Notions of time and space

What is time?

Is time an intrinsic property of nature? Is time a construct which is based on observable events?

Is time an extrinsic property of nature? Is time an extrinsic phenomenon? Is time a construct which is based on observable events?

Is time an extrinsic property of nature? Is time a construct which is based on observable events?

Is time an extrinsic property of nature? Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?

Is time a construct which is based on observable events?
**Time & Space**

### 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 (ET)**
  - 1963: based on observing the motion of the solar system

- **Universal Time Coordinated (UTC)**
  - Since 1964: An atomic clock which is synchronized with IAU (International Astronomical Union) time by introducing occasional leap ticks. UTC - TAI $< 0.5 s$

- **Ephemeris Time (ET) 1 & 2**
  - 1967: corrections to the ET model

- **1 second**
  - $1/86,400$ of a mean solar day
  - $1/31,566,925.9747$ of the tropical year for 1900

#### ‘Real-time’ clock

- **Real-time** 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.

- Using an existing time-frame
  - by employing a RTC-module (a timer based on the notion of seconds).
  - 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 ‘1 second’ resolution and accuracy of time (in reference to ‘a second’) … not its origin or meaning.

---

### What is embodiment?

#### Edmund Husserl

- Founder of the phenomenological tradition
- as a tool 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.

- Phenomenology uses the life-world as both the raw material and the context of its research.

- Husserl rejected pure abstract and formalized reasoning.

---

### Phenomenology

**The phenomena of experience as the central aspects and blocks of understanding**

- Ontology: the nature of being and categories of existence
- Epistemology: the study of knowledge

---

**Applied to and trying to combine aspects of**

- ‘How do we know?’
- ‘What is it that we are doing’
- ‘What is the world like?’
- ‘What is the nature of being?’
- ‘What are the rules of thought?’

---

**Martin Heidegger**

- Moved phenomenology from a discussion about mental phenomena separated from the physical world (Carnelian daydream) 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
  - *Zuhanden* (being-at-hand): being an inseparable part of the world in which it occurs or being as always present and active in the world as an unconscious but accessible background
  - *Vorhanden* (ready-to-hand): equipment as a part of actual interaction with the world
  - *In handy (present-at-hand): equipment as a conscious model*
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 a 'universal intelligence', 'autonomy', 'conscience' or any other cognitive process which is independent of a physical environment.
- There is no such thing as a Universal morphology (mechanical design, robot device, ...) which is useful or even operational in all physical environments.

Interfacing with time

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

Specifying timing requirements

To satisfy timing requirements

Satisfying timing requirements

What is time? / What is embodiment?

What is time in Ada Real-Time?

package Ada.Real-Time is

package Ada.Real-Time.Timing_Events is

Ada.Real-Time.Timing_Events package
Notions of time in RT-Java

RT-Java time classes

- Time root class: public abstract class Time implements java.lang.Comparable
- Similar to Ada.Real-Time, but no mandatory accuracy.

Real-time clock interface in POSIX

```
int clock_settime (clockid_t clock_id, time_t tv_sec, clock_t tv_nsec);
```

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.

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

Interfacing with time

```
Start_Time := Clock;
Action;
Next_Time := Clock + Interval;
```

Programming primitive 'Timer'

Semantic: Activate a specified routine after a predefined time span or at a predefined absolute time.

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

Absolute delay implemented by relative delay?

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

Absolute delay implemented by relative delay?

```
task body T is
  constant Start_Time := Time;
  begin loop
    Action;
    delay Interval - (Clock - Start_Time);
    end loop;
end T;
```
**Time & Space**

### Interfacing with time

**Zero delay (Ada)**

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

Delay statements activate the scheduler.

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


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

Yield statements activate the scheduler.

Yield is the same thing in a nicer word.


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

Yield_To_Higher statements activate the scheduler.

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

### Relative regular timer (Ada)

```
protected Event_Handlers is
  procedure Timer_Routine (Event : in out Timing_Event);
private
  Interval : constant Duration := 5.0; -- acc.
end Event_Handlers;
protected body Event_Handlers is
  procedure Timer_Routine (Event : in out Timing_Event) is
    begin
      Action;
      Next_Time := Clock + Interval;
      Set_Handler (Event, Next_Time, Timer_Routine'access);
      Next_Time := Next_Time + Interval;
      Set_Handler (Event, Next_Time, Timer_Routine'access);
      -- if executed in a real-time system, the drift will be significantly smaller compared to delay statements.
    end Timer_Routine;
end Event_Handlers;
```

### Absolute regular timer (Ada)

```
protected Event_Handlers is
  procedure Timer_Routine (Event : in out Timing_Event);
private
  Interval : constant Duration := 5.0; -- acc.
end Event_Handlers;
protected body Event_Handlers is
  procedure Timer_Routine (Event : in out Timing_Event) is
    begin
      Action;
      Next_Time := Clock + Interval;
      Set_Handler (Event, Next_Time, Timer_Routine'access);
      Next_Time := Next_Time + Interval;
      Set_Handler (Event, Next_Time, Timer_Routine'access);
      -- if executed in a real-time system, the drift will be significantly smaller compared to delay statements.
    end Timer_Routine;
end Event_Handlers;
```

### 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).

### Timeouts on semaphores (POSIX)

```
if (sem_timedwait (&call, &timeout) < 0) {
  /* something went bad */
}
else {
  /* semaphore is locked successfully, go ahead */
}
```

Exception handling is nice, isn't it?
Interfacing with time

Relative timeouts on entry calls

For task-entry calls and protected object calls:

```ada
task body Sensor is
  T : Temperature;
begin
  loop
    -- find temperature T somewhere
    select
      Controller.Call (T);
      or
      delay 8.5; -- seconds
    -- action if temperature could not be delivered in time
    end select;
  end loop;
end Sensor;
```

Absolute timeouts on entry calls

For task-entry calls and protected object calls:

```ada
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;
```

Relative timeouts on incoming calls

For incoming calls:

```ada
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;
  end loop;
end Controller;
```

Absolute timeouts on incoming calls

```ada
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;
  end loop;
end Controller;
```

Non-blocking acceptance of incoming calls

```ada
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 temperature could not be delivered in time
    -- e.g. try reading from the alternative source
    end select;
  end loop;
end Controller;
```

Timeout on actions

```ada
select
  delay 8.5; -- seconds
-- action if the temperature was not available in time
-- e.g. try reading from the alternative source
end select;
```

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.
### Interfacing with Time

#### Timeouts on Actions Example

1. Get a first approximation and employ spare time for refinements:
   ```
   Deadline := ...  
   ```
2. Compulsory computations (save first result):
   ```
   Get_New_Data (Current_Sensor_Data);  
   ```
3. Before the deadline, improve approximations and before the deadline:
   ```
   select ...  
   ```
4. Compulsory computations (save result on current data):
   ```
   Precise_Result := False;  
   ```
5. Below computations did not finish before new data arrived: take measures
   ```
   if time left: improve, improve, improve  
   ```
6. Optimizing computations (save results after each iteration)
   ```
   loop  
   ```
7. The most precise result is achieved before the deadline occurred.
   ```
   Precise_Result := True;  
   ```
8. If time left: improve, improve, improve
   ```
   end loop;  
   ```
9. The deadline is fulfilled and there is a usable result in any case.
   ```
   select ...  
   ```
10. Continue process with the best result possible
    ```
    `
Specifying timing requirements

Temporal scopes

Common attributes:
- Minimal & maximal delay after creation
- Minimal elapsed time
- Minimal execution time
- Absolute deadline

Deadlines can be:
- "Hard": Single failure leads to severe malfunction and/or disaster
- "Soft": Results are still useful after the deadline

Time scopes:
1. Periodic
   - Controls, routines, schedules, streaming processors
2. Aperiodic
   - On spot, once-only actions, alarms

Real-time Euclid example

```
realTimeUnit := 1.0 % seconds

var
  Reactor : module
  TempController : process

tempController := TempController.periodic
end
```

Real-time Euclid example

```
win := RealTimeEuclid阅览
```

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.

Common temporal scope attributes

<table>
<thead>
<tr>
<th>Periodic</th>
<th>Aperiodic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enables</td>
<td>Constraints in CRL on:</td>
</tr>
<tr>
<td>Tasks</td>
<td>- Time: timeconstraint (next</td>
</tr>
<tr>
<td>Concurrency</td>
<td>- Concurrency: activation</td>
</tr>
<tr>
<td>Direct recursion</td>
<td>- Direct recursion: assert lower and upper recursion limits (general recursions are not covered), result is meaningless after deadline</td>
</tr>
</tbody>
</table>

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

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.

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

(CRL was suggested by Stoyenko, Marlowe and Younis in 1995)

• Full featured language
• Compiled to attributed C++

CRL – Language Status

(a language for complex real-time systems)
Specifying timing requirements

Real-time Java

```java
public class RealtimeThread extends java.lang.Thread implements SchedulingParameter: Priority
{
    // Implementation of scheduling parameters

    public synchronized void addIfFeasible();
    public synchronized void deschedulePeriodic();
    public void interrupt();
    public static RealtimeThread currentRealtimeThread();
    // More implementation details...
}
```

Real-Time Java – Language Status

- 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.

Notions of time and space

Esterel

Since Esterel is a synchronous language, all actions and communications take zero time by definition.

1. Analysing and reducing the problem to a zero-time atomic system
2. Satisfying timing requirements

The big topics:

- What is time? / What is embodiment?
- Interfacing with time
- Specifying timing requirements
- Satisfying timing requirements
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

Two paths towards fulfilling real-time requirements

- **Real-time logic approach**
  - Formal, connects its specifications & offers calculus for asynchronous, real-time systems.
  - Needs to ignore most real world effects, like jitter, drifts, failures, interferences, etc. pp.
  - Gave a correct solution according to the specification.

- **Complex systems approach**
  - Deals with existing computer systems, sensors, coercers a set of approximating methods.
  - Not complete or correct in any formal sense.
  - Deals with real-world systems, gives `real` systems, passes rigorous experiments.

Summary

- **Time & Space**
- **Real-time logic approach**
  - Chapter "Reliability" page 471
- **Complex systems approach**
  - Chapter "Reliability" page 471

Satisfying timing requirements

Two paths towards fulfilling real-time requirements

1. **System identification and compile-time analysis:**
   - Calculate or limit statement durations. See Data-sheets.
   - Calculate or limit iterations and recursions. See Program verification methods.
   - Analyze potential dead- or life-locks see chapter "Resource Control".
   - Calculate schedulability see chapter "Scheduling".

2. **Run-time analysis and checks:**
   - Dynamic scheduling: Revalidate schedulability see chapter "Scheduling".
   - Check for all constraints and assertions at run-time see chapter "Reliability".

3. **Supply fault-tolerant behaviours:**
   - Error recoveries, mode changes, … see 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 see 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.