

Software Architecture

Bertrand Meyer

ETH Zurich, March-July 2007

Lecture 12: Embedded and real-time systems

General Programming languages

Programming languages — claiming to be **general purpose languages** — should also support (among others)

- Concurrent programming
- Real-time programming

Overview

- Motivation / Goal
- Definition of embedded and real-time systems
- Characteristics of real-time systems
- Example of a real-time system
- Real-time facilities
 - Notion of time
 - Clocks, delays and timeouts
 - Temporal scopes
- Conclusion

Definition: Embedded system

Embedded system:

The computer is an information processing component **within (embedded in)** a larger engineering system.

e.g. washing machine, process control computer, ABS, ASR, ESP, SBC in vehicles, ...

Market share ...

Over **95 % of all microprocessors** in the world are used for embedded and real-time systems

Definition: Real-time system

Real-time system (Young, 1982):

Any information processing activity or system which has to **respond** to externally generated input stimuli within a **finite** and **specified period**.

The **correctness** of a real-time system depends not only on the logical result of the computation, but also **on the time** at which the results are produced ...

→ a correct but a **late** response is as bad as a **wrong response** ...

Hard and soft real-time systems

Hard real-time

Systems where it is **absolutely imperative** that responses occur within the required deadline.
e.g. flight control systems, ...

Soft real-time

Systems where deadlines are important but which will still function correctly if deadlines are **occasionally missed**.
e.g. data acquisition system, ...

A single real-time system may have **both hard and soft** real-time subsystems

SBC – Sensotronic Brake Control

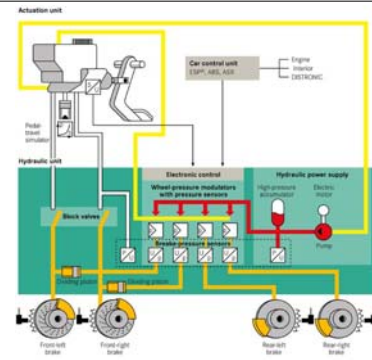


Safety systems of cars

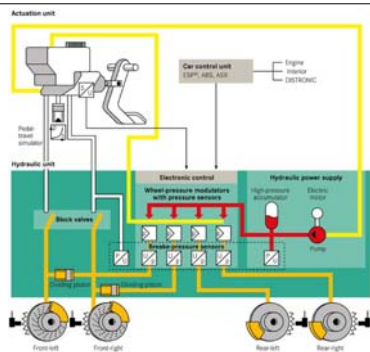
Mechatronics:
Mechanics and Electronics (+ Software)

- ABS
Anti-lock Braking System
- ASR
Anti Spin Regulation
- ESP
Electronic Stability Program

SBC – Sensotronic Brake Control



SBC – Sensotronic Brake Control



SBC functionalities

Dry Brake

Keep with short brake pulses the brake discs always dry and fully functional.

SBC Hold

A *drive-away assistant* prevents the vehicle from rolling backwards or forward when starting on a hill or steep incline.

SBC Stop

In stop-and-go traffic the car brakes automatically, when the foot is lifted off the accelerator pedal

SBC Soft Stop (not released yet)

In city traffic soft-stop allows soft, jerkless stopping

Characteristics of real-time systems

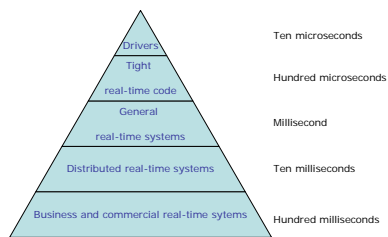
- Large and complex
up to 20 million lines of code estimated for the *Space Station Freedom*
- Concurrent control of separate system components
- Facilities to interact with special purpose hardware
- Extreme reliable and safe
- Guaranteed response times

Worst-case vs. best-case (cont.)

Real-time approach:

- Use a self-balancing binary tree
- Use a different sorting algorithm (e.g. Mergesort is slower than quicksort on average, but predictable)
- \Rightarrow Resulting average real-time performance will be slower, but its worst-case performance will be better than that of the conventional one

Precision of Measurement



What happens when a deadline is missed?

Hard real-time systems cannot tolerate late results:

- Something unrecoverable happens
e.g. SBC fails \Rightarrow car does not stop \Rightarrow accident
- Degraded mode (provide limited or in extrem cases no functionality for the failed subsystem)

Soft real-time systems can tolerate (once in a while) late results:

- Try to reproduce the result although we are already late

Worst-case vs. best-case

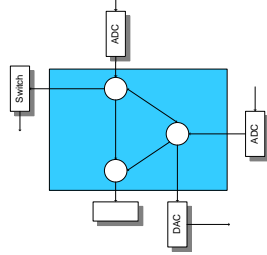
Worst-case is more important than **best-case**:

- A dynamically constructed binary tree can degenerate into a structure with linear search time
- A quicksort can take $O(n^2)$ time

Components of a real-time system

- Hardware
CPU, sensors, ADC, DAC, ...
- Real-time OS
e.g. VxWorks, QNX, Real-Time Linux, Windows CE .NET, ...
- Real-time application and real-time runtime system
e.g. assembler language, C with Real-Time Posix, Ada, Real-Time Java

A simple embedded and real-time example



Notion of time

Linearity: $\forall x, y : (x < y) \vee (y < x) \vee (x = y)$

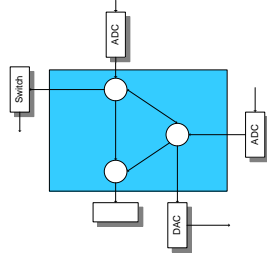
Transitivity: $\forall x, y, z : (x < y \wedge y < z) \Rightarrow x < z$

Irreflexibility: $\forall x : \text{not}(x < x)$

Density: $\forall x, y : x < y \Rightarrow \exists z : (x < z < y)$

→ The passage of time is equated with a **real line**.

A simple embedded and real-time example



Access to a Clock

- Direct access to the **environment's time frame**
e.g. transmitters for **UTC = Universal Time Coordinated**, UTC service of **GPS**
- Using an **internal hardware clock** that gives an adequate approximation to the passage of time in the environment

T

Real-Time Facilities

- Notion of time
- Clocks
- Delays
- Timeouts
- Temporal scopes

Ada: Real-time clock (1)

```
package Ada.Real_Time is
  type Time is private;
  Time_First: constant Time;
  Time_Last: constant Time;
  Time_Unit: constant := -- smallest amount of real time representable by the Time type;

  type Time_Span is private;
  Time_Span_First: constant Time_Span;
  Time_Span_Last: constant Time_Span;
  Time_Span_Zero: constant Time_Span;
  Time_Span_Unit: constant Time_Span;

  Tick: constant Time_Span; -- value of Tick must be no greater than 1 millisecond
  function Clock return Time; -- range of Time must be at least 50 years
  function "+" (Left: Time; Right: Time_Span) return Time;
  function "+" (Left: Time_Span; Right: Time) return Time;
  -- similarly for "-", "<", etc
end package;
```

DAC = Digital to analogue converter

X separate object

P

enter

Ada: Real-time clock (2)

```
function To_Duration(TS: Time_Span) return Duration;
function To_Time_Span(D: Duration) return Time_Span;

function Nanoseconds (NS: Integer) return Time_Span;
function Microseconds(US: Integer) return Time_Span;
function Milliseconds(MS: Integer) return Time_Span;
type Seconds_Count is range implementation-defined;
procedure Split(T : in Time; SC: out Seconds_Count;
               TS : out Time_Span);

function Time_Of(SC: Seconds_Count;
                TS: Time_Span) return Time;

private
  -- not specified by the language
end Ada.Real_Time;
```

RT Java Time Types (1)

```
public abstract class HighResolutionTime implements
    java.lang.Comparable
{
    public abstract AbsoluteTime absolute(Clock clock,
        AbsoluteTime destination);

    ...

    public boolean equals(HighResolutionTime time);

    public final long getMilliseconds();
    public final int getNanoseconds();

    public void set(HighResolutionTime time);
    public void set(long millis);
    public void set(long millis, int nanos);
}
```

Example: Timing a sequence in Ada

```
declare
  use Ada.Real_Time;
  Start, Finish : Time;
  Interval : Time_Span := To_Time_Span(1.7);
begin
  Start := Clock;
  -- sequence of statements
  Finish := Clock;
  if Finish - Start > Interval then
    raise Time_Error; -- a user-defined exception
  end if;
end;
```

RT Java Time Types (2)

```
public class AbsoluteTime extends HighResolutionTime
{
    // various constructor methods including
    public AbsoluteTime(AbsoluteTime T);
    public AbsoluteTime(long millis, int nanos);

    public AbsoluteTime absolute(Clock clock, AbsoluteTime dest);

    public AbsoluteTime add(long millis, int nanos);
    public final AbsoluteTime add(RelativeTime time);

    ...

    public final RelativeTime subtract(AbsoluteTime time);
    public final AbsoluteTime subtract(RelativeTime time);
}
```

Clocks in Real-Time Java

- Similar to those in Ada
- `java.lang.System.currentTimeMillis` returns the number of milliseconds since 1/1/1970 GMT and is used by `java.util.Date`
- Real-time Java adds real-time clocks with high resolution time types

RT Java Time Types (3)

```
public class RelativeTime extends HighResolutionTime
{
    // various constructor methods including
    public RelativeTime(long millis, int nanos);
    public RelativeTime(RelativeTime time);

    public AbsoluteTime absolute(Clock clock,
        AbsoluteTime destination);

    public RelativeTime add(long millis, int nanos);
    public final RelativeTime add(RelativeTime time);

    public void addInterarrivalTo(AbsoluteTime destination);

    public final RelativeTime subtract(RelativeTime time);

    ...
}

public class RationalTime extends RelativeTime
{ . . . }
```

RT Java: Clock Class

```
public abstract class Clock
{
    public Clock();

    public static Clock getRealtimeClock();

    public abstract RelativeTime getResolution();

    public AbsoluteTime getTime();
    public abstract void setTime(AbsoluteTime time);

    public abstract void setResolution(RelativeTime resolution);
}
```

POSIX Real-Time Clocks

```
#define CLOCK_REALTIME ...; // clockid_t type

struct timespec {
    time_t tv_sec; /* number of seconds */
    long tv_nsec; /* number of nanoseconds */
};
typedef ... clockid_t;

int clock_gettime(clockid_t clock_id, struct timespec *tp);
int clock_settime(clockid_t clock_id, const struct timespec *tp);
int clock_getres(clockid_t clock_id, struct timespec *res);

int clock_getcpuclockid(pid_t pid, clockid_t *clock_id);
int clock_getcpuclockid(pthread_t_t thread_id, clockid_t *clock_id);

int nanosleep(const struct timespec *rqtp, struct timespec *rmtp);
/* nanosleep return -1 if the sleep is interrupted by a */
/* signal. In this case, rmtp has the remaining sleep time */
```

RT Java: Measuring Time

```
{
    AbsoluteTime oldTime, newTime;
    RelativeTime interval;
    Clock clock = Clock.getRealtimeClock();

    oldTime = clock.getTime();
    // other computations
    newTime = clock.getTime();

    interval = newTime.subtract(oldTime);
}
```

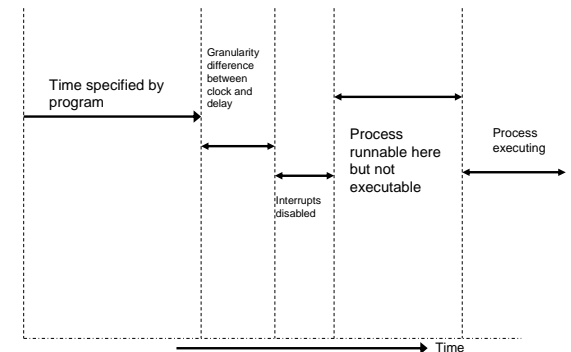
Delaying a Process (thread)

- The execution of a process (thread) must be sometimes delayed either for a relative period of time or until some time in the future
- Relative delays
Start := Clock; -- from calendar
loop
 exit when (Clock - Start) > 10.0;
end loop;
- Busy-waits are not efficient, therefore most languages and operating systems provide some form of delay primitive
- In Ada, this is a delay statement
delay 10.0;
- In POSIX: sleep and nanosleep
- Java: sleep; RT Java provides a high resolution sleep

Clocks in C and POSIX

- ANSI C has a standard library for interfacing to calendar time
- This defines a basic time type *time_t* and several routines for manipulating objects of type *time*
- POSIX requires at least one clock of minimum resolution 50 Hz (20ms)

Delays



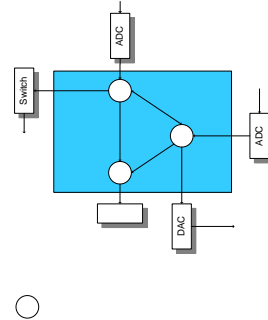
Absolute Delays

- In Ada

```
Start := Clock;
First_action;
delay 10.0 - (Clock - Start);
Second_action;
```
- Unfortunately, this might not achieve the desired result, therefore we use:

```
Start := Clock;
First_action;
delay until Start + 10.0;
Second_action;
```
- As with `delay`, `delay until` is accurate only in its lower bound
- RT Java - sleep can be relative or absolute
- POSIX requires use of an absolute timer and signal

A simple embedded and real-time example



Drifts

- The time overrun associated with both relative and absolute delays is called the *local drift* and it **cannot** be eliminated
- It is possible, however, to eliminate the *cumulative drift* that could arise if *local drifts* were allowed to superimpose

Control Example in Ada (1)

```
with Ada.Real_Time; use Ada.Real_Time;
with Data_Types; use Data_Types;
with IO; use IO;
with Control_Procedures;
use Control_Procedures;
procedure Controller is
```

```
task Temp_Controller;

task Pressure_Controller;
```

Periodic Activity

```
task body T is
  Interval : constant Duration := 5.0;
  Next_Time : Time;
begin
  Next_Time := Clock + Interval;
  loop
    Action;
    delay until Next_Time;
    Next_Time := Next_Time + Interval;
  end loop;
end T;
```

If Action takes 6 ms, the delay statement will have no effect

Will run on average every 5 ms
local drift only

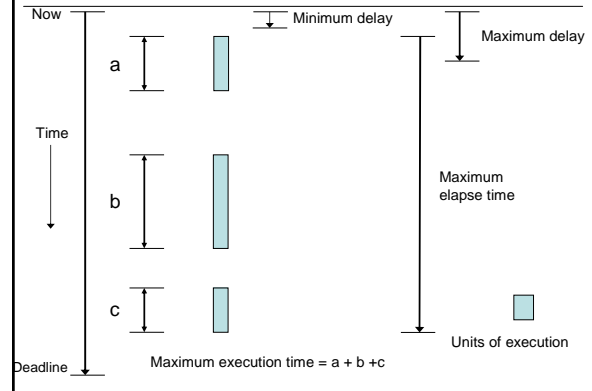
Control Example in Ada (2)

```
task body Temp_Controller is
  TR : Temp_Reading; HS : Heater_Setting;
  Next : Time;
  Interval : Time_Span := Milliseconds(200);
begin
  Next := Clock; -- start time
  loop
    Read(TR);
    Temp_Convert(TR,HS);
    Write(HS);
    Next := Next + Interval;
    delay until Next;
  end loop;
end Temp_Controller;
```

Control Example in Ada (3)

```
task body Pressure_Controller is
  PR : Pressure_Reading; PS : Pressure_Setting;
  Next : Time;
  Interval : Time_Span := Milliseconds(150);
begin
  Next := Clock; -- start time
  loop
    Read(PR);
    Pressure_Convert(PR,PS);
    Write(PS);
    Next := Next + Interval;
    delay until Next;
  end loop;
end Pressure_Controller;
begin
  null;
end Controller;
```

Temporal Scopes (2)



Timeouts on Actions

```
select
  delay 0.1;
then abort
  -- action
end select;
```

- If **action** takes too long (more than 100 ms), the **action** will be aborted
- Java supports timeouts through the class *Timed*.

Specifying Processes and TS

```
process periodic_P;
...
begin
  loop
    IDLE
    start of temporal scope
    ...
    end of temporal scope
  end;
end;
```

Time constraints:

- maximum and/or minimum times for IDLE
- At the end of the temporal scope a **deadline** must be met

Temporal Scopes (1)

Temporal scope:

Collection of statements with an associated timing constraint

- **Deadline** — the time by which the execution of a TS must be finished
- **Minimum delay** — the minimum amount of time that must elapse before the start of execution of a TS
- **Maximum delay** — the maximum amount of time that can elapse before the start of execution of a TS
- **Maximum execution time** — of a TS
- **Maximum elapse time** — of a TS

Temporal scopes with **combinations** of these attributes are also possible

Deadline

The **deadline** can itself be expressed in terms of either

- **absolute time**
- **execution time** since the start of the temporal scope, or
- **elapsed time** since the start of the temporal scope.

Aperiodic Processes

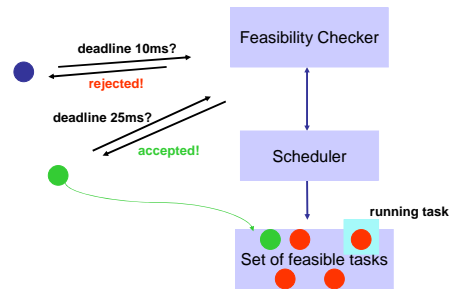
Aperiodic temporal scopes usually arise from asynchronous events (outside of the embedded system)

```

process aperiodic_P;
...
begin
  loop
    wait for interrupt
    start of temporal scope
    ...
    end of temporal scope
  end;
end;

```

Dynamic scheduling



Real-time tasks

- Tasks (processes, threads) repeatedly processing jobs
 - **Periodic tasks:**
 - jobs released at a **fixed period**
 - PeriodicTask = (WCET, Period, Deadline)
 - **Sporadic tasks:**
 - jobs released arbitrarily, but with a guaranteed **minimum inter-arrival time**
 - SporadicTask = (WCET, MinInterarrivalTime, Deadline)
- Non-deterministically triggered jobs
 - > **Aperiodic tasks:**
 - job release times cannot be predicted
 - AperiodicTask = (WCET, Deadline)

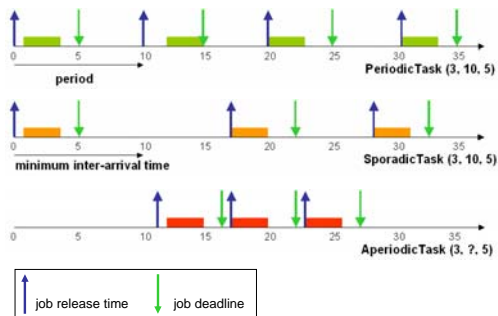
Conclusion

Only basic facilities of real-time systems covered

Not covered:

- Languages for temporal scope (e.g. Real-Time Euclid, Pearl, Esterel ...)
- Fault tolerance
- Scheduling and priorities

Real-time tasks



References

- Simon D., An Embedded Software Primer, 3rd printing, Addison Wesley, 2000
- Burns A., Wellings A., Real Time Systems and Programming Languages, 3rd edition, Addison Wesley, 2001
- Dibble P., Real Time Java Platform Programming, 1st edition, Sun Microsystems Press (A Prentice Hall Title), 2002