Reactive programming

and how it fits within control systems

Vincent Michel @ ESRF

ICALEPCS 2017 - Barcelona

GitHub: **vxgmichel/icalepcs-reactive-programming**

Slides: **tinyurl.com/icalepcs-rp**

⚠ **Warning :** contains real code chunks!

What is reactive programming ?

It's → About → Propagating → Changes

Hum, this looks like a pipeline...

Imperative → assignment

```
C = A + B
```

C is **NOT** updated if A or B changes

Reactive → definition

```
C := A + B
```

C **IS** updated if A or B changes

Examples

Python properties

```
@property
def C(self):
    return self.A + self.B
```

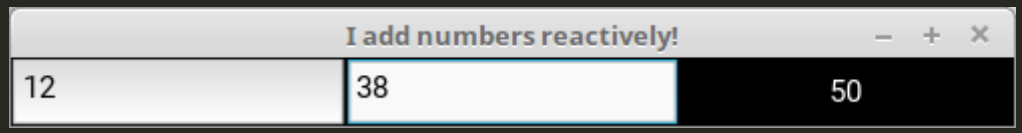
```
obj.A = 1
obj.B = 2
assert obj.C == 3
# Here comes the change!
obj.B = 10
assert obj.C == 11
```

Descriptive, but **not asynchronous**

Kivy/QML (declarative approach)

```
TextInput:
    id: A
    text: '0'
TextInput:
    id: B
    text: '0'
Label:
    id: C
    text: str(int(A.text) + int(B.text))
```

The kivy app:



Rx/RxPy (constructive approach)

```
# A counts every second starting from 0
A = Observable.interval(1000)

# B delays A by 0.5 seconds
B = A.delay(500)

# C sums the latest values from A and B
C = A.combine_latest(B, lambda a, b: a + b)
```

Marble diagram

```
A stream: —0————————1————————2————————3————————>
B stream: ————————0————————1————————2————————3—>
C stream: ————————0——1——2——3——4——5——6—>
```

How/when is it useful?

Event-based channels ≈ reactive data streams

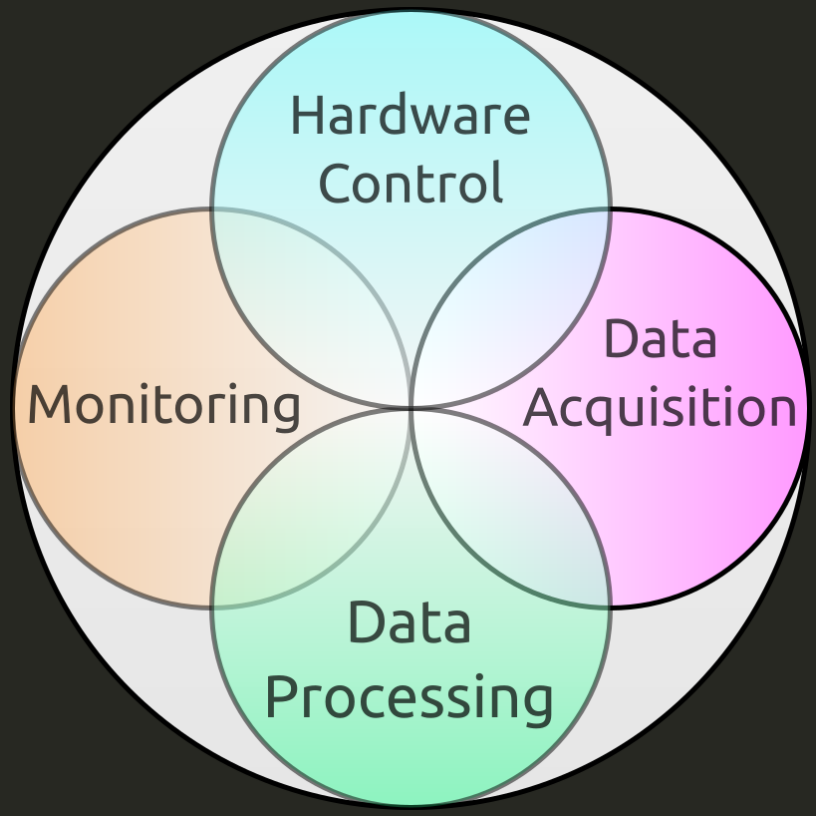
λ

Less state to manage → more functionnal, less side effect

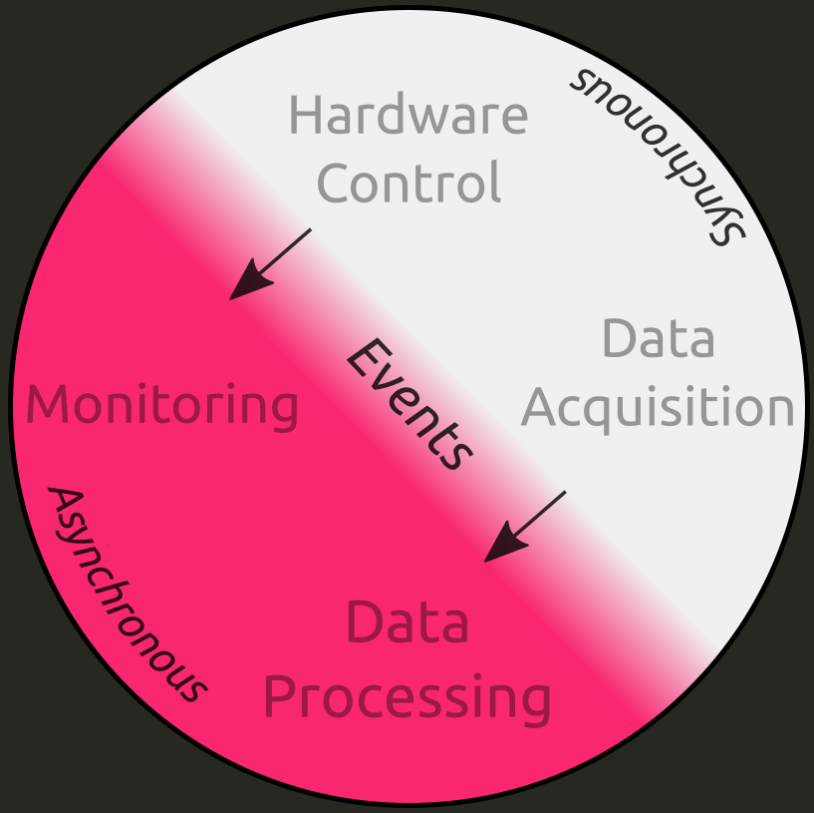
def

A declarative interface hides the implementation logic

What about control systems?



Where does reactive programming apply?



Monitoring and events

Golden rule

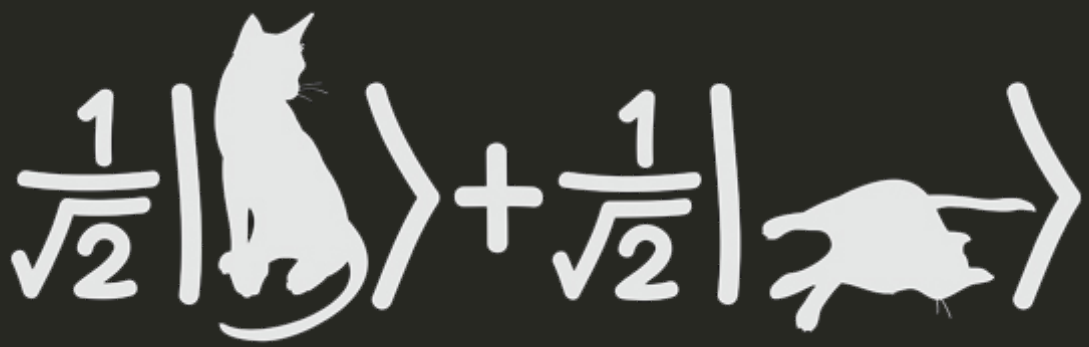
Monitoring shoudn't affect the world

Monitoring and events

Golden rule

Monitoring shoudn't affect the world

(unless your experiment includes a cat in a box)



Implications

The monitoring system **should never** trigger a hardware request

A system-agnostic service is managing and **protecting** the hardware

→ It does not care about the number of interested agents

How to get the hardware values then?

Reading from a cache is OK

But it introduces some **latency**

PUB/SUB is much nicer!

Should we give up on RPC?

REQ/REP is perfectly fine for running explicit commands

Because commands are the result of a **user decision**

However, the monitoring system **is not** a user

In practice, what can be done reactively?

Apply conversions, **e.g.** converting hardware units to SI

Integrate values, **e.g.** accumulating current to compute a charge

Combine values, **e.g.** creating logical conditions for the alarm system

Has this been implemented somewhere?

Yes!

3656 facade devices currently running at **MAX-IV**

•

Sensors: Temperature sensor, Flow gauge, Vacuum gauge, etc.

Actuators: Vacuum valve, Camera screen, Beam scraper, etc.

C := A + B strikes back!

```
class Addition(Facade):

    A = proxy_attribute([...])

    B = proxy_attribute([...])

    @logical_attribute(bind=['A', 'B'])
    def C(self, a, b):
        return a + b
```

•

The library is available on GitHub

MaxIV-KitsControls/tango-facadedevice

Documented

tango-facadedevice.readthedocs.io

Full tutorial, API reference and examples

Unit-tested

travis-ci.org/MaxIV-KitsControls/tango-facadedevice

100% of code coverage :)

Released

pypi.org/project/facadedevice

v1.0.1

Thank you!

Questions ?

Presentation written in **Markdown** and rendered by **remark**

Sources and examples on **GitHub**

vxgmichel/icalepcs-reactive-programming