UNIVERSIDAD DE CANTABRIA

Sistemas de Tiempo Real

Profesores: José Carlos Palencia palencij@unican.es
Héctor Pérez Tijero hector.perez@unican.es

Material basado en el tutorial realizado por Software Engineering Institute, Carnegie Mellon University

Programa

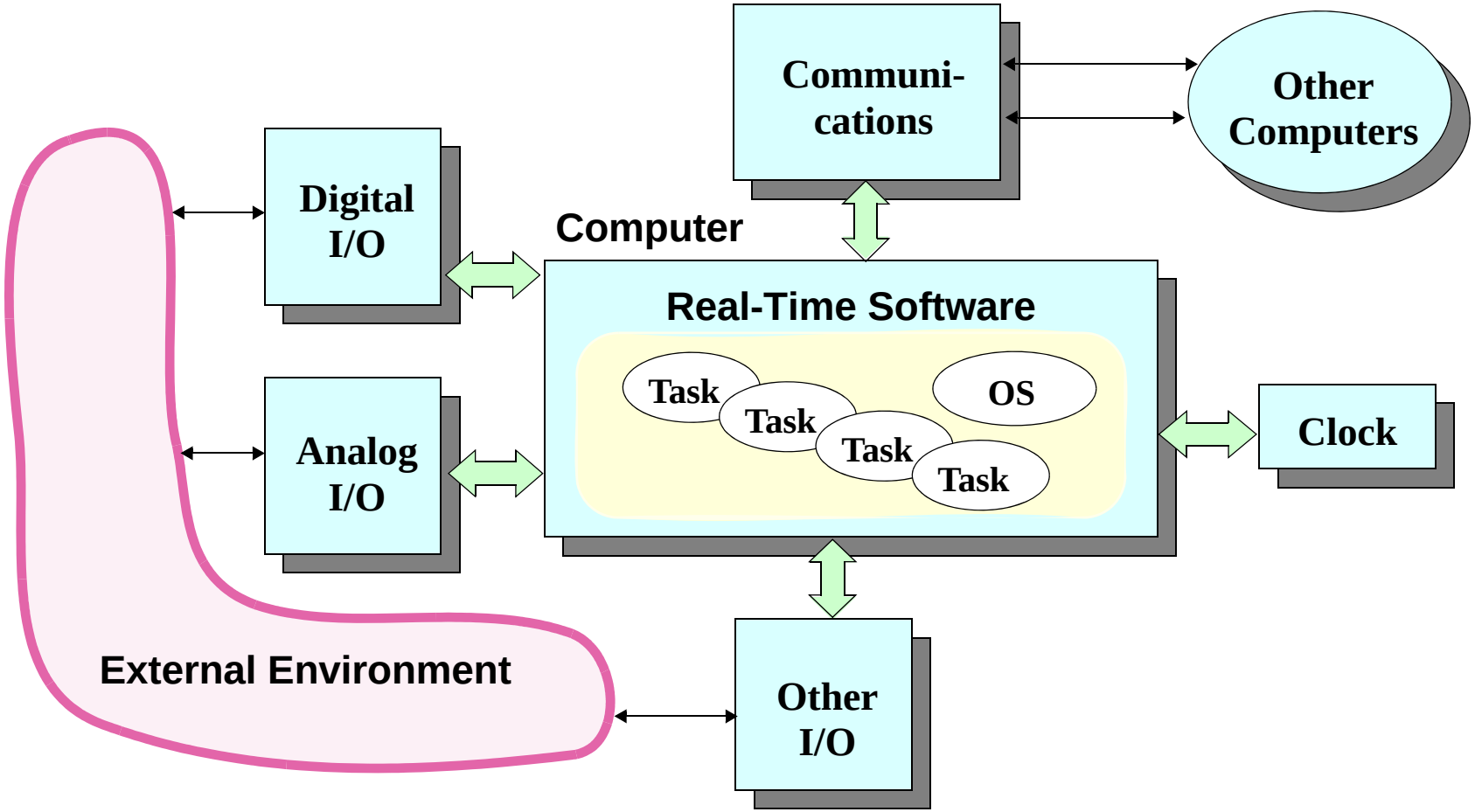
1. Introducción.
2. Eventos periódicos. Modelo básico.
3. Extensiones de la teoría básica.
4. Modelado de sistemas de tiempo real.
5. Recursos compartidos. Protocolos de sincronización.
6. Eventos aperiódicos. Servidores aperiódicos.
7. Soporte en sistemas operativos y en lenguaje ADA.
8. Aspectos avanzados.
9. Sistemas distribuidos.
10. Caso de estudio.
11. Bibliografía

1. Introduction

UNIVERSIDAD DE CANTABRIA

1.1 Properties and definition of a real-time system
1.2 Differences between real-time and conventional systems
1.3 Process scheduling
1.4 Timing requirements
1.5 Real-time scheduling policies
1.6 Modelling and analyzing real-time systems
1.7 Short history of real-time analysis
1.8 Sample case
1.9 MAST modelling environment

1.1 Properties and Definition of a Real-Time System



Real-time systems

UNIVERSIDAD DE CANTABRIA

A Real-time system is a combination of a computer, hardware I/O devices, and special-purpose software, in which:

- **there is a strong interaction with the environment**
- **the environment changes with time**
- **the system simultaneously controls and/or reacts to different aspects of the environment**

As a result:

- **timing requirements are imposed on software**
- **software is naturally concurrent**

To ensure that timing requirements are met, the system's timing behavior must be *predictable*

Notes:

UNIVERSIDAD DE CANTABRIA

In many real-time systems the activities that the system must execute are known in advance, and the code is active in the computer memory, ready to execute. The execution of each of the different activities is triggered by the arrival of a stimulus called an *event.* Examples of events are those produced by a clock to activate periodic activities, an interrupt request from a HW device, etc.

The activity itself that is triggered by the event is called a *response*. Since each response is usually implemented as an independent thread of control called a *task*, we will make no further distinction between responses and tasks (sometimes, a response may be implemented using several tasks, or no tasks at all, but we can think of a response as a conceptual task).

During the execution of the response, different system *resources* may be needed, such as the CPU, pieces of shared data, hardware devices, etc. Since a response may be rather complex depending on the kinds of resources needed to execute each part, we will decompose a response into one or more *actions*. An action is characterized because it uses a certain number of resources during all of its execution, and it does not change the scheduling parameters used to request those resources.

The execution of the response (and sometimes of the individual actions) may have *timing requirements* that must be met. In many cases the timing requirement refers to the maximum response time that is allowed for the response, counting from the arrival of the triggering event. This kind of requirement is called a *deadline*.

Real-Time Systems

UNIVERSIDAD DE CANTABRIA

"In Real-time applications, the correctness of computation depends upon not only the results of computation, but also the time at which outputs are generated."

Real-time requirements:

- **Perform periodic activities, requested at regular intervals**
  - **Complete work before a deadline**
- **Respond to aperiodic events, requested at irregular intervals**
  - **In some cases, complete work before a deadline**
  - **Meet an average response time requirement**
- **Ensure critical deadlines are met, even during transient overload**
- **Ensure consistency of shared resources**

Notes:

UNIVERSIDAD DE CANTABRIA

Real-time systems generally consist of independently schedulable threads of control called tasks. Here the term *task* refers to any schedulable unit of processing, like a process or thread.

Tasks may be periodic or aperiodic

- **Periodic** tasks are initiated at regular intervals.
- **Aperiodic** tasks are initiated at irregular intervals.

Tasks have deadlines which may be *hard* or *soft*.

- Failure to meet **hard** deadlines results in system failure.
- Failure to meet **soft** deadlines degrades system performance.

Tasks may be further categorized as being *critical* or *non-critical*. Failure of a *critical* task to meet a hard deadline (if one exists) is considered catastrophic.

1.2 Differences Between Real-Time and Conventional Systems

UNIVERSIDAD DE CANTABRIA

Predictability **of the response time**

Criteria **for real-time systems differ from that for time-sharing systems.**

	Time-Share Systems	Real-Time Systems
Capacity	High throughput	Ability to meet timing requirements: Schedulability
Responsiveness	Fast average response	Ensured worst-case latency
Overload	Fairness	Stability of critical part

Worst case cannot be checked by *testing*

Notes:

UNIVERSIDAD DE CANTABRIA

In time-sharing systems, the measures of merit are high throughput, fast average response time, and fairness to all the users. Different measures of merit are appropriate for real-time systems, and these different measures make real-time programming substantively different from non-real-time programming. In particular, non-real-time programming is often concerned with improving average-case performance, but real-time programming is typically concerned with ensuring that worst-case behavior is acceptable.

In real-time systems, the ability to meet deadlines is paramount. Schedulability is the appropriate measure of this requirement, while in other kinds of systems, average throughput is a good measure. Schedulability can be thought of as a refinement of the throughput measure by counting only those tasks that can meet their deadlines.

In real-time systems, latency or worst-case response for hard real-time tasks is used as a criterion, while fast average response time is retained as a measure for soft real-time tasks. However, the notion of fairness is discarded and replaced with the new notion of stability.

Both fairness and stability concern the characteristics of system resource allocation. In a real-time system, when all the timing constraints are met, there will be no starvation and fairness is not an issue. When it is impossible to meet all the timing constraints, we must ensure the deadlines of critical tasks, even at the cost of starving non-critical tasks, i.e., the notion of fairness is not appropriate.

1.3 Process Scheduling

UNIVERSIDAD DE CANTABRIA

Compile-time schedules:

- time triggered or cyclic executives
- predictability through static schedule
- logical integrity often compromised by timing structure
- difficult to handle aperiodic events & dynamic changes
- difficult to maintain

Run-time schedules:

- priority-based schedulers
- preemptive or non preemptive
- analytical methods needed for predictability
- separates logical structure from timing
- more flexibility

Notes:

UNIVERSIDAD DE CANTABRIA

- Compile-time:
  - Cyclic executive - where all work (tasks) are fit into a common period (major frame) and executed non-preemptively.

- Run-time, preemptive, fixed priority:
  - Deadline monotonic - preemptive execution where tasks are assigned priorities based on fixed deadlines.
  - Rate monotonic - preemptive scheduling where tasks are assigned priorities based on rates, or periods.

- Run-time, preemptive, dynamic priority:
  - Earliest deadline first - preemptive scheduling, where tasks are assigned priorities dynamically, based on closeness of current deadlines.
  - Least laxity first - preemptive scheduling, where tasks are assigned priorities dynamically, based on the amount of slack time that remains between time needed to complete work and time to the deadline.

- Run-time, non preemptive: once a task starts executing, it runs until completion.

Example: Control in an Automobile

Activities to control:

Speed measurement
C=4 ms.
T=20 ms.
D=5 ms.

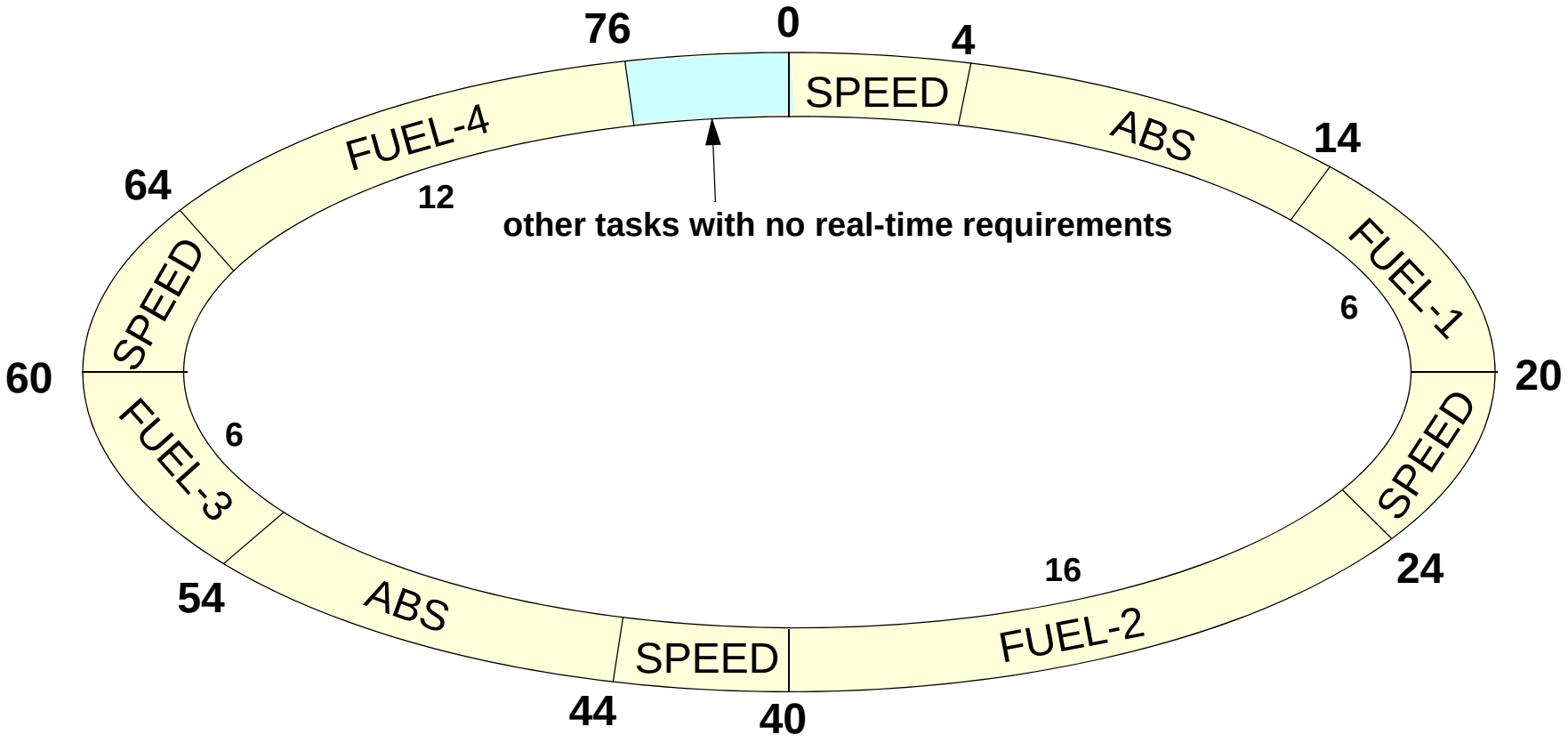
ABS Control
C=10 ms.
T=40 ms.
D=40 ms.

Fuel Injection
C=40 ms.
T=80 ms.
D=80 ms.

C = Execution time
T = Period
D = Deadline

Example: cyclic solution

A cyclic solution implies breaking the longest activity in several parts:



Example: concurrent solution

The concurrent solution is easier to design and maintain:

Speed measurement

```
loop
   Measure_Speed;
   next:=next+0.02;
   Sleep_until next;
end loop;
```

ABS control

```
loop
   ABS_Control;
   next:=next+0.04;
   Sleep_until next;
end loop;
```

Fuel injection

```
loop
   Fuel_Injection;
   next:=next+0.08;
   Sleep_until next;
end loop;
```

Activities with no timing requirements

```
loop
   do work;
   ...
end loop;
```

Example: maintenance

Suppose that we need to execute an aperiodic activity with a 1ms execution time. The minimum interarrival time is 80ms., but the deadline is 20 ms:

1. **Cyclic solution:**
   - **Sample at least every 19 ms to check if an aperiodic event has arrived**
   - **This implies breaking ABS and FUEL in several parts each**
2. **Concurrent solution:**
   - **Add a new high-priority process and repeat the schedulability analysis**
   - **No modifications to existing code**

1.4 Real-Time Scheduling Policies: Fixed-Priority Scheduling

UNIVERSIDAD DE CANTABRIA

Fixed-priority preemptive scheduling is very popular for practical applications, because:

- **Timing behavior is simpler to understand**
- **Behavior under transient overload is easy to manage**
- **A complete analytical technique exists**
- **High utilization levels may be achieved (typically 70% to 95% of CPU)**
- **Supported in standard concurrent languages or operating systems:**
  - **Ada 2005's RT-annex, Java RTSJ**
  - **Real-time POSIX**

Notes:

Criteria for real-time scheduling policies:

- Predictability: easy to analyze and understand timing behavior
- Achieve high utilization and still guarantee hard deadlines.
- Provide fast aperiodic response and still guarantee hard deadlines.
- Low overhead: dynamic policies tend to have higher overhead, since they may reorder the run queue at each scheduling decision.
- Maintain stability under transient overload
- Widely available in commercial standard products

Fixed-priority scheduling meets all these criteria, and is an excellent choice for building practical applications.

Dynamic priority scheduling

UNIVERSIDAD DE CANTABRIA

Most common policy is earliest deadline first (EDF):

- the priority varies with time, because the task with the closest absolute deadline is chosen first
- 100% CPU utilization may be achieved

However:

- behavior is more complex, and so is the analysis
- treatment of transient overload is more complex
- not supported by standard operating systems
  - supported in Java RTSJ
  - recently added to Ada 2005

Mixed EDF/FPS schemes are possibly the best approach

1.5 Modelling and Analyzing Real-Time Systems

UNIVERSIDAD DE CANTABRIA

Real-time analysis is an engineering basis for analyzing and designing real-time systems

Provides guidelines for optimum priority assignment

Provides an analytical framework for verifying timing requirements

Helps to identify timing bottlenecks and errors

Notes:

Scheduling theory is a mathematical approach for analyzing real-time tasks to determine their schedulability. It can be applied to a wide range of practical real-time systems.

In this tutorial, we will first introduce the mechanics of the analysis in an incremental way. First, we will start with a very restricted system for which we use the first theoretical results that appeared in rate monotonic scheduling. We will then systematically relax the assumptions and extend the basic theory to cover every aspect of a real-world system.

We will initially focus on fixed priority scheduling, but will later introduce concept of dynamic priority scheduling.

Why Are Deadlines Missed?

UNIVERSIDAD DE CANTABRIA

For a given task, consider:

- ***preemption*: time waiting for higher-priority tasks**
- ***execution*: time to do its own work**
- ***blocking*: time delayed by lower-priority tasks**

The task is *schedulable* if the sum of its preemption, execution, and blocking is less than its deadline for the worst possible case:

- ***Execution* is unavoidable (unless requirements change).**
- ***Preemption* can be minimized by choosing an optimum priority assignment**
- **Main focus: identify and reduce *blocking***

Notes:

For a task to meet its deadline, it must accommodate

- Preemption from higher-priority tasks,
- Its own execution time, and
- Delays caused by lower-priority tasks (known as priority inversion or blocking).

Execution time is unavoidable, unless the code itself is modified. Preemption can be minimized by choosing the appropriate (optimum) priority assignment.If execution and optimized preemption exceed the maximum system capacity, one is faced with a classical throughput problem and the only remedy is to reduce the workload (which means changing the system requirements) or to increase the capacity by using a faster computer.

Experience has shown that priority inversion (blocking) and suboptimal preemption are major sources of missed deadlines. So we focus on identifying sources of priority inversion and suboptimal preemption, and try to reduce their effects to enhance schedulability.

1.6 Short History of Real-Time Analysis

UNIVERSIDAD DE CANTABRIA

Initial paper by Liu and Layland in 1973

- introduced Rate Monotonic and EDF scheduling
- only applicable to a very restrictive case with independent periodic tasks, and deadlines=periods
- optimal priority assignments (when deadlines are at end of period)
- analytic formulas to check schedulability (utilization tests)

Exact schedulability tests (response time analysis)

- developed by Harter (1984) and Joseph and Pandya (1986)

Extended to handle arbitrary deadlines

- Lehoczky (1990)

Extended to handle input jitter

- Tindell (1994)

Extensions to basic theory

UNIVERSIDAD DE CANTABRIA

Priority inversion and task synchronization

- Immediate priority ceilings
  - Lampson and Redell (1980)
  - Baker (1991)
- shared resources with priority inheritance
  - Sha, Rajkumar, Lehoczky (1990)
- multiprocessor systems
  - Rajkumar, Sha, Lehoczky (1988), Rajkumar (1990)

Aperiodic tasks

- Sporadic server, fixed priorities
  - Sprunt, Sha, Lehoczky (1989)

Multiprocessor and distributed systems, OS issues, etc.

UNIVERSIDAD DE CANTABRIA

Holistic Analysis.

- Tindell (1994) and Tindell and Clark (1994)
- Palencia et al (1997)

Offset Based Analysis

- Tindell (1994)
- Extended to distributed systems by Palencia and González (1998)
- Optimized by Palencia and González (1999), and by Redell (2003)

Priority assignment techniques

- Rate Monotonic for deadlines equal to periods
  - Liu and Layland (1973)
- Deadline monotonic assignment for pre-period deadlines
  - Leung and Layland (1982)
- Deadlines larger than periods
  - Audsley (1991)
- Distributed systems
  - Tindell, Burns, and Wellings (1992)
  - Gutiérrez García and González Harbour (1995)

Dynamic priorities

UNIVERSIDAD DE CANTABRIA

EDF response time analysis

- Single processor systems
  - Spuri (1996)
- Distributed systems
  - Holistic analysis: Spuri (1996)
  - Offset-based analysis: Palencia and González Harbour (2003)
- Synchronization
  - SRP: Baker (1991)
- Aperiodic tasks
  - Constant bandwidth server, Abeni and Buttazzo (1998)
- Hierarchical scheduling
  - González Harbour and Palencia (2003)

Influence in Standards

UNIVERSIDAD DE CANTABRIA

IEEE POSIX standards

- Real-time extensions (fixed priorities, priority inheritance protocols)
- Advanced real-time extensions (execution-time clocks, sporadic servers)

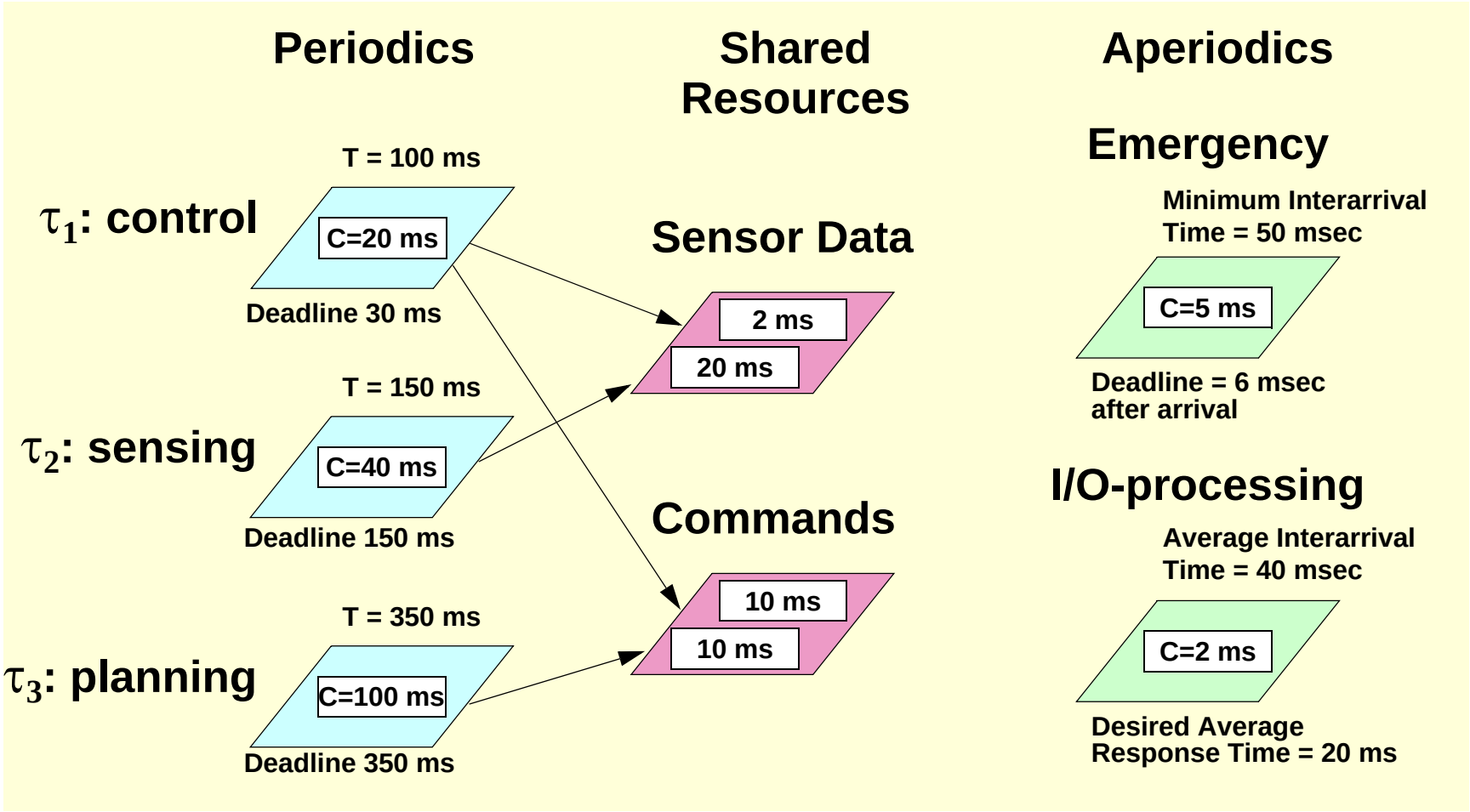
Ada

- 1983: fixed priorities
- 1995: priority ceilings in protected objects
- 2005: execution time clocks, EDF

Real-Time Java

- fixed priorities, EDF, priority inheritance, ...

1.8 A Sample Case



Notes:

We will use this sample problem throughout the tutorial as a common thread, starting with the periodic tasks, then gradually adding the other components in the task set. This sample task set is simple and yet embodies many different types of real-time requirements. The task set is completely analyzable using the methods outlined in this tutorial.

Task set characteristics:

3. Periodic task τ_1 (control): execution time = 20 msec; period = 100 msec; deadline is 30 ms after the start of each period. In addition, τ_3 may block τ_1 for 10 msec by sharing a shared object containing commands, and task τ_2 may block τ_1 for 20 msec by sharing a data object containing sensor data.
4. Periodic task τ_2: execution time = 40 msec; period = 150 msec; deadline is at the end of each period.
5. Periodic task τ_3: execution time = 100 msec; period = 350 msec; deadline is at the end of each period.
6. Emergency event handling task: execution time = 5 msec; worst-case interarrival time = 50 msec; deadline is 6 msec after arrival.
7. IO-processing event handling task: average execution time = 2 msec; average interarrival time = 40 msec; 20 msec average response time is needed.

1.9 MAST Modelling Environment Introduction

UNIVERSIDAD DE CANTABRIA

Many real-time systems are now distributed

- **Cyclic executives being replaced by run-time schedulers**
- **Fixed priority and EDF scheduling are most popular among the run-time scheduling policies**

As we have seen, schedulability analysis techniques have evolved a lot in the last decade

- **more complex**

Motivation

UNIVERSIDAD DE CANTABRIA

The latest schedulability analysis techniques are difficult to apply by hand

Need to integrate all the techniques in a single tool suite

- integration needed new theory

Few free-software schedulability analysis tools

Need to integrate soft & hard real-time analysis

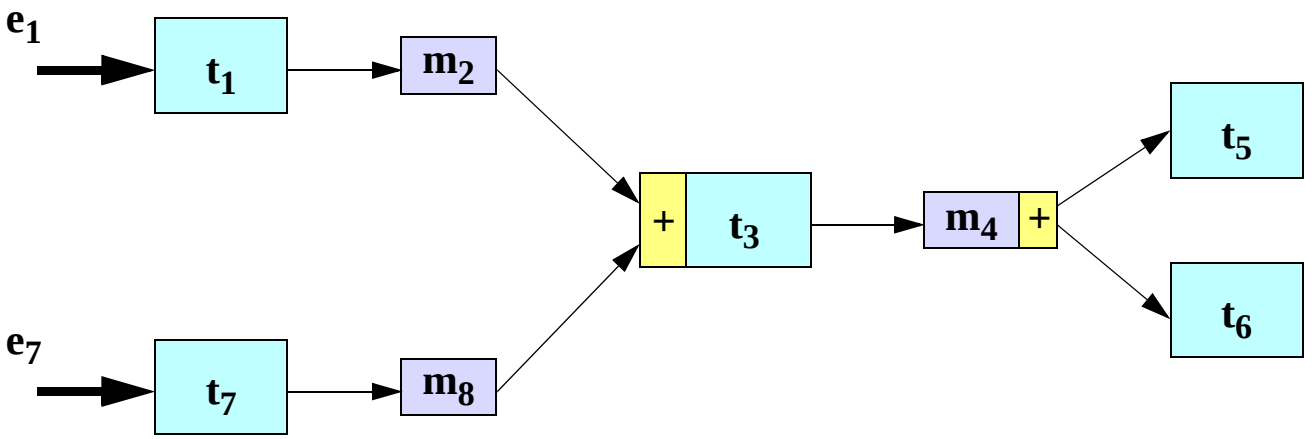
Need for an open tool that can be used as a plug-in module for other development tools

- e.g., UML tools

Motivation (cont'd)

Need for a rich and flexible model of the real-time system:

- distributed, multiprocessor, or single processor
- composable software modules
- separation of architecture, platform, and software modules
- rich set of event-driven patterns; e.g.:



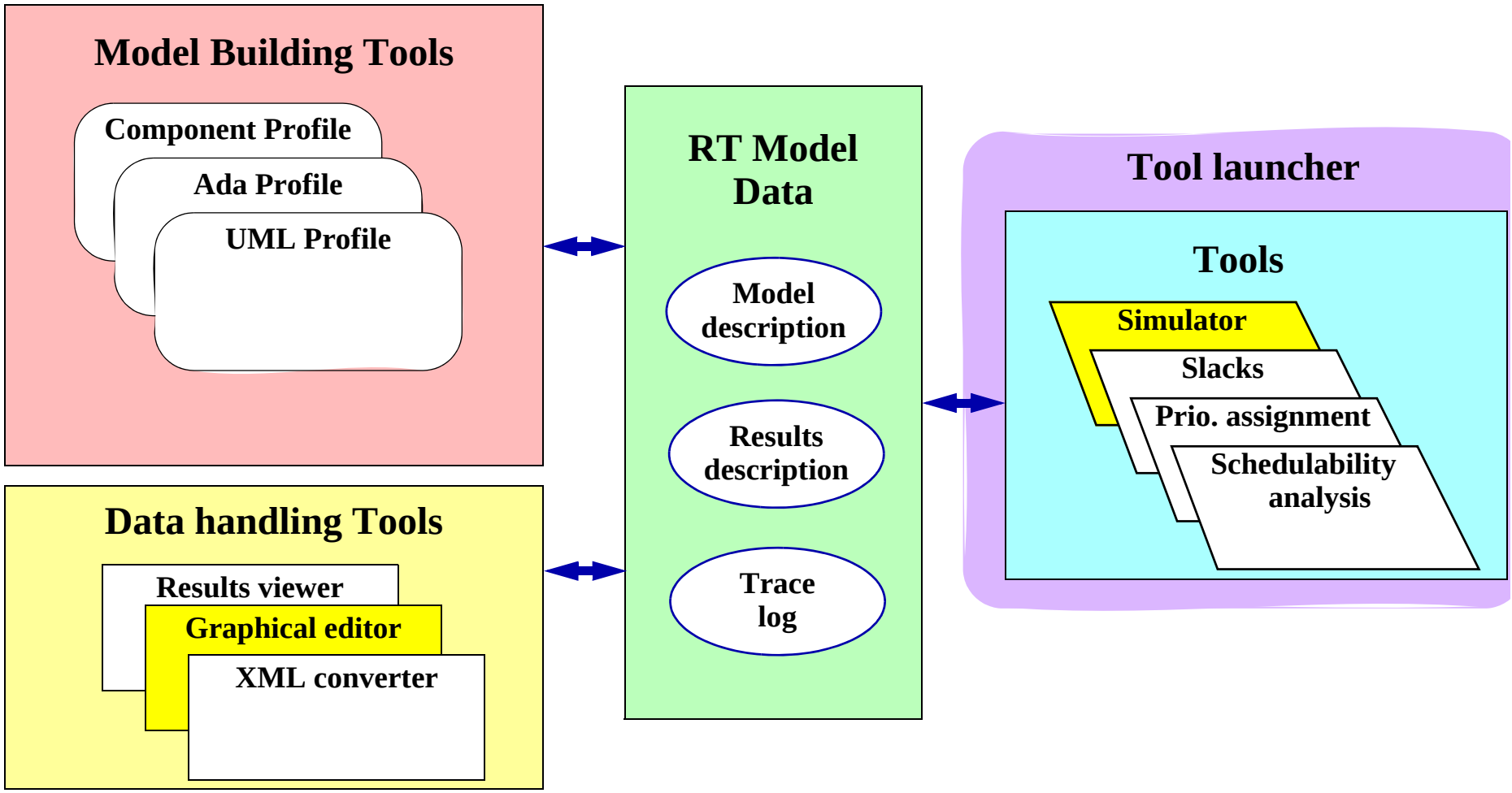
Objectives

- Develop a *model* for describing the timing behavior of event-driven distributed real-time systems
- *Open model* that may evolve to include new characteristics or points of view of the system
- Develop a set of *tools* for analyzing the timing behavior of the application:
  - Schedulability analysis (hard real-time requirements)
  - Synchronization blocking delays calculation
  - Discrete-event simulation (soft real-time)
  - Priority assignment
  - Sensitivity analysis (slacks)

Objectives (cont'd)

- Free software (GPL license)
- Textual and XML model description languages for easy integration with other tools:
  - MAST model description
  - Results of applying the tools

MAST Environment



Summary of MAST

UNIVERSIDAD DE CANTABRIA

MAST defines a model for describing real-time systems

- distributed and multiprocessor
- complex synchronization and event-driven schemes
- composable software modules
- independence of architecture, platform and modules

MAST provides an open set of tools

- hard and soft real-time analysis
- automatic blocking times, priority assignment, sensitivity analysis...

XML specification language allows easy integration with other tools (i.e., UML tools)

Integration into the design process

UNIVERSIDAD DE CANTABRIA

Components built with their own timing behavior model

- passive components: operations and shared resources
- active components: single or multithreaded, distributed, ...

The model is parameterized

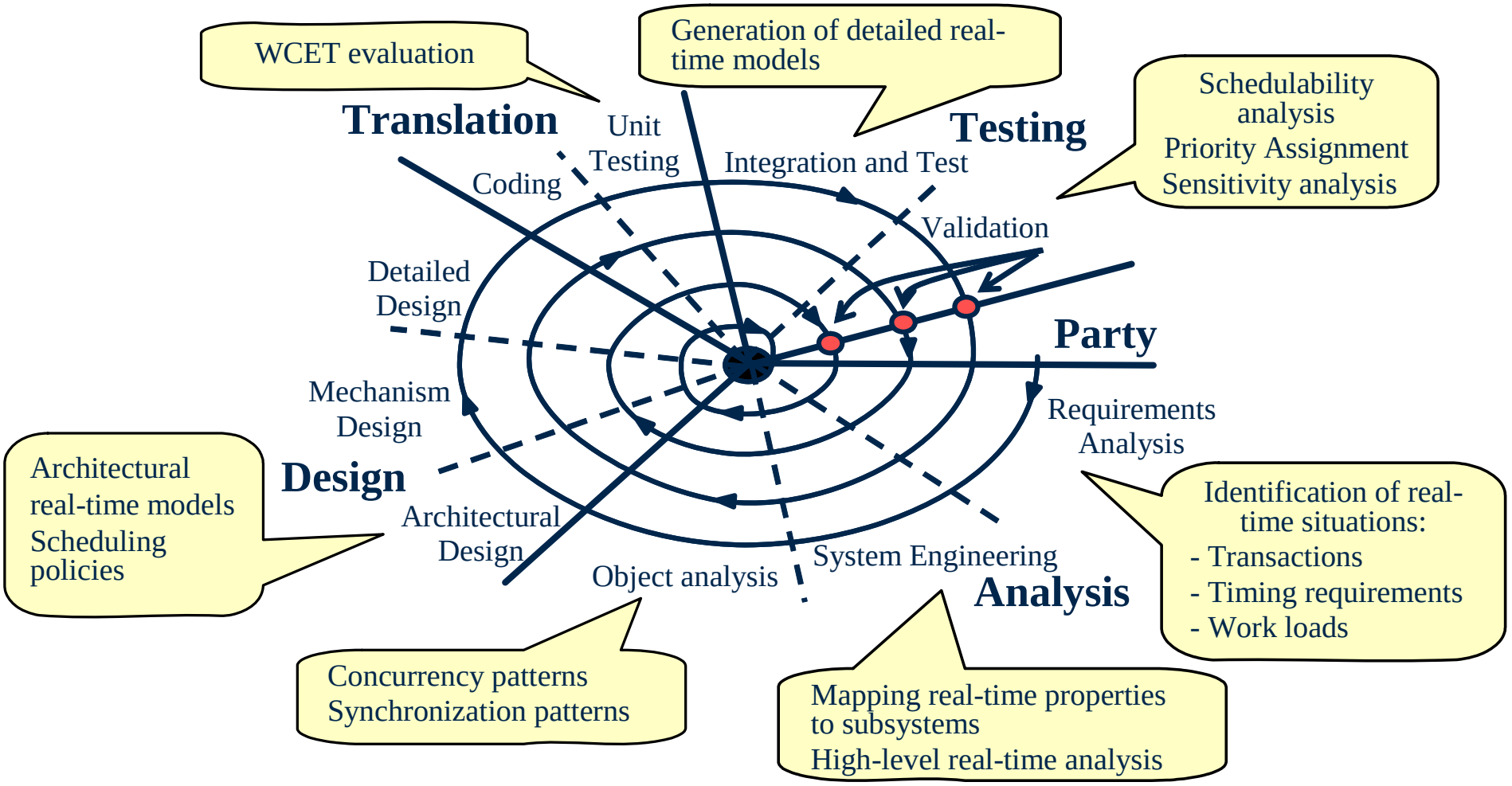
- i.e., actual data for WCETS

Deployment tool

- instantiates the parameterized component models
- provides the platform model
- integrates them with the real time situation model

Automatic schedulability analysis is then made

Integration into the design process



Future work: MAST

UNIVERSIDAD DE CANTABRIA

Finish current tools:

- graphical editor
- integrate simulator

Implement missing tools:

- Multiple-Event Analysis
- Full support for EDF

Speed up the response time analysis

URL

UNIVERSIDAD DE CANTABRIA

http://mast.unican.es

Summary of introduction

UNIVERSIDAD DE CANTABRIA

Real-time goals are: predictability, guaranteed deadlines, and stability in overload.

Real-time analysis

- **based on scheduling theory**
- **analytic formulas to determine schedulability**
- **framework for reasoning about system timing behavior**
- **separation of timing and functional concerns**

Provides an engineering basis for designing real-time systems