

Synchronous Programming of Reactive Systems

Pascal Raymond, Verimag-CNRS

Reactive Systems

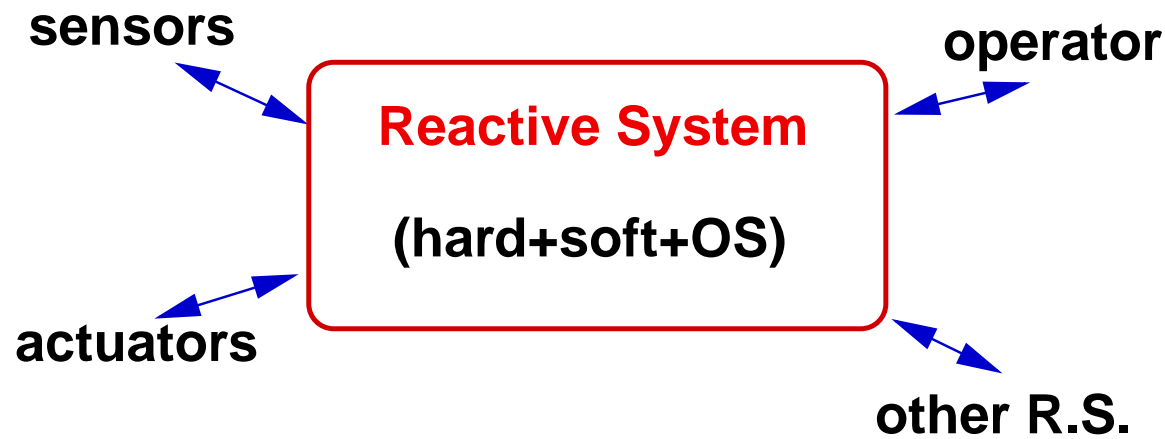
Overview

- Permanent reaction to an environment *that cannot wait*
≠ transformational (e.g. compiler)
- Real-time constraint
≠ interactive (e.g. IHM, browser etc)
The environment is (partly) the physic world

Examples

- Control/command in industry, embeded systems in transportation
- Very critical (powerplants, airplanes), or less (mobile phones).

General Scheme

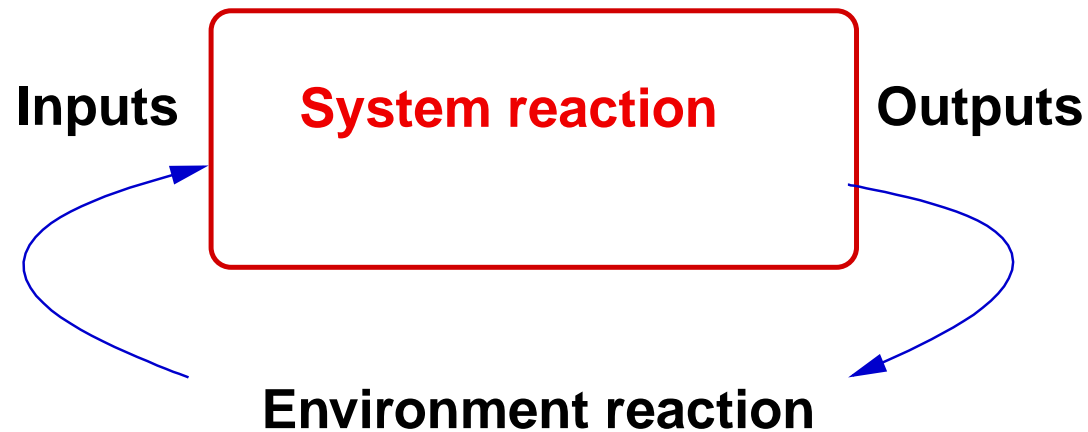


- Environment: interface with physics, human operator, other reactive systems ...
- The “program”: a particular software with a particular OS on a particular hardware

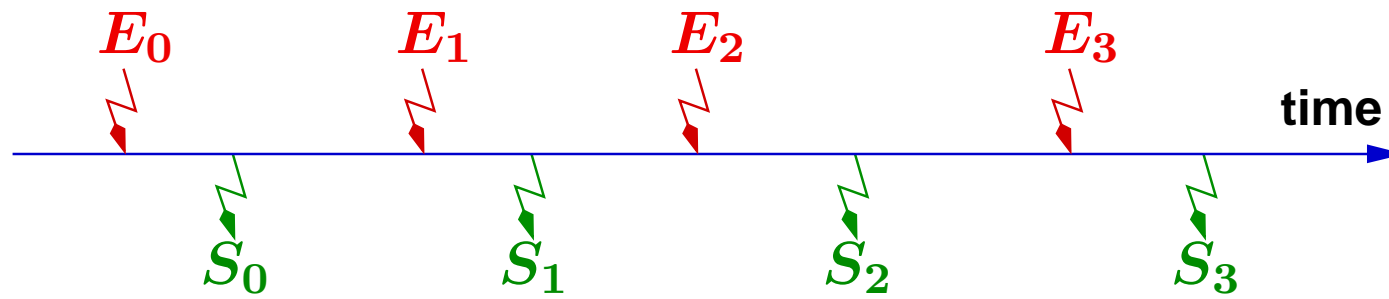
Lots of problems!

⇒ **Let's focus on *functionnality***

Behavior of a reactive system



- “Software” inputs/outputs (Boolean, integer or floating values)
- Execution = sequence of reactions
- Real-time = E and S are alternating among time



Fonctionnality

Determinisme

- A given input sequence always produce the same output sequence
- As a consequence:
 S_i is fonctionnaly determined by the sequence E_1, E_2, \dots, E_i
- $\forall i \ S_i = \phi(E_1, E_2, \dots, E_i)$

Another constraint: bounded memory

- $\exists M_0, g \ S_i = f(M_i, E_i) \ M_{i+1} = g(M_i, E_i)$

Implementation of a reactive system _____

First, identify:

- inputs E , outputs S
- the necessary memory M , with its initial value M_0

Then define:

- The output function $S_i = f(M_i, E_i)$
- The transition function $M_{i+1} = g(M_i, E_i)$

At last: implement all that with a program

Simple implementation (event-driven)

```
System(E, S)
```

```
  memory M
```

```
  M := M0
```

```
  loop
```

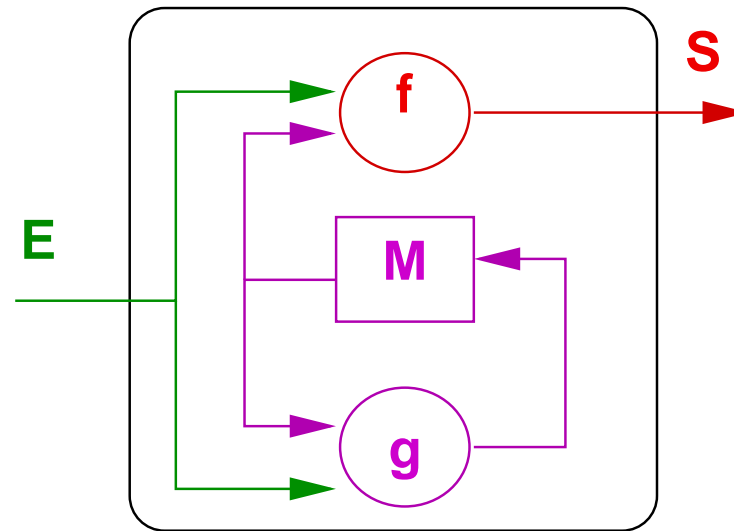
```
    wait(E)
```

```
    S = f(M, E)
```

```
    M = g(M, E)
```

```
    write(S)
```

```
  end loop
```



What about real-time?

execution time < reaction time of the environment

Even simpler implementation (sampling)

```

System(E, S)
  memory M
  M := M0
  each period do
    read(E)
    S = f(M, E)
    M = g(M, E)
    write(S)
  end

```

Real-time?

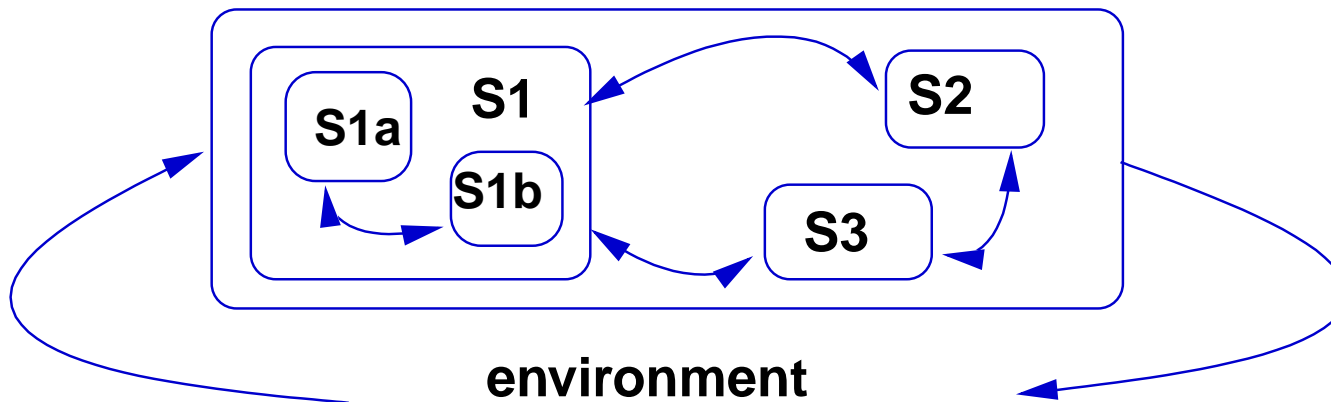
execution time < period

and ad hoc period for a *known environment*

Synchronous Programming _____ Implementation of a reactive system

Complex Reactive System

- Lots of inputs, outputs, memories
- Output/transition functions are untractable
- Classical solution: hierarchical and parallel decomposition



Expected behavior: each sub-system locally behaves as a real-time system

Logical concurrency

- Concurrency may be mandatory (distributed system),
- or just logical: the actual architecture is centralized

Implementation with concurrent processes

Logical concurrency becomes physical concurrency:

- One process for each sub-system
- Scheduling/communication at execution time
 - ★ System calls (real-time OS)
 - ★ Language statements (multi-tasks languages)

⇒ **Problem: what is the global behavior?**

Problems related to the multi-task approach

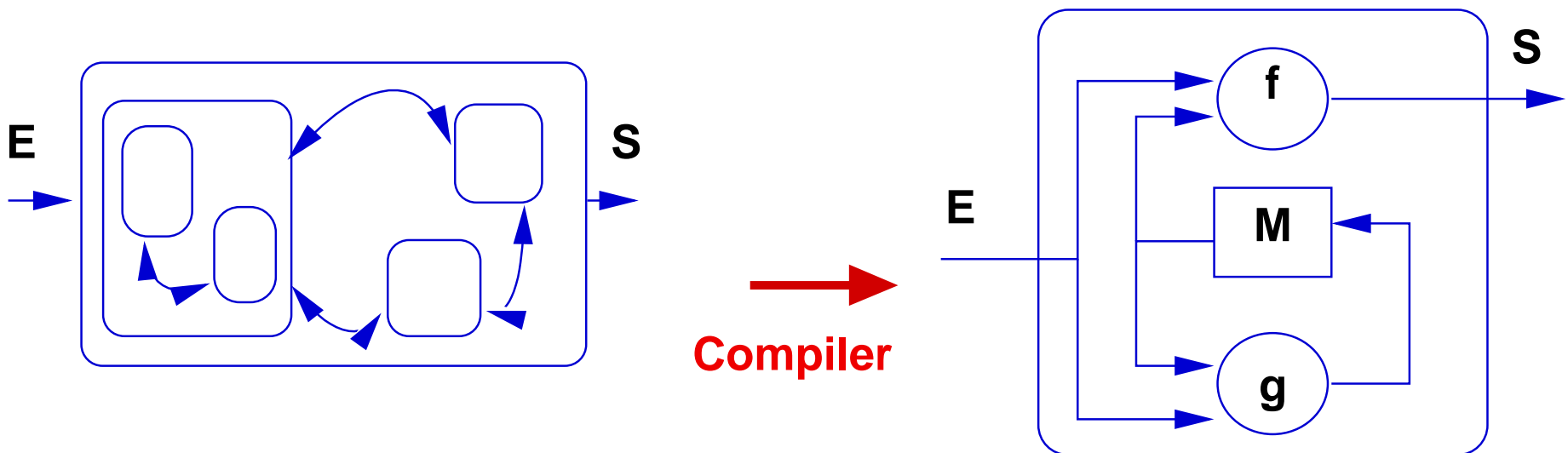
dynamic scheduling *is unpredictable*:

- The communication order (even with priorities or rendez-vous) is unpredictable \Rightarrow hard to guarantee determinism
- Execution time is unpredictable \Rightarrow hard to guarantee real-time

Synchronous approach

Conciliate:

- modular and concurrent design
- determinism and real-time



Synchronous hypothesis

Ideally (design level)

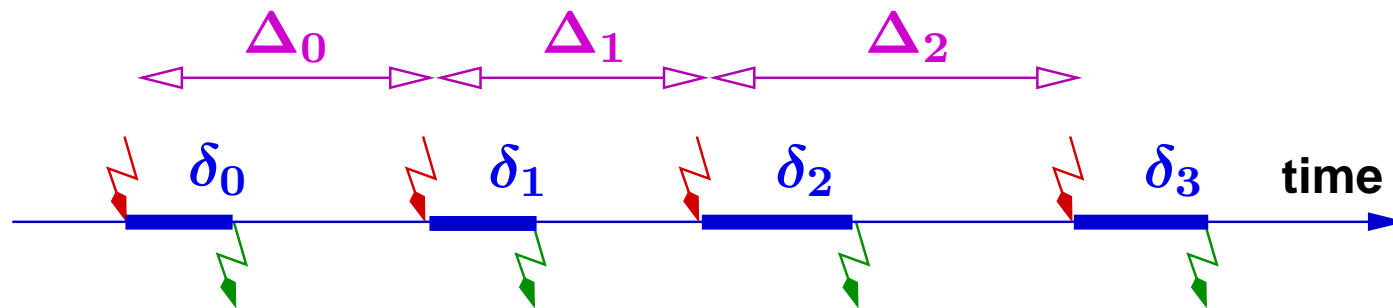
- Non blocking, instantaneous communication (synchronous broadcast)
- Instantaneous reaction
- Composition is free: $0 + 0 = 0$ (idealized modularity)
- Leads to a notion of discrete, logical time (inputs sequence)

Concretely (execution level)

Atomic reactions are *simple* (no unbounded loops, bounded memory):

⇒ there exist a upper bound to the reaction time

⇒ which can be *evaluated* for a given architecture



- let δ_{max} be an upper bound of all δ_i (for a given hardware),
- let Δ_{max} be an upper bound of all Δ_i (for a given environment),
- Synchronous hypothesis is valid if $\delta_{max} < \Delta_{max}$

Is it really new? _____

Classical in synchronous circuits

- Sequential (i.e. clocked) circuits, with gates and latches
- Communicating Mealy machines (synchronous automata)

Classical in control engineering

(data-flow formalisms)

- differential or finite difference equations
- block-diagrams, analog networks

Less classical in software

Synchronous languages

Same principles

- Synchrony (discrete time)
- Logical concurrency
- Compilation to simple sequential code (static scheduling)

Different styles

- Declarative, data-flow:
 - ★ textual (Lustre, Signal), or graphical (Scade/Lustre)
- Imperative, sequential:
 - ★ textual (Esterel), or graphical (SynchCharts)

Industrial use

Avionics

- Airbus, Honeywell, Eurocopter, Hispano-Suiza (Scade/Lustre)
- Dassault (Esterel)

Nuclear plants

- Schneider-Electric, EDF (Scade/Lustre)

CAD

- Cadence, Synopsys (Esterel)

Telecom

- Thomson, TI (Esterel)

Synchronous Programming _____ Industrial use