Synchronous Programming of Reactive Systems

Pascal Raymond

Verimag-CNRS

MOSIG - Embedded Systems

Reactive Systems

Overview

- Permanent reaction to an environment *that cannot wait*

  $\neq$ transformational (e.g. compiler)

- Real-time constraint

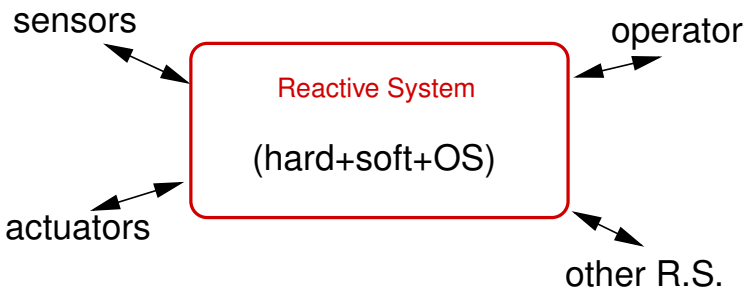
  $\neq$ interactive (e.g. IHM, browser etc)

  The environment is (partly) the physical world

Examples

- Control/command in industry, embedded systems in transportation
- Very critical (power-plants, airplanes), or less (mobile phones).

General Scheme

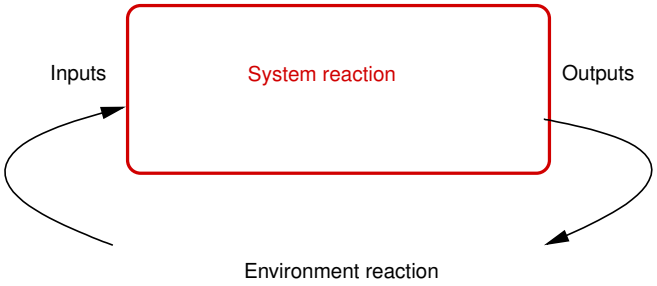


- Environment: interface with physics, human operator, other reactive systems ...
- The “program”: a particular software and a particular OS running on a particular hardware

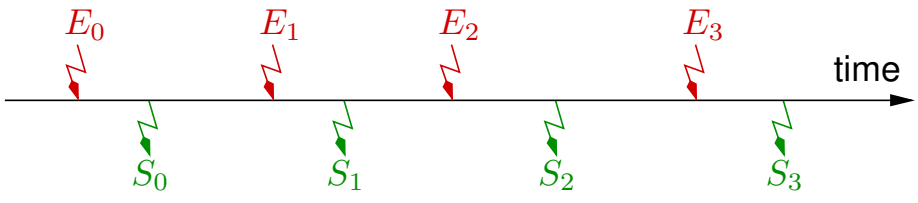
Lots of problems!

$\Rightarrow$ Let’s focus on *functionality*

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



Functionality

Determinism

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

Additional constraint: bounded memory

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

Implementation of a reactive system

First, identify:

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

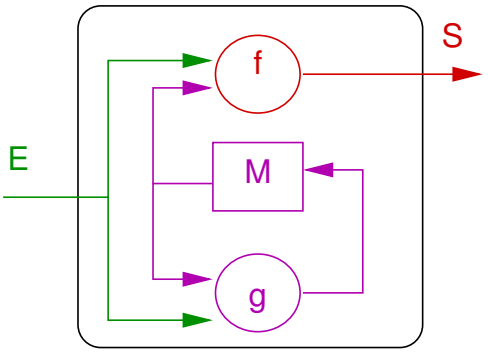
Then define:

- 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 using some programming language (e.g. C, assembly)

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
```

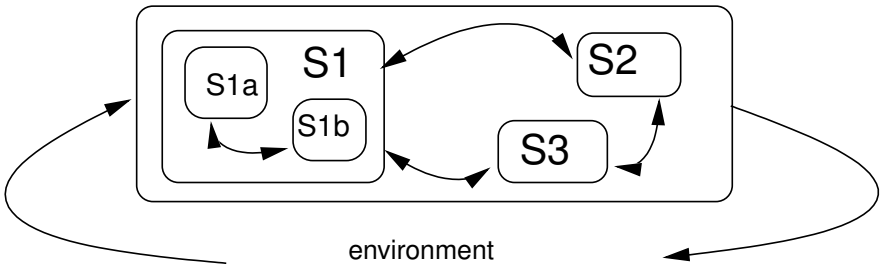
Real-time?

execution time $<$ period

and ad hoc period for a *known environment*

Complex Reactive System

- Lots of inputs, outputs, memories
- Output/transition functions are intractable
- 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

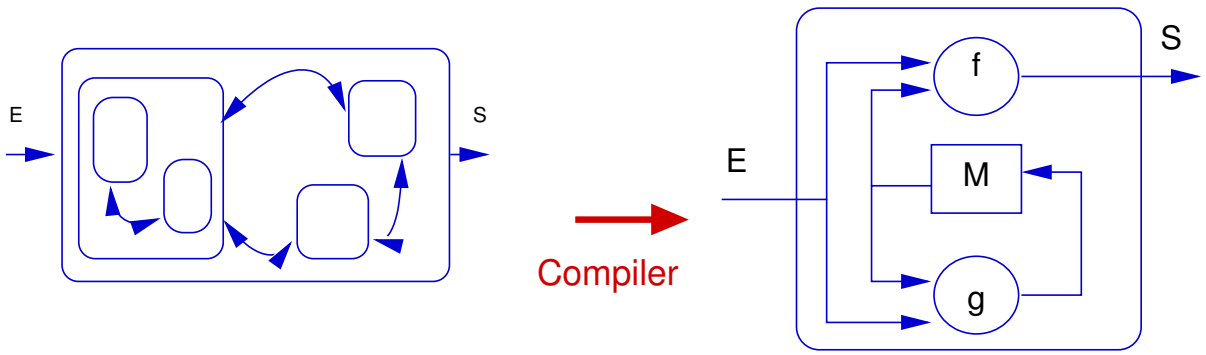
dynamic scheduling *is unpredictable*:

- The communication order (even with priorities or rendezvous) 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

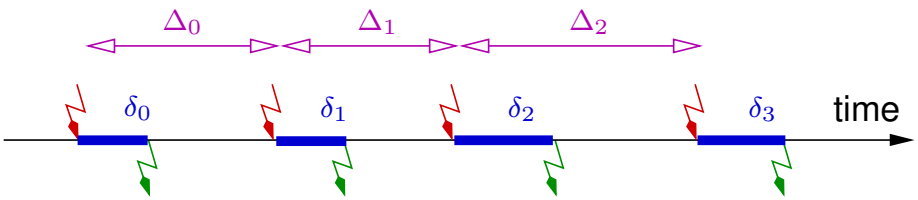
Ideally (design level)

- 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): $\Rightarrow$ there exists an upper bound to the reaction time

$\Rightarrow$ 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_{min} be a lower bound of all Δ_i (for a given environment),
- Synchronous hypothesis is valid if $\delta_{max} < \Delta_{min}$

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/Syldex)
- Imperative, sequential:
  - ↪ textual (Esterel), or graphical (SynchCharts)

Industrial use

Main domains (and companies) that are using synchronous languages/tools:

Avionics, Space, Defense

- Airbus, BAE, EADS, Lockheed, Rolls-Royce, Embraer ...

Railway

- Alstom Trans., Ansaldo STS, AREVA TA, RATP, Siemens Mob., Thales RSS ...

Nuclear power plants

- AREVA NP, Rolls-Royce CN ...

Misc. critical industry

- BMW, Schindler Elevators, Mitsubishi, Subaru, Toyota ...