380

Real-Time & Embedded Systems 2019

4

Time & Space

Uwe R. Zimmer - The Australian National University

381

References

[Burns2009]
Alan Burns and Andy Wellings;
Real-Time Systems and Programming Languages;
Addison Wesley, fourth edition 2009

[Dibble2009]
Dibble, Peter
RTSJ 1.1 - JSR-282 Early Draft Review version 6
2009 pp. 1-18

[Dourish2001]
Dourish, Paul
Where the Action Is
MIT Press, Cambridge, Massachusetts, London, England 2001

[AdaRM2012]
Ada Reference Manual - Language and Standard Libraries;
ISO/IEC 8652:201x (E)

382

Notions of time and space

The big topics:

What is time? / What is embodiment?

☞ **Interfacing** *with time*

☞ **Specifying** *timing requirements*

☞ **Satisfying** *timing requirements*

383

Notions of time and space

What is time?

☞ Do we exist in time, or is time part of our existence?

Is time an intrinsic property of nature? ☞ Platonism:

Time is an **external phenomenon**. Thus simultaneous events happen at the same exact absolute time. There is the underlying assumption that time is progressing, even if no changes can be observed.

Is time a construct which is based on observable events? ☞ Reductionism:

Time is the observation of **distinguishable events**. If the observed events are 'regular', a useful time-reference can be constructed. If all possible observers detect one event before another one, they are said to be in sequence. If this cannot be assumed for all possible observers, they are said to be simultaneous. Therefore the notion of time is reduced to a notion of **causality**.

☞ Can time be interpolated between observable events?

Time & Space

Notions of time and space

What is time?

A mathematical notion of time:

- Transitivity: $x < y \wedge y < z \Rightarrow x < z$
- Linearity: $x < y \vee y < x \Rightarrow x \neq y$
- Irreflexivity: $\neg(x < x)$
- Density: $x < y \Rightarrow \exists z \mid x < z \wedge z < y$
- Continuity: $x < y \Rightarrow \forall z \mid x + z < y + z$

Time & Space

Notions of time and space

What is time?

A mathematical notion of time:

- Transitivity: $x < y \wedge y < z \Rightarrow x < z$
- Linearity: $x < y \vee y < x \Rightarrow x \neq y$
- Irreflexivity: $\neg(x < x)$
- Density: $x < y \Rightarrow \exists z \mid x < z \wedge z < y$
- Continuity: $x < y \Rightarrow \forall z \mid x + z < y + z$

Real clocks have limited resolution.

Real clocks run asynchronously.

Time & Space

Notions of time and space

What is time?

A physical notion of time:

- **1676: Rømer proofs the existence of the speed of light** (and measures it).
- **1687: Newton's "Principia Mathematica" assumes an absolute time**, independent of space itself and independent of events.
- **1905: The concept of absolute time is destroyed** first by Einstein and (a few weeks later) by Poincaré.
- **1915: Einstein's general theory of relativity eliminates the independence of time (space) and events in time (space)**.

Time & Space

Notions of time and space

What is time?

A physical notion of time:

- **1676: Rømer proofs the existence of the speed of light** (and measures it).
- **1687: Newton's "Principia Mathematica" assumes an absolute time**, independent of space itself and independent of events.
- **1905: The concept of absolute time is destroyed** first by Einstein and (a few weeks later) by Poincaré.
- **1915: Einstein's general theory of relativity eliminates the independence of time (space) and events in time (space)**.

☞ One principal consequence for measurements of time:

Clocks under higher gravity or in faster observation frames are slower

Practical consequences: clocks in satellites need to be adjusted accordingly.

Notions of time and space

What time is it?

Moon calendars	since about 4000 BC
Sundials and water clocks	since about 1500 BC
Solar calenders	since about 50 BC
Spring based clocks	since about the 13th century
Gregorian calendar	1582: corrections to previous solar calenders
Pendulum clocks	since 1656
Greenwich Mean Time GMT	1675: Royal Greenwich observatory founded
Universal time UT0	1884: Mean solar time at Greenwich meridian
Quartz crystal clocks	since 1927
Universal time UT1	Continuously updated corrections to UT0, ☞ polar motion
Universal time UT2	Continuously updated corrections to UT1, ☞ variations in the speed of rotation of the earth

Notions of time and space

What time is it?

International Atomic Time TAI	1955: A caesium 133 atomic clock (current accuracies: one miss in 10^{13} ticks, e.g. approximately once every 300,000 years)
Ephemeris Time ET0	1961: based on observing the motion of the solar system.
Universal Time Coordinated UTC	since 1964: An TAI based clock which is synchronized to UT1 (by introducing occasional leap ticks). $\lvert UTC - IAT \rvert < 0.5s$
Ephemeris Time ET1	1967: corrections to the ET0 model
Ephemeris Time ET2	1972: corrections to the ET1 model
'1 second'	• 1/86,400 of a mean solar day. • 1/31,566,925.9747 of the tropical year for 1900 (*Ephemeris Time* as defined in 1955). • 9,192,631,770 periods of the radiation corresponding to the transition between two hyper-fine levels of the ground state of the caesium 133 atom.

Notions of time and space

What time is it?

☞ How real is real-time?

"Real-time" clock ☞ usually understood as an external time source.

Real-time systems as an engineering discipline:

☞ It is usually of no importance *how* time is defined or interpreted.
☞ It is important to define *which* accessible reference time is denoted "real-time".

"Real-time" in engineering is the notion of an actual, accessible, generated or external, measured-by-events reference time.

Notions of time and space

What time is it?

☞ Which time frames can be used as real-time?

☞ Generating a time frame
- by any local timer generating a regular interrupt.
- by employing a RTC-module (a timer based on the notion of seconds).

☞ Using an existing time-frame
- by employing time-stamps or sequence numbers of received sensor-readings.
- by a radio receiver for UTC or TAI (available in some countries).

*Common programming languages guarantee a '**resolution**' and '**accuracy**' of time*
(in reference to 'a second') … not its origin or meaning.

Time & Space

Notions of time and space

The big topics:

What is time? / What is embodiment?

☞ **Interfacing** ***with time***

☞ **Specifying** ***timing requirements***

☞ **Satisfying** ***timing requirements***

Time & Space

Notions of time and space

What is embodiment?

Working hypothesis:

Embodied phenomena are those that by their very nature occur in real time and real space.

… to be refined …

Time & Space

Notions of time and space

What is embodiment?

Phenomenology

The phenomena of experience as the central aspects and building blocks of understanding.

(rather than: "Finding the 'truth'")

Applied to and trying to combine aspects of:

- Ontology (about the nature of being and categories of existence)
- Epistemology (the study of knowledge)

Time & Space

Notions of time and space

What is embodiment?

Edmund Husserl (1859-1938, Vienna, Halle, Göttingen, Freiburg):

- Founder of the phenomenological tradition
  … as a trial to establish modern science which is firmly grounded on the phenomena of experience (instead of being an abstract mathematical construct).
- Phenomenology originally as a method to examine the nature of intentionality.
- Coined the terms
  - *Noema*: the objects of consciousness.
  - *Noesis*: the mental experiences of those objects.
  - *Lebenswelt* (*life-world*): the inter-subjective world of everyday-experience.

☞ Husserl rejected pure abstract and formalized reasoning.

396

Time & Space

Notions of time and space

What is embodiment?

Martin Heidegger (1889-1976, Freiburg):

- Moved phenomenology from a discussion about mental phenomena separated from the physical world (Cartesian dualism) to a discussion about connected physical and mental phenomena.
- Moved the central questions from epistemology to ontology ('Being and Time', 1927):

The meaning is not 'in the head' but in the world!

- Coined terms:
  - *Dasein* (*being-in-the-world*): 'being as inseparable from the world in which it occurs'
    ☞ being as always purposeful and active
    ☞ the world as an unconscious but accessible background
  - *Zuhanden* (*ready-to-hand*): 'equipment as a part of actual interaction with the world'
  - *Vorhanden* (*present-at-hand*): 'equipment as a conscious model'

397

Time & Space

Notions of time and space

What is embodiment?

Maurice Merleau-Ponty (1908-1961, Paris (Sorbonne)):

- 'The Phenomenology of Perception' (1945)
- Embodiment has three implications:
  - a body as a *physical entity*.
  - a body as a *set of physical skills and situated responses* gained from the physical world.
  - a body as a *set of 'cultural skills'* gained from the cultural world in which it is embedded.
- Embodied perception as a bi-directional sensation and a basis for empathy (perception in itself does not exist).

☞ see also: Phenomenology of Jean-Paul Sartre.

Recent works in robotics (and insights about biological sensors) blur the line between action and perception even further.

398

Time & Space

Notions of time and space

What is embodiment?

Working hypothesis:

Embodied phenomena are those that by their very nature occur in real time and real space.

Refinement:

Embodiment is the property of any engagement with the real world which (may) makes this engagement meaningful.

(Paul Dourish)

☞ Embodied phenomena are the essence of meaningful interaction

(Real-time and embedded systems are the technical instantiations of embodiment)

399

Time & Space

Notions of time and space

What is embodiment?

Embodiment is the property of any engagement with the real world which (may) makes this engagement meaningful.

Implications:

There is no such thing as

Universal 'intelligence', **'autonomy'**, **'conscience'** or any other cognitive process which is independent of a physical environments.

There is no such thing as a

Universal morphology (mechanical design, robot, device, …) which is useful or even operational in all physical environments.

400

Time & Space

Notions of time and space

What is embodiment?

Embodiment is the property of any engagement with the real world which (may) makes this engagement meaningful.

> An actual or potential existence of a 'universal intelligence' or a 'universal robot' can of course not be disproved.

Implications:

There is no such thing as

Universal 'intelligence', **'autonomy'**, **'conscience'** or any other cognitive process which is independent of a physical environments.

There is no such thing as a

Universal morphology (mechanical design, robot, device, …) which is useful or even operational in all physical environments.

401

Time & Space

Notions of time and space

What is embodiment?

Meaningfully embedded systems are part of an 'ecological niche'
(Rolf Pfeifer)

- **The operational environment** is supportive and employed by the system.
- **The embedded system** is constructed as a part of the operational environment and according to the task.
- **The task** is meaningful considering the morphology and cognitive ability of the system as well as the response from the environment.

☞ Meaningful embedded systems have a **purpose** and are: **situated**, **embodied**, and **self-sufficient**.

402

Time & Space

Notions of time and space

What is embodiment?

☞ Embodied skills as part of a meaningful embedded system thus depend on

A tight coupling between perception and action
… up to the level where the distinction between both can become difficult.

This requires:

To operate under real-time constraints ☞ *Real-time systems*

To construct meaningful morphologies ☞ *Embedded system*

403

Time & Space

Notions of time and space

The big topics:

What is time? / What is embodiment?

☞ **Interfacing *with time***

☞ **Specifying *timing requirements***

☞ **Satisfying *timing requirements***

Time & Space

Interfacing with time

What time is it in …

	Syntactical resolution	Required range	Required resolution	Actual resolution detectable
Java	ms	undefined	undefined	no
RT Java	ms, ns	undefined	undefined	yes
Ada	ms, µs, ns	> 50 years	< 1 ms	yes
POSIX	ms, ns	undefined	< 20 ms	yes
C	int (as seconds)	undefined	undefined	no
Hardware timers	$1/f_t$ seconds typically 10 ns … 1 µs	$2^n/f_t$ seconds typically 100 ms … 100's years	configurable	yes

Time & Space

Notions of time in Ada

Ada.Real-Time *package*

```
package Ada.Real_Time is
   type Time is private;
   Time_First : constant Time;
   Time_Last  : constant Time;
   Time_Unit  : constant := 10#1.0#E-9; -- ns

   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; -- actual clock resolution < 1 ms

   function Clock return Time;

   (… operations on and conversions with time and time_span …)
end Ada.Real_time;
```


Time & Space

Notions of time in Ada

Ada.Real-Time.Timing_Events *package*

```
package Ada.Real_Time.Timing_Events is
   type Timing_Event is tagged limited private;
   type Timing_Event_Handler is
      access protected procedure (Event : in out Timing_Event);
   procedure Set_Handler (Event   : in out Timing_Event;
                          At_Time : in Time;
                          Handler : in Timing_Event_Handler);
   procedure Set_Handler (Event   : in out Timing_Event;
                          In_Time : in Time_Span;
                          Handler : in Timing_Event_Handler);
   function Current_Handler (Event : Timing_Event) return Timing_Event_Handler;
   procedure Cancel_Handler (Event     : in out Timing_Event;
                             Cancelled : out Boolean);
   function Time_Of_Event (Event : Timing_Event) return Time;
 private ... -- not specified by the language
 end Ada.Real_Time.Timing_Events;
```


Time & Space

Notions of time in RT-Java

RT-Java time classes

Time root class:
`public abstract class HighResolutionTime implements java.lang.Comparable`
direct known subclasses:
`AbsoluteTime, RelativeTime, RationalTime`

- Similar to Ada.Real-Time, but no mandatory accuracy.
- Adds the concept of frequency ('rational time'), but does not guarantee for equidistant instantiations.

Clock Class:

```
public abstract class Clock
{
   public static   Clock        getRealtimeClock ();
   public abstract RelativeTime getResolution    ();
   public abstract void         setResolution    (RelativeTime resolution);
}
```

408

Time & Space

Notions of time in RT-Java

RT-Java timer classes

```
public abstract class Timer extends AsyncEvent

   protected Timer (HighResolutionTime time,
                    Clock clock,
                    AsyncEventHandler handler)

public class OneShotTimer extends Timer

   public OneShotTimer (HighResolutionTime time,
                        Clock clock,
                        AsyncEventHandler handler)

public class PeriodicTimer extends Timer

   public PeriodicTimer (HighResolutionTime start,
                         RelativeTime interval,
                         Clock clock,
                         AsyncEventHandler handler)
```


409

Time & Space

Notions of time in POSIX

Real-time clock interface in POSIX

```
#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 */
```

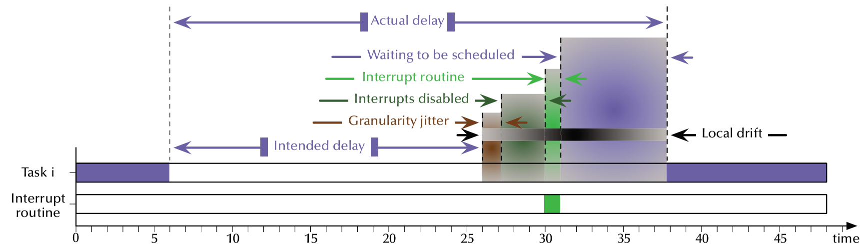

410

Time & Space

Interfacing with time

Programming primitive 'Delay'

Semantic: **Suspend** the current task **immediately** and **for (at least)** a **predefined time span** or **until (at least)** a **predefined absolute time**.



"Local delay drift" summarizes all additional (unintended) delays.

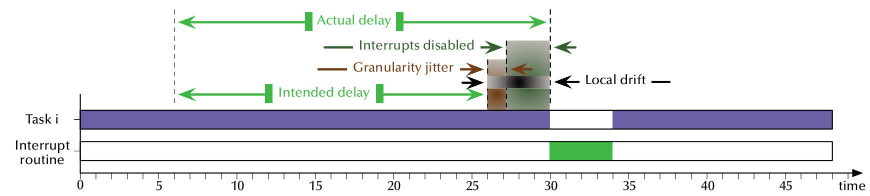
411

Time & Space

Interfacing with time

Programming primitive 'Timer'

Semantic: **Activate** a specified routine **after** a **pre-defined time span** or **at** a **predefined absolute time**.



"Local delay drift" summarizes all additional (unintended) delays.

Time & Space

Interfacing with time

Relative delay

(Ada)

```
task T;
task body T is
   begin
      loop
         Action;
         delay 5.0; -- sec.
      end loop;
   end T;
```

This loop will delay for **at least** 5 seconds.
☞ Local and cumulative drift!

Time & Space

Interfacing with time

Absolute delay

(Ada)

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

This loop will delay **on average** for 5 seconds.
☞ Local drift only!

Note that this also holds, if `Action` is sporadically (yet not always) longer than 5 seconds.

Time & Space

Interfacing with time

Absolute delay implemented by relative delay?

(Ada)

```
task body T is
   Interval   : constant Duration:= 5.0; -- sec.
   Start_Time : Time;
   begin
      loop
         Start_Time := Clock
         Action;
         delay Interval - (Clock - Start_Time);
      end loop;
   end T;
```


Time & Space

Interfacing with time

Absolute delay implemented by relative delay?

(Ada)

```
task body T is
   Interval   : constant Duration:= 5.0; -- sec.
   Start_Time : Time;
   begin
      loop
         Start_Time := Clock
         Action;
         delay Interval - (Clock - Start_Time);
      end loop;
   end T;
```

don't do it

☞ Delay time calculation is **not atomic**!

Time & Space

Interfacing with time

Zero delay

(Ada)

```
task T;
task body T is
   begin
      loop
         Action;
         delay 0.0; -- sec.
      end loop;
   end T;
```

Allows explicitly for a task switch

`delay` statements activate the scheduler.

☞ `delay 0.0` allows the scheduler to activate a runnable task of at least the same priority.
(Real-time systems do not schedule in a time-slicing fashion, but switch tasks on events only.)

Time & Space

Interfacing with time

Ada.Dispatching.Yield

(Ada)

```
task T;
task body T is
   begin
      loop
         Action;
         Yield;
      end loop;
   end T;
```

`Yield` is the same thing in a nicer word.

`Yield` statements activate the scheduler.

☞ `Yield` statements allow the scheduler to activate a runnable task of at least the same priority.
(Real-time systems do not schedule in a time-slicing fashion, but switch tasks on events only.)

Time & Space

Interfacing with time

Ada.Dispatching.Non_Preemptive.Yield_To_Higher

(Ada)

```
task T;
task body T is
   begin
      loop
         Action;
         Yield_To_Higher;
      end loop;
   end T;
```

`Yield_To_Higher` indicates that the task is willing to suspend *only if* there is a runnable, higher priority task.

`Yield_To_Higher` statements activate the scheduler.

☞ Applies to non-preemptive schedulers only
– preemptive schedulers would switch to a runnable, higher priority task anyway.

Time & Space

Interfacing with time

Relative regular timer

(Ada)

```
protected Event_Handlers is
   procedure Timer_Routine (Event : in out Timing_Event) ;
private
   Interval : constant Duration := 5.0; -- sec.
end Event_Handlers;
protected body Event_Handlers is
   procedure Timer_Routine (Event : in out Timing_Event) is
      begin
         Action;
         Set_Handler (Event, Interval, Timer_Routine'access);
      end Timer_Routine;
end Event_Handlers;
```

`Timer_Routine` will activate after **at least** 5 seconds.
☞ Local and cumulative drift!

Time & Space

Interfacing with time

Absolute regular timer

(Ada)

```
protected Event_Handlers is
   procedure Timer_Routine (Event : in out Timing_Event) ;
private
   Interval  : constant Duration := 5.0; -- sec.
   Next_Time : Time              := Clock + Interval;
end Event_Handlers;
protected body Event_Handlers is
   procedure Timer_Routine (Event : in out Timing_Event) is
      begin
         Action;
         Next_Time:= Next_Time + Interval;
         Set_Handler (Event, Next_Time, Timer_Routine'access);
      end Timer_Routine;
end Event_Handlers;
```

Timer_Routine will activate **on average** every 5 seconds.
☞ Local drift only!

☞ If executed in a real-time system, the drift will be significantly smaller compared to delay statements.

Time & Space

Interfacing with time

Delay and timers

Absolute & relative delays and timers are available in:
Real-Time Java, Pearl, Ada, … and many other real-time oriented languages.

Only absolute delays and timers are available in *strict real-time systems*:
Occam2, Ada (Ravenscar profile), …

Only relative delays or timers are available in *'low-level' environments*:
POSIX: nanosleep (absolute delays need to be constructed by employing timers and signals as well as setting interrupt masks).

Time & Space

Interfacing with time

Timeouts

As a third alternative to busy-waiting and infinite blocking, timeouts are implemented in:

- Shared variable communications
  - Semaphore
  - Conditional critical regions
  - Monitors
  - Protected objects
- Message passing between processes
  - Asynchronous and synchronous message transfers
  - Remote procedure calls
  - Remote objects
- Actions

Time & Space

Interfacing with time

Timeouts on semaphores

(POSIX)

Suspend current process until the semaphore call is open or the relative timeout has passed:

```
if (sem_timedwait (&call, &timeout) < 0) { /* something went bad */
   if (errno == ETIMEDOUT) { /* let's check a global variable */
      /* timeout occurred, try something else */
   }
   else {
      /* some other error occurred - panic a little */
   }
}
else {
   /* semaphore is locked successfully, go ahead */
};
```

Around this line you shall start praying for a thread-safe errno

Time & Space

Interfacing with time

Timeouts on semaphores

(POSIX)

Suspend current process until the semaphore call is open or the relative timeout has passed:

```
if (sem_timedwait (&call, &timeout) < 0) { /* something went bad */

   if (errno == ETIMEDOUT) { /* let's check a global variable */
      /* timeout occurred, try something else */
   }
   else {
      /* some other error occurred - panic a little */
   }
 }
 else {
    /* semaphore is locked successfully, go ahead */
};
```

Exception handling is nice isn't it?

Time & Space

Interfacing with time

Relative timeouts on entry calls

(works for task-entry calls and protected object calls)

```
task body Sensor is
T : Temperature;
begin
   loop
      -- find temperature T somewhere
      select
         Controller.Call (T);
      or
         delay 0.5; -- seconds
         -- action if temperature could not be delivered in time
      end select;
   end loop;
end Sensor;
```

Try calling for 500 ms

Time & Space

Interfacing with time

Absolute timeouts on entry calls

(works for task-entry calls and protected object calls)

```
task body Sensor is
T : Temperature;
begin
   loop
      -- find temperature T somewhere
      select
         Controller.Call (T);
      or
         delay until Deadline;
         -- action if temperature could not be delivered in time
      end select;
   end loop;
end Sensor;
```

Try calling until an absolute time

Time & Space

Interfacing with time

Non-blocking entry calls

(works for task-entry calls and protected object calls)

```
task body Sensor is
T : Temperature;
begin
   loop
      -- find temperature T somewhere
      select
         Controller.Call (T);
      else
         -- action if temperature could not be delivered immediately
         -- e.g. refine the measurement further
      end select;
   end loop;
end Sensor;
```

If the entry is not open then do not block this task and try something else

Time & Space

Interfacing with time

Relative timeouts on incoming calls

```
task body Controller is
Current_Temp : Temperature;
begin
   loop
      select
         accept Call (T: Temperature) do
            Current_Temp:= T;
         end Call;
      or
         delay 1.0; -- second
         -- action if the temperature was not available in time
      end select;
      -- normal processing
   end loop;
end Controller;
```

Accept sequences of any number of calls with less than 1 s interleaving.

Time & Space

Interfacing with time

Absolute timeouts on incoming calls

```
task body Controller is
Current_Temp : Temperature;
begin
   loop
      select
         accept Call (T: Temperature) do
            Current_Temp:= T;
         end Call;
      or
         delay until Deadline;
         -- action if the temperature was not available in time
      end select;
      -- normal processing
   end loop;
end Controller;
```

Accept sequences of any number of calls until an absolute time.

Time & Space

Interfacing with time

Non-blocking acceptance of incoming calls

```
task body Controller is
Current_Temp : Temperature;
begin
   loop
      select
         accept Call (T: Temperature) do
            Current_Temp:= T;
         end Call;
      else
         -- action if the temperature was not available in time
         -- e.g. try reading from the alternative source
      end select;
      -- normal processing
   end loop;
end Controller;
```

Only accepts calls if there are calls already waiting to be processed

Time & Space

Interfacing with time

Timeout on actions

All timeout schemas introduced up to now suspend / activate processes at well defined synchronization points.

What if we need to interrupt a running process due to a timeout?

☞ *Asynchronous transfer of control methods*

- Inside a single process: "Timeout on actions"
  ☞ discussed briefly here and in-depth in the chapter about asynchronism.
- Between processes: Part of scheduling methods
  ☞ discussed in the chapter about scheduling.

432

Time & Space

Interfacing with time

Relative timeout on actions

(Ada)

```
select
   delay 0.5; -- seconds
   -- below computations did not finish in time: take measures
then abort
   -- hard to predict sequence of computations
end select;
```


433

Time & Space

Interfacing with time

Absolute timeout on actions

(Ada)

```
select
   delay until Deadline;
   -- below computations did not finish in time: take measures
then abort
   -- hard to predict sequence of computations
end select;
```


434

Time & Space

Interfacing with time

Externally triggered timeout on actions

(Ada)

```
select
   Get_New_Data (Current_Sensor_Data);
   -- below computations did not finish before new data arrived: take measures
then abort
   -- hard to predict sequence of computations
end select;
```


435

Time & Space

Interfacing with time

Timeout on actions example

Common real-time systems concept:

Timeliness is often more important than Precision

1. Get a first approximation in fixed amount of time and well before the deadline.
2. Inspect the deadline and if there is enough spare time then proceed with:
3. Improve the result and keep a record of improvements (while keeping an eye on the deadline) and end in either of the cases:

   3a. The most precise result is achieved before the deadline occurred.

   3b. Otherwise use the closest approximation achieved before the deadline occurred.

☞ The deadline is fulfilled and there is a usable result in any case.

Time & Space

Interfacing with time

Timeout on actions example

Get a first approximation and employ spare time for refinements:

```
Deadline := … -- set an absolute deadline for the computations
-- compulsory computations (save first result)
select
   delay until Deadline;
   Precise_Result := False;
then abort
   while Result_Can_Be_Improved loop
      -- optimising computations (save results after each iteration)
   end loop;
   Precise_Result := True;
end select;
-- use result
```

Take a first guess safely before the deadline

If time left: improve, improve, improve

Continue process with the best result possible – given the deadline

Time & Space

Interfacing with time

Timeout on actions example

Time-base can also be given externally, e.g, via a protected call:

```
loop
   select
      Get_New_Data (Current_Sensor_Data);
      -- employ results based on previous data
      -- compulsory computations (save first result)
      -- employ first result on current data
   then abort
      while Result_Can_Be_Improved loop
         -- optimising computations (save results after each iteration)
      end loop;
   end select;
end loop;
```

On arrival of new data: ☞ abort current iterations

precise reactions

estimate first result

first reactions

If time left: improve, improve, improve

Time & Space

Interfacing with time

Timeout on actions

(RT-Java)

```
public class Timed extends AsynchronouslyInterruptedException
   implements java.io.Serializable
{
   public Timed (HighResolutionTime time) throws IllegalArgumentException;
   public boolean doInterruptible (Interruptible logic);
   public void    resetTime       (HighResolutionTime time);
}
```

☞ Similar semantic, yet *not* on compiler level, *but on library level*.

(Timeouts on actions in POSIX need to be manually implemented by employing timers, signals, and multiple processes.)

Time & Space

Notions of time and space

The big topics:

What is time? / What is embodiment?

☞ **Interfacing *with time***

☞ **Specifying *timing requirements***

☞ **Satisfying *timing requirements***

440

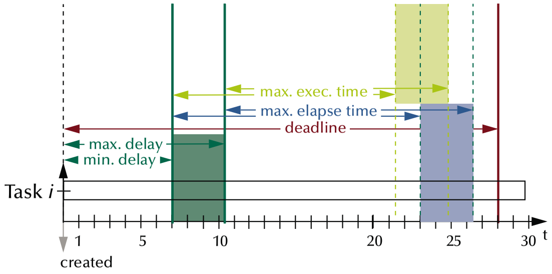
Time & Space

Specifying timing requirements

Temporal scopes

Common attributes:

- Minimal & maximal **delay** after creation
- Maximal **elapsed time**
- Maximal **execution time**
- Absolute **deadline**



441

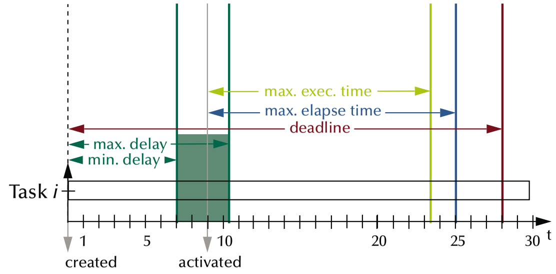
Time & Space

Specifying timing requirements

Temporal scopes

Common attributes:

- Minimal & maximal **delay** after creation
- Maximal **elapsed time**
- Maximal **execution time**
- Absolute **deadline**



442

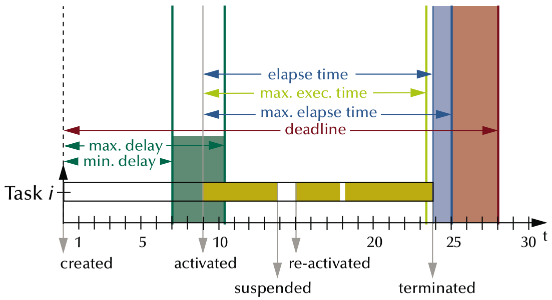
Time & Space

Specifying timing requirements

Temporal scopes

Common attributes:

- Minimal & maximal **delay** after creation
- Maximal **elapsed time**
- Maximal **execution time**
- Absolute **deadline**



443

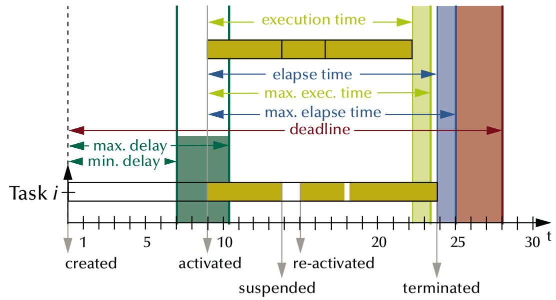
Time & Space

Specifying timing requirements

Temporal scopes

Common attributes:

- Minimal & maximal **delay** after creation
- Maximal **elapsed time**
- Maximal **execution time**
- Absolute **deadline**



Specifying timing requirements

Common temporal scope attributes

Temporal scopes can be:

Periodic	☞ controllers, routers, schedulers, streaming processes, …
Aperiodic	☞ periodic 'on average' tasks, i.e. regular but not rigidly timed, …
Sporadic / Transient	☞ user requests, alarms, I/O interaction, …

Deadlines can be:

"Hard"	☞ single failure leads to severe malfunction and/or disaster
"Firm"	☞ results are meaningless after the deadline
	☞ only multiple or permanent failures lead to malfunction
"Soft"	☞ results are still useful after the deadline

Semantics defined by application

Specifying timing requirements

Language support levels

1. **Time scopes** as **part of the language**:
   (Possible) schedulability analysis by the compiler
   ☞ Real-time Euclid, CRL, DSP, Pearl, …
2. **Time scopes** as **provided by (standardized) libraries**:
   (Possible) schedulability analysis by means of external tools
   ☞ Real-time Java, Ada, …
3. **Signal relation primitives** as part of the language:
   Causality analysis
   ☞ Esterel
4. **Timing primitives** as part of the language:
   Causality analysis (deadlock analysis) – no schedulability analysis
   ☞ Ada, C#, …

Specifying timing requirements

How to handle time-unbound primitives?

(loops, recursions, synchronizations, dynamic allocations, aborts)

1. Exclude them
   ☞ Real-time Euclid, Ada Ravenscar Profile
2. Expand them to become individually safe
   ☞ (e.g. by adding mandatory timeout)
3. Attribute the code with additional constraints, enabling a full pre-runtime analysis
   ☞ CRL, Pearl, DSP

Specifying timing requirements

Real-time Euclid

Language features:

- Recursions and "goto" statements are prohibited.
- Loops are restricted to (time) bounded loops.
- Processes are static and non-nested.

Time scopes:

```
periodic <Frameinfo> first activation <TimeOrEvent>
atEvent <ConditionalId> <Frameinfo>
```

Specifying timing requirements

Real-time Euclid example

```
realTimeUnit := 1.0 % seconds
var Reactor          : module
var startMonitoring : activation condition atLocation 16#A10D
   process TempController : periodic
                              frame 60.0
                              first activation atTime  600.0
                                            or atEvent startMonitoring
      % import list
      %
      % execution part
      %
   end TempController
end Reactor
```

Connect an event variable to an interrupt address

Define the time scope for the process

No loop statements as tasks are static and activated according to time scope

Specifying timing requirements

Real-time Euclid example *(emulated in Ada)*

```
task body Temp_Controller is
  begin
    select
      accept Start_Monitoring;
    or
      delay 600.0; -- sec.
    end select;
    declare
      Next_Release : Duration := Clock + 60.0; -- sec.
    begin
      loop
        -- execution part
        delay until Next_Release;
        Next_Release := Next_Release + 60.0; -- sec.
      end loop;
    end;
  end Temp_Controller;
```

Same effect, yet no schedulability analysis!

Specifying timing requirements

Real-time Euclid – Language status

Real-Time Euclid was suggested by Kligerman and Stoyenko in 1986.

- Additional schedulability analysis modules became available.
- Stayed in an academic context, but influenced many more recent Real-time systems.

Specifying timing requirements

CRL

(a language for complex real-time systems)

Constraints in CRL on:

- **Time**:
  ```
  timeconstraint
      {use | nosoonerthan | nolaterthan <abs_time>} endtimeconstraint
  ```
  (also relative constraints)
- **Iterations**:
  assert lower and upper limits for the number of iterations per block.
- **Activations**:
  ```
  activationdeactivationconstraint
    [periodic <Frameinfo> firstactive <TimeOrEvent>
  | atEvent <ConditionalId> <Frameinfo>] endactivationdeactivationconstraint
  ```
- **Direct recursions**:
  assert lower and upper recursion limits (general recursions are not covered).

452

Time & Space

Specifying timing requirements

CRL

(a language for complex real-time systems)

Evaluating those constraints and assertions in CRL:

- **Timing**:
  ☞ Verified at compile-time
- **Activation / Deactivation**:
  ☞ Checked for schedulability at compile-time and enforced by the scheduler at run-time.
- **Iterations** and **recursion**:
  ☞ Either verified at compile-time or checked at run-time.

453

Time & Space

Specifying timing requirements

CRL – Language Status

(a language for complex real-time systems)

CRL was suggested by Stoyenko, Marlowe and Younis in 1995

- Full featured language
- Compiled to attributed C++

☞ Stayed also in an academic context.

454

Time & Space

Specifying timing requirements

Pearl

Explicit time-scope expressions:

```
TaskStart      ::= [StartCondition] ACTIVATE <task>
StartCondition ::= AT <time> [Frequency] |
                   AFTER <duration> [Frequency] |
                   WHEN <interrupt> [AFTER <duration> [Frequency]] |
                   Frequency
Frequency      ::= ALL <duration>
                      [{UNTIL  <time>     } |
                      {DURING <duration>}]
```

☞ Schedulability analysis (at compile-time or run-time) possible (although not defined by the language).

☞ Pearl refinement: a combination of Pearl and Real-time Euclid ☞ High-Integrity Pearl.

455

Time & Space

Specifying timing requirements

DPS: Distributed Programming System

Explicit time-scope expressions at the 'statement level': e.g. time-scope for a software-engineer:

```
from 11:00 to 19:30 every 45 do
   start elapse 10 do
      setup_coffee_machine
      power_coffee_machine_up
      find_favorite_cup
      put_coffee_in_favorite_cup
      clean_coffee_machine
   end
   start after 3 elapse 25 by 20:00 do
      drink_coffee
   end
end
```

☞ DPS-compiler: breaks all codes down to processes and schedules

Time & Space

Specifying timing requirements

Real-time Java

Real-time Java comes with libraries providing:

- Multiple sets of predefined time-scope parameters
- A scheduler class (with a predefined priority scheduler)

☞ Schedulability (feasibility) analysis possible.

Time & Space

Specifying timing requirements

Real-time Java

```
public abstract class SchedulingParameters
{
   public SchedulingParameters ();
}
public class PriorityParameters extends SchedulingParameters
{
   public PriorityParameters (int priority);
   public int  getPriority ();
   public void setPriority (int priority) throws …;
   …
}
```

priority is the only default scheduling parameter

The absence of range types makes priority an int

Time & Space

Specifying timing requirements

Real-time Java

```
public abstract class ReleaseParameters
{
   protected ReleaseParameters
               (RelativeTime      cost,
                RelativeTime      deadline,
                AsyncEventHandler overrunHandler,
                AsyncEventHandler missHandler);
   public RelativeTime      getCost();
   public AsyncEventHandler getCostOverrunHandler();
   public RelativeTime      getDeadline();
   public AsyncEventHandler getDeadlineMissHandler();
}
```

Measuring execution time (cost) is not required by the language, i.e. overrunHandler might never be activated

Note that deadline is a RelativeTime.

Time & Space

Specifying timing requirements

Real-time Java

```
public class PeriodicParameters extends ReleaseParameters
{
   public PeriodicParameters
              (HighResolutionTime start,
               RelativeTime       period,
               RelativeTime       cost,
               RelativeTime       deadline,
               AsyncEventHandler  overrunHandler,
               AsyncEventHandler  missHandler);
   public RelativeTime       getPeriod ();
   public HighResolutionTime getStart ();
   public void               setPeriod (RelativeTime period);
   public void               setStart (HighResolutionTime start);
}
```

460

Time & Space

Specifying timing requirements

Real-time Java

```
public class AperiodicParameters extends ReleaseParameters
{
   public AperiodicParameters
             (RelativeTime      cost,
              RelativeTime      deadline,
              AsyncEventHandler overrunHandler,
              AsyncEventHandler missHandler);
}
```


461

Time & Space

Specifying timing requirements

Real-time Java

```
public class SporadicParameters extends AperiodicParameters
{
   public SporadicParameters
             (RelativeTime      minInterarrival,
              RelativeTime      cost,
              RelativeTime      deadline,
              AsyncEventHandler overrunHandler,
              AsyncEventHandler missHandler);
   public RelativeTime getMinimumInterarrival ();
   public void         setMinimumInterarrival (RelativeTime minimum);
}
```


462

Time & Space

Specifying timing requirements

Real-time Java

```
public class RealtimeThread extends java.lang.Thread
   implements Schedulable
{
   public RealtimeThread (SchedulingParameters s,
                          ReleaseParameters r,
                          MemoryParameters m,
                          MemoryArea a);
   public synchronized void           addToFeasibility ();
   public synchronized void           addIfFeasible ();
   public static       RealtimeThread currentRealtimeThread () throws …;
   public synchronized void           schedulePeriodic ();
   public synchronized void           deschedulePeriodic ();
   public              boolean        waitForNextPeriod () throws …;
   public synchronized void           interrupt ();
   public static       void           sleep (…) throws …;
   …
}
```

Scheduling parameter: Priority

Release parameters: Periodic, Aperiodic, or Sporadic

463

Time & Space

Specifying timing requirements

Real-time Java

```
public class NoHeapRealtimeThread extends RealtimeThread
{
   public RealtimeThread (SchedulingParameters s,
                          ReleaseParameters r,
                          MemoryArea a) throws …;
   …
}
```

Time & Space

Specifying timing requirements

Real-time Java

isFeasible implements a runtime schedulability analysis

```
public abstract class Scheduler
{
   protected Scheduler ();
   protected abstract boolean addToFeasibility (Schedulable s);
   public    abstract void    fireSchedulable (Schedulable s);
   public    abstract boolean isFeasible ();
   protected abstract boolean removeFromFeasibility (Schedulable s);
   public             boolean setIfFeasible (Schedulable s,
                                             ReleaseParameters r,
                                             MemoryParameters m);
   …
}
```


Time & Space

Specifying timing requirements

Real-time Java

PriorityScheduler is the only required instance.

```
public class PriorityScheduler extends Scheduler
{
   public static final int MAX_PRIORITY;
   public static final int MIN_PRIORITY;
   protected PriorityScheduler ();
   protected boolean addToFeasibility (Schedulable s);
   public    void    fireSchedulable (Schedulable s);
   public    boolean isFeasible ();
   protected boolean removeFromFeasibility (Schedulable s);
   public    boolean setIfFeasible (Schedulable s,
                                    ReleaseParameters r,
                                    MemoryParameters m);
   …
}
```


Time & Space

Specifying timing requirements

Real-Time Java – Language Status

- Stayed in an academic niche since its introduction in 2001.
  No practical deployment stories in high integrity systems are known.
- Original reference implementation by TimeSys.
  Last remaining, current implementation might be JamaicaVM.
- Real-time Java is based on the concept of scoped memory
  – yet scoped memory does not embed smoothly into the larger Java context, while a growing community is working on "real-time" garbage collectors.

☞ Real-time Java might be replaced by a new breed of Java virtual machines which come with "real-time" garbage collectors.

Time & Space

Specifying timing requirements

Ada

… only a few time scope expressions at language level!

Extensions are used to specify other time scopes, based on

- Tasks
- Schedulers (fixed priorities, dynamic priorities and earliest deadline first)
- Timers
- Asynchronous transfer of control
- Code attribution via contracts

Time & Space

Specifying timing requirements

Esterel

Since Esterel is a synchronous language, …

… all actions and communications take zero time by definition.

☞ There is no expression for continuous, non-zero time-scopes.
☞ Time is interpreted as a sequence of events.
☞ Time-scopes translate to signal-relations and signal-counters.

Yet, continuous time scopes need to be taken into account while

1. Analysing and reducing the problem to a zero-time atomic system
2. Implementing the synchronous system on an actual system.

☞ Continuous time-scopes required for the validation of the zero-time assumption!

Time & Space

Notions of time and space

The big topics:

What is time? / What is embodiment?

☞ **Interfacing** *with time*

☞ **Specifying** *timing requirements*

☞ **Satisfying** *timing requirements*

Time & Space

Satisfying timing requirements

Two paths towards fulfilling real-time requirements

☞ **Real-time logic** approach

Formal, correct in its specifications & offers calculus for asynchronous, real-time systems.
➘ Needs to ignore most real world effects, like jitters, drifts, failures, interferences, etc. pp..
➚ **Gives a correct solution according to the specification.**

☞ **Complex systems** approach

Deals with existing computer systems, sensors, & offers a set of approximating methods.
➘ Not complete or correct in any formal sense.
➚ **Deals with real-world systems, gives 'robust' systems, passes rigorous experiments.**

Time & Space

Satisfying timing requirements

Two paths towards fulfilling real-time requirements

Complex systems approach

1. System identification and compile-time analysis:
   - ☞ Calculate or limit statement durations. ☞ Data-sheets.
   - ☞ Calculate or limit iterations and recursions. ☞ Program verification methods.
   - ☞ Analyse potential dead- or life-locks ☞ *chapter "Resource Control".*
   - ☞ Calculate schedulability ☞ *chapter "Scheduling".*
2. Run-time analysis and checks:
   - ☞ Dynamic scheduling schemes: Re-validate schedulability ☞ *chapter "Scheduling".*
   - ☞ Check for all constraints and assertions at run-time ☞ *chapter "Reliability".*
3. Supply fault-tolerant behaviours:
   - ☞ Error recoveries, mode changes, … ☞ *chapter "Reliability".*

Satisfying timing requirements

Two paths towards fulfilling real-time requirements

Real-time logic approach ☞ *chapter "Reliability"*

1. Reduce the problem:
   ☞ Reduce any asynchronous, analogue, dynamical, fractal, jitter-, drift, or failure-affected parts of the system to a fully synchronous and discrete system.
   ☞ Formulate the specification on the basis of the reduced, synchronous system.
2. Verify the reduced system:
   ☞ Verify the reduced synchronous against the specification ☞ Program verification methods.
3. Compile the reduced system to an actual system:
   ☞ The resulting actual system will be executable on real machines and employ real devices.
4. Re-check the actual system:
   ☞ Complex Systems approach.

Summary

Time & Space

- **What is time? / What is embodiment?**
  - Approaches by different faculties to understand the foundations of this course
- **Interfacing with time**
  - Formulating local, time-dependent constraints
  - Access time, delay processes, timers
  - Timeouts, asynchronous transfer of control
- **Specifying timing requirements**
  - Formulating global timing-constraints
  - Understanding time-scope parameters (and expressing them in different languages)
- **Satisfying timing requirements**
  - Real-time logic approach & Complex systems approach