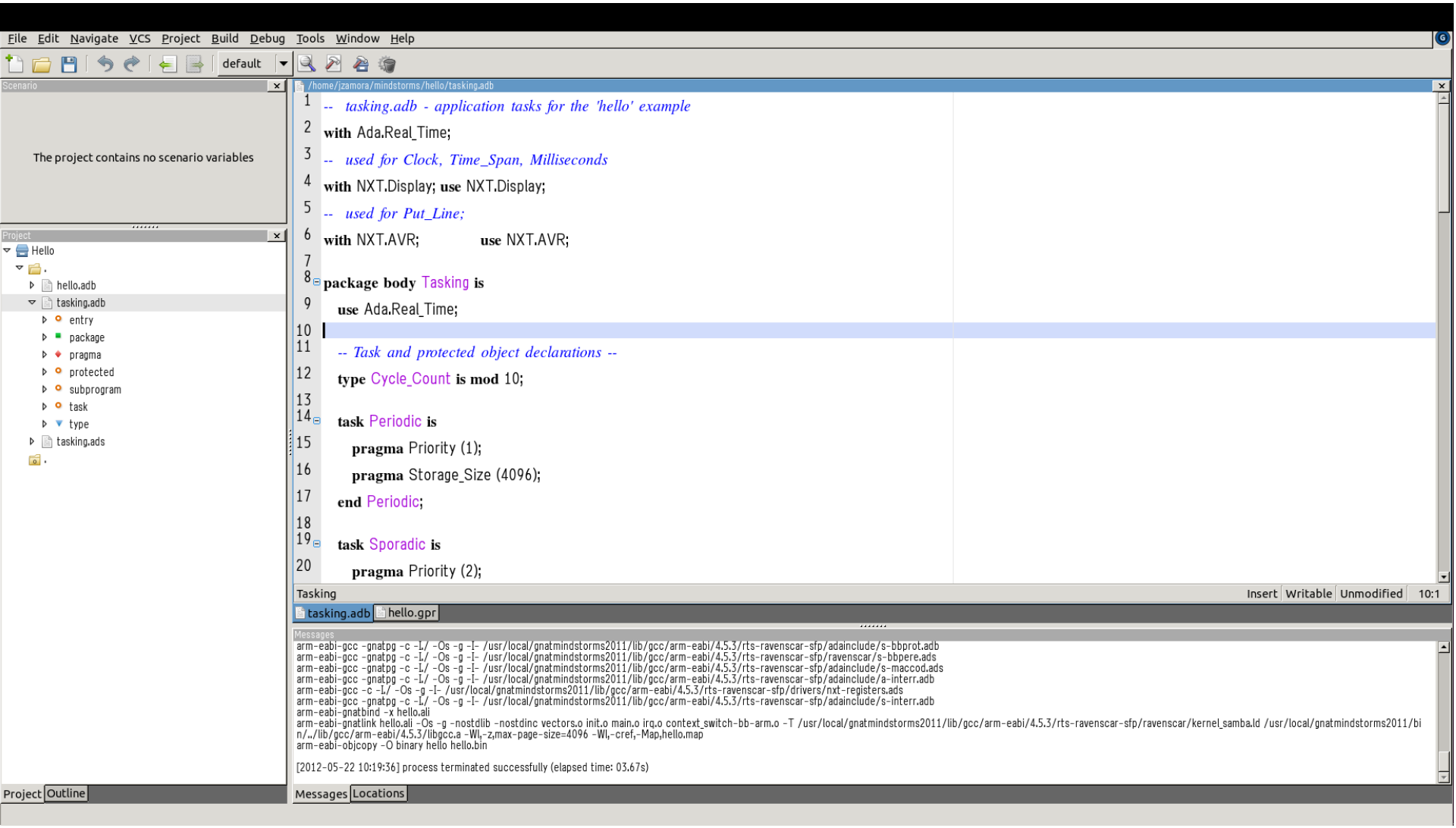
# Development tools

- 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

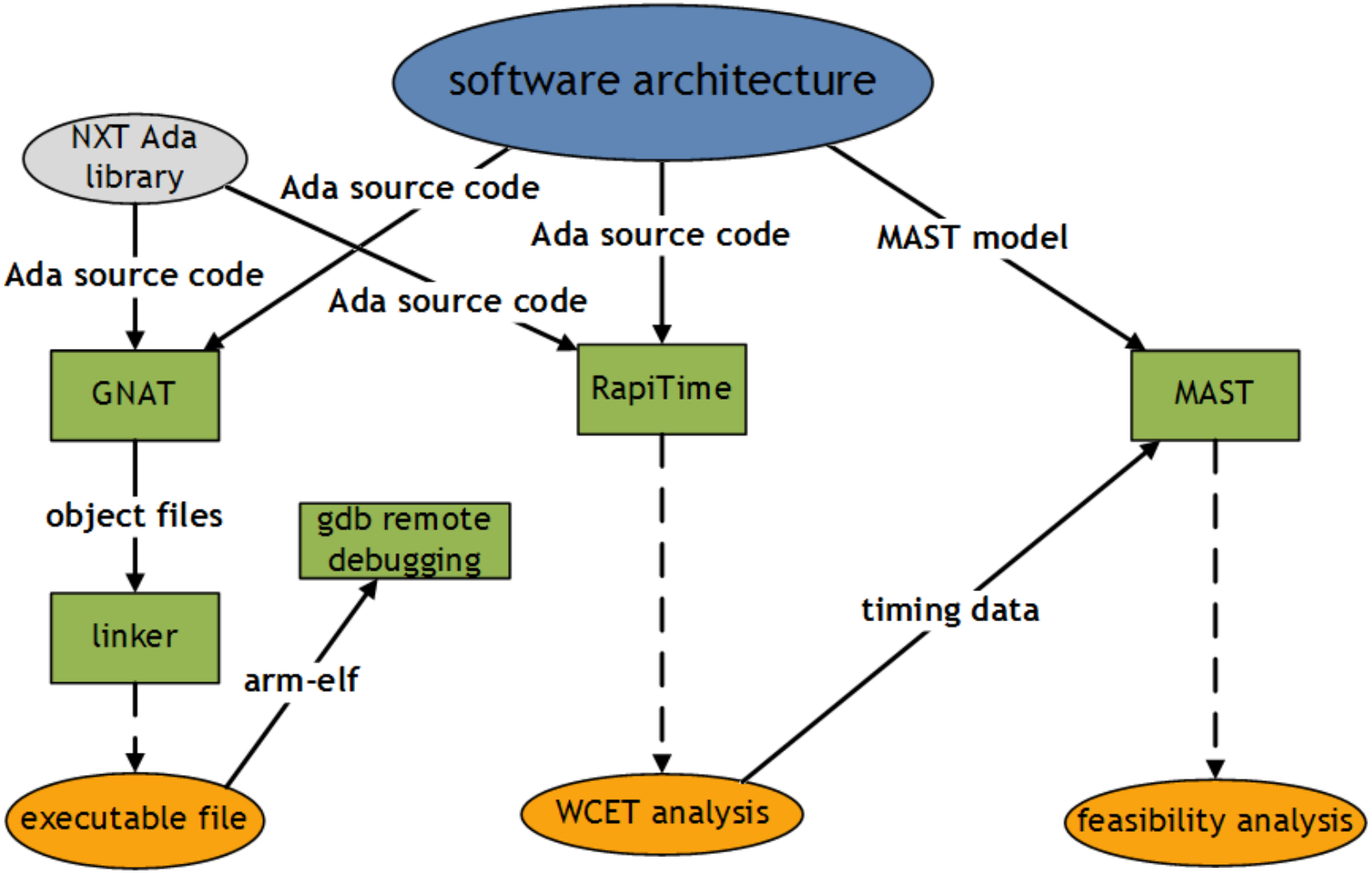
# A simple example

- 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

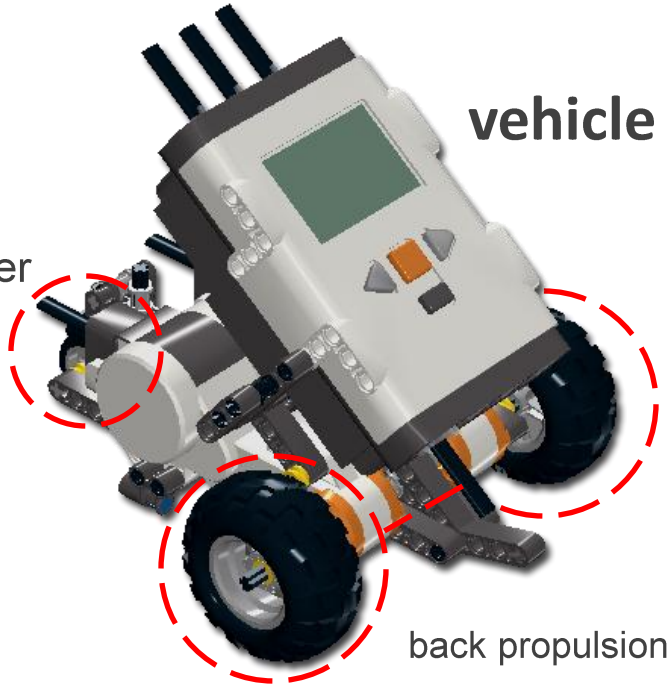


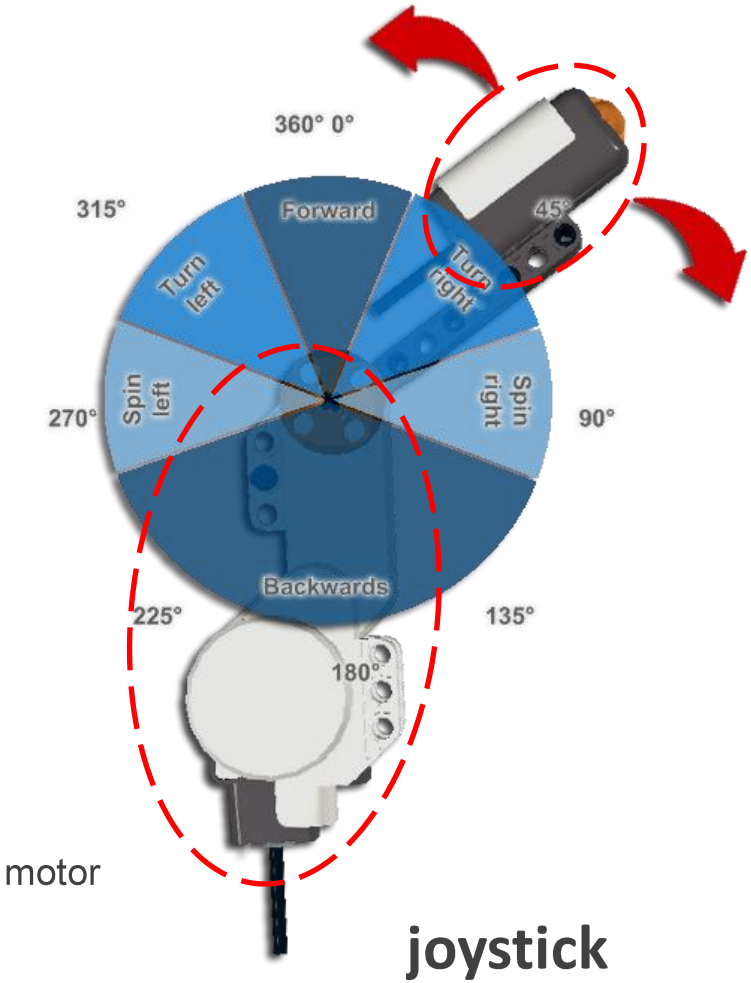
# Development process



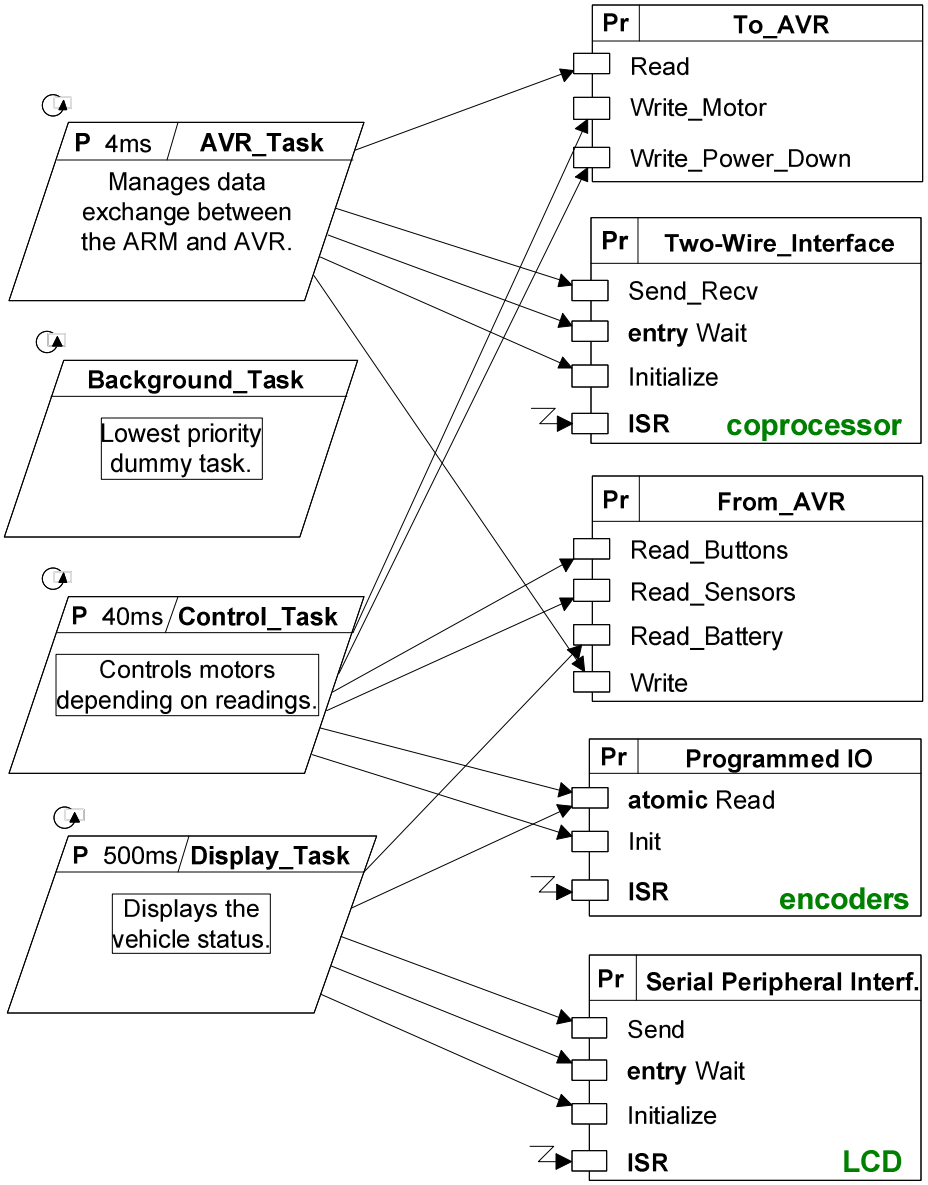
# Vehicle example

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



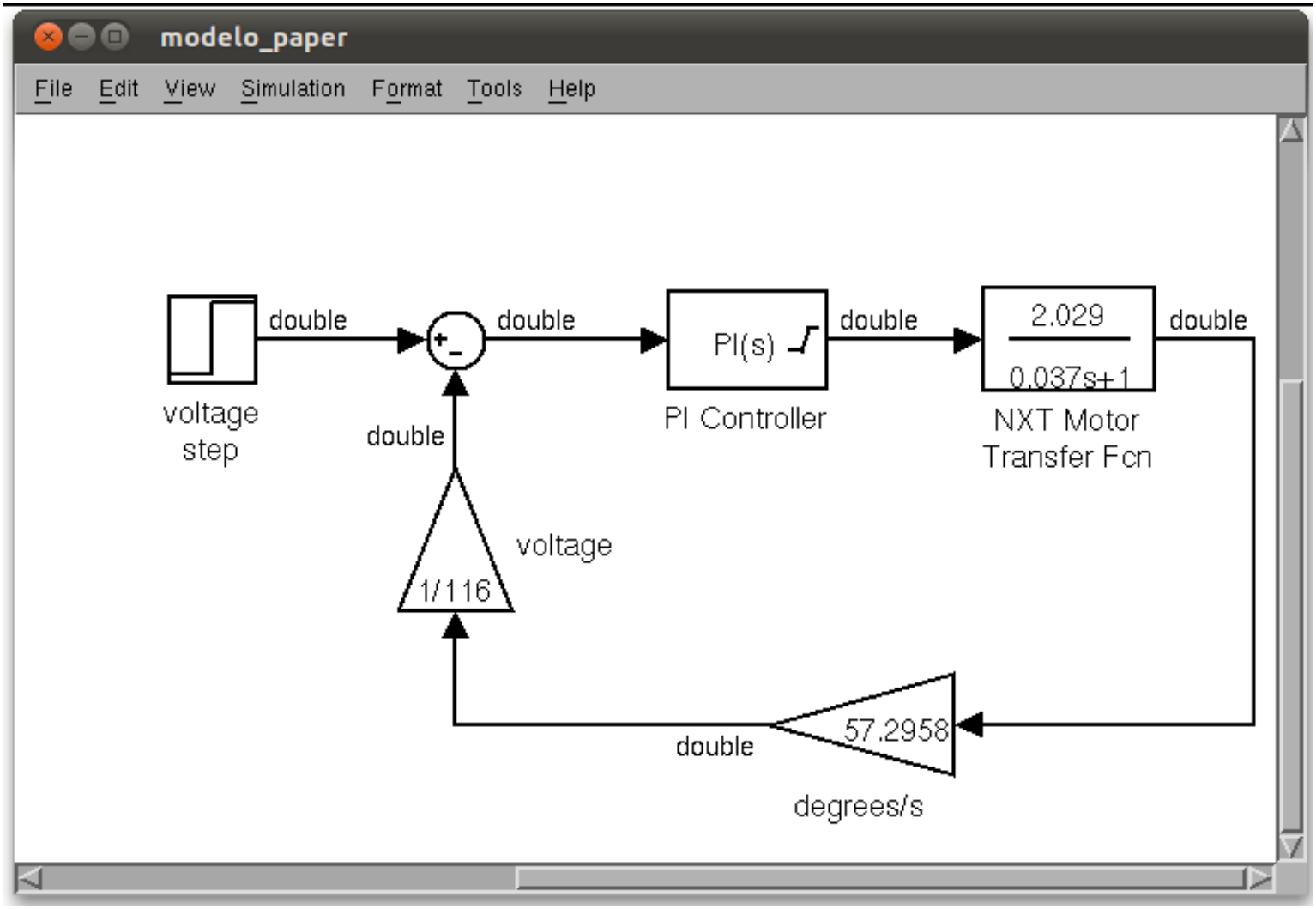
# Software architecture



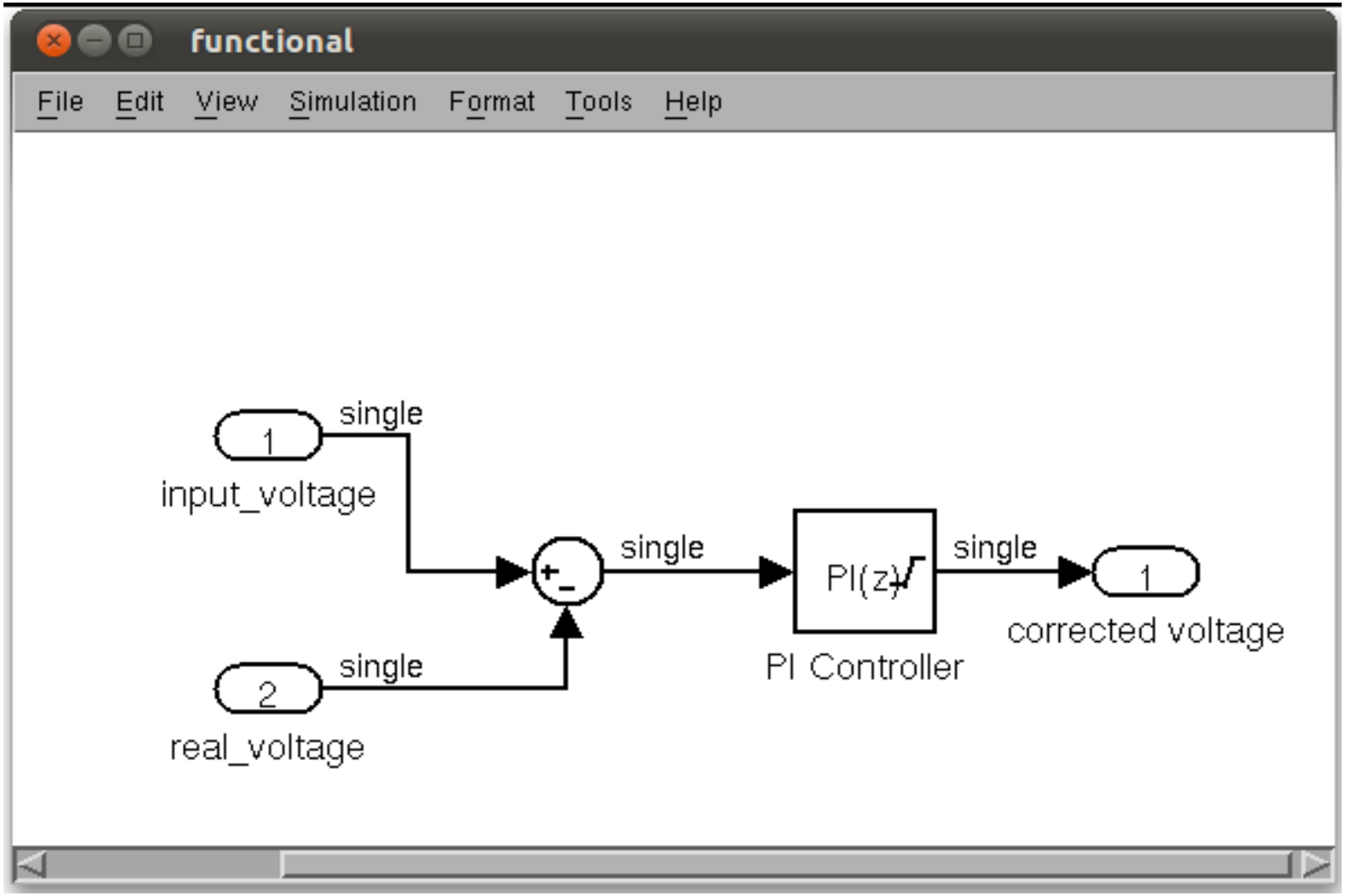
# Demonstration

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

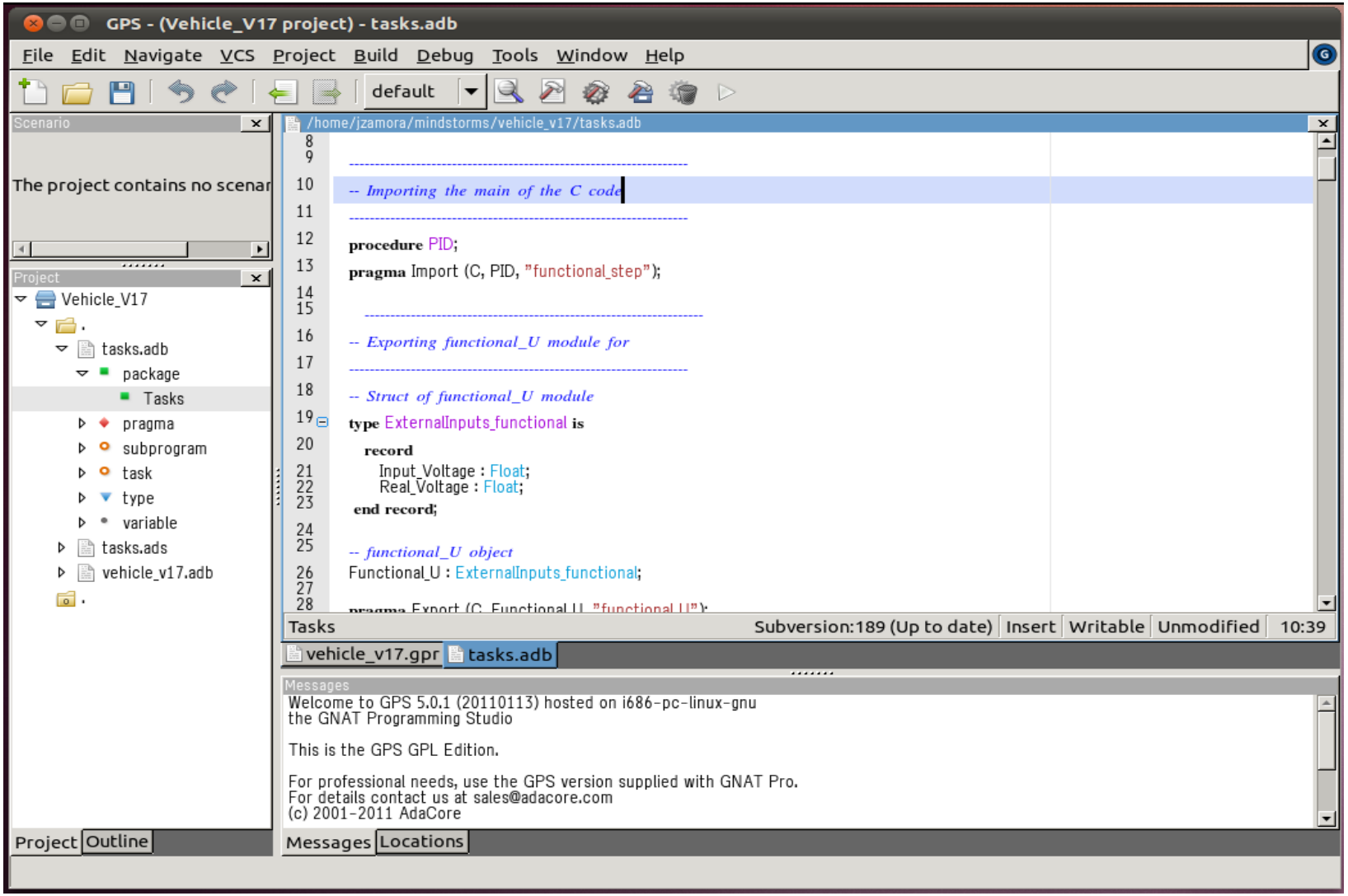
# Simulink control model



# Simulink model for code generation



# Including functional code



# 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.