Towards a "Synchronous Reactive" UML Profile?

Robert de Simone

Charles André

Motivation

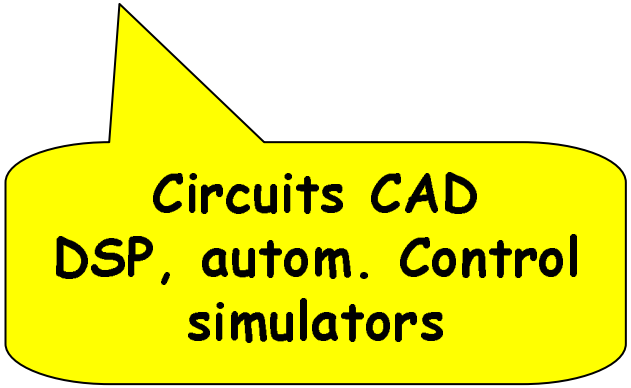
- NOT adhere to UML current semantics
- Promote the synchronous paradigm:
  - Natural in several fields: HW, block-diagrams, …
  - Sound Mathematical Models
  - Language constructs: precise semantics
  - Compilation = Model transformation
- NO run-to-completion semantics (simulation) with non-synthetizable programs

Strategy

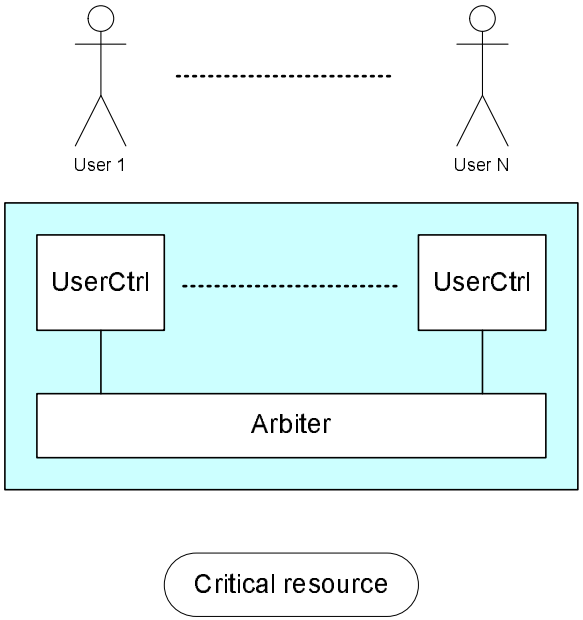
- REUSE some of the many diagrams of the UML
- Provide scientific arguments to people willing to lobby the OMG (Thales, CEA)

Where S/R should be used

- Applications:
  - Embedded controller, HMI, HW/SW, …
- Formalisms:
  - Strictly synchronous
    - SCADE, Esterel Studio, Block Diagrams
  - Almost synchronous (but "simulation" sem.)
    - Statecharts, VHDL/Verilog, Simulink, Scicos

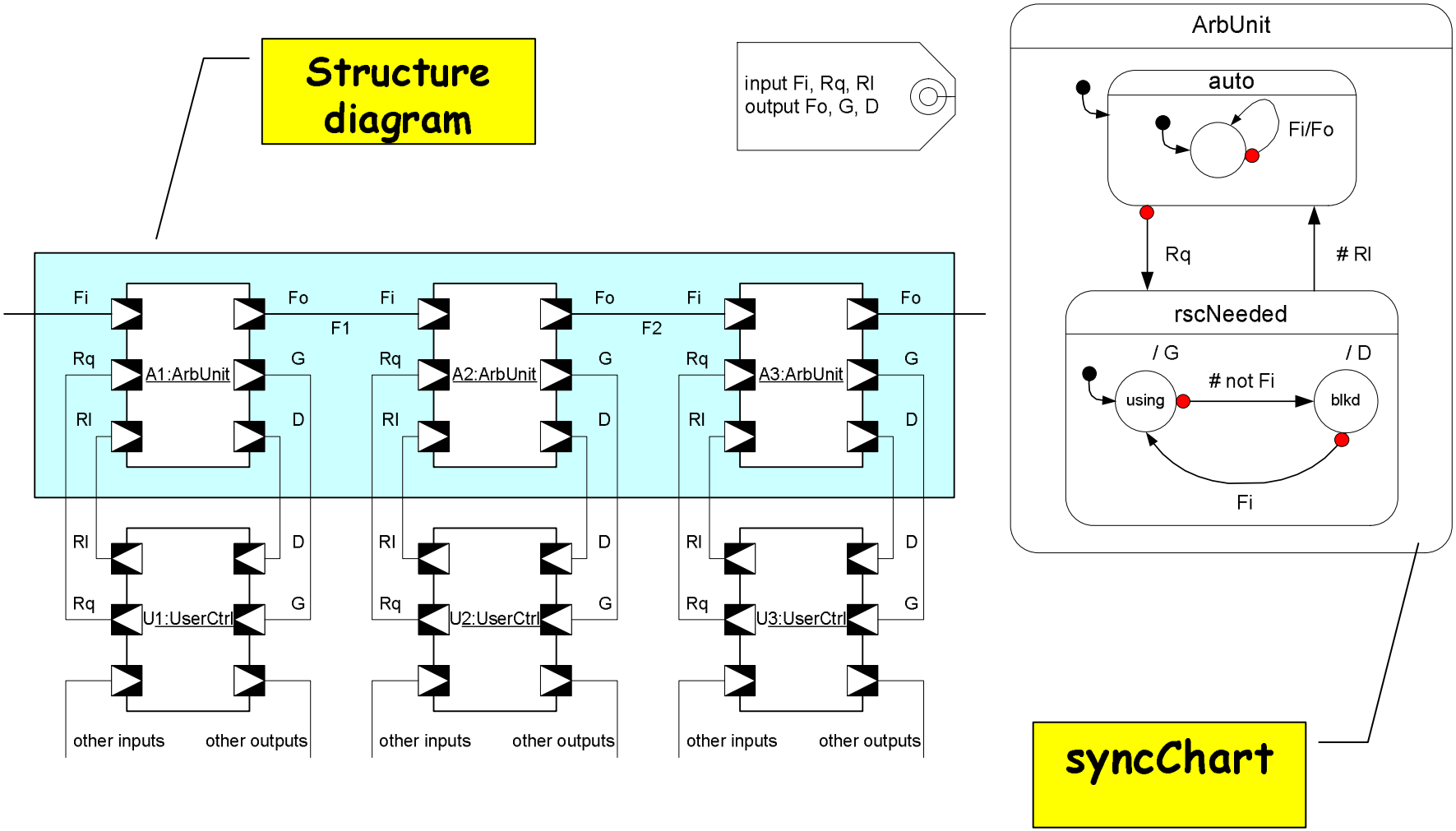


Example of an Arbiter

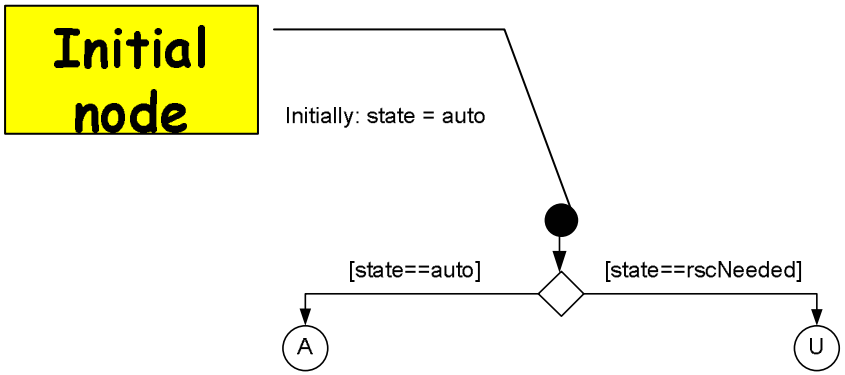


- Users: Rq and Rl
- Arbiter: G, D
- Exclusive access
- Static priority
- D valued with the nb of candidates for the resource with higher priority
- Linear description vs. the nb of users

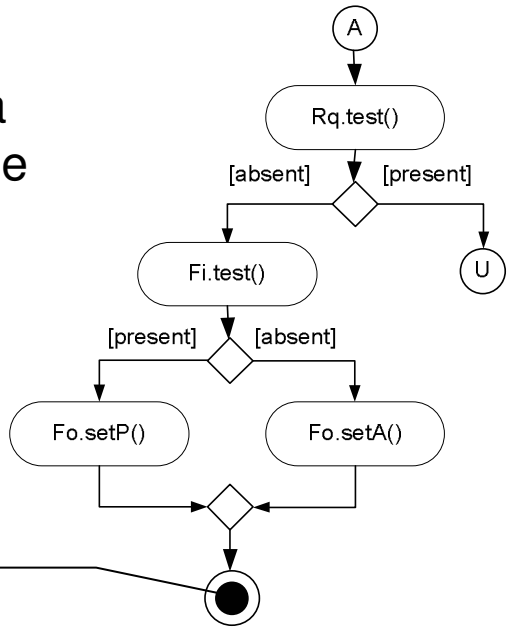
State-based synchronous modeling

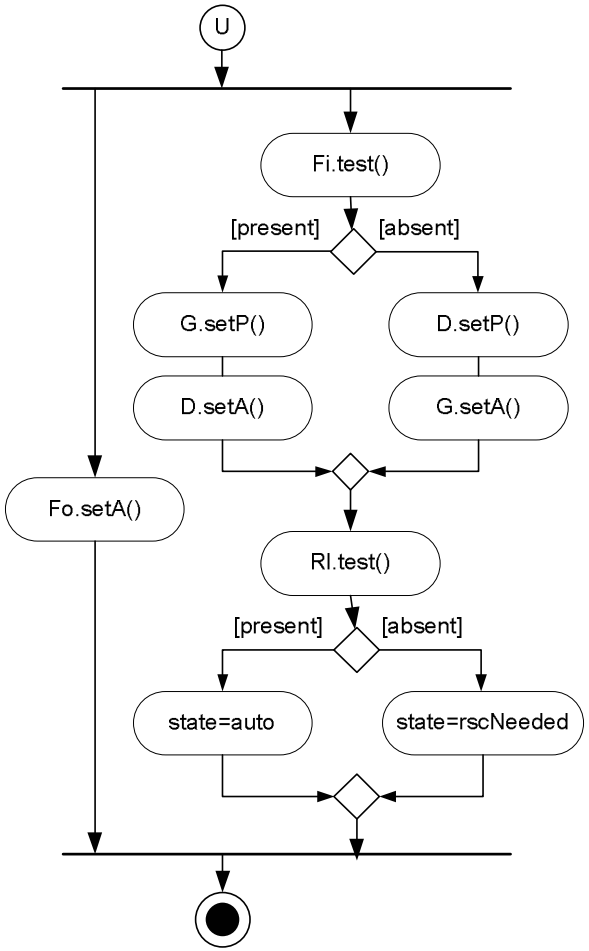


Activity-oriented approach



Activity diagram for a reaction of the ArbUnit





Activities from initial to final within one instant

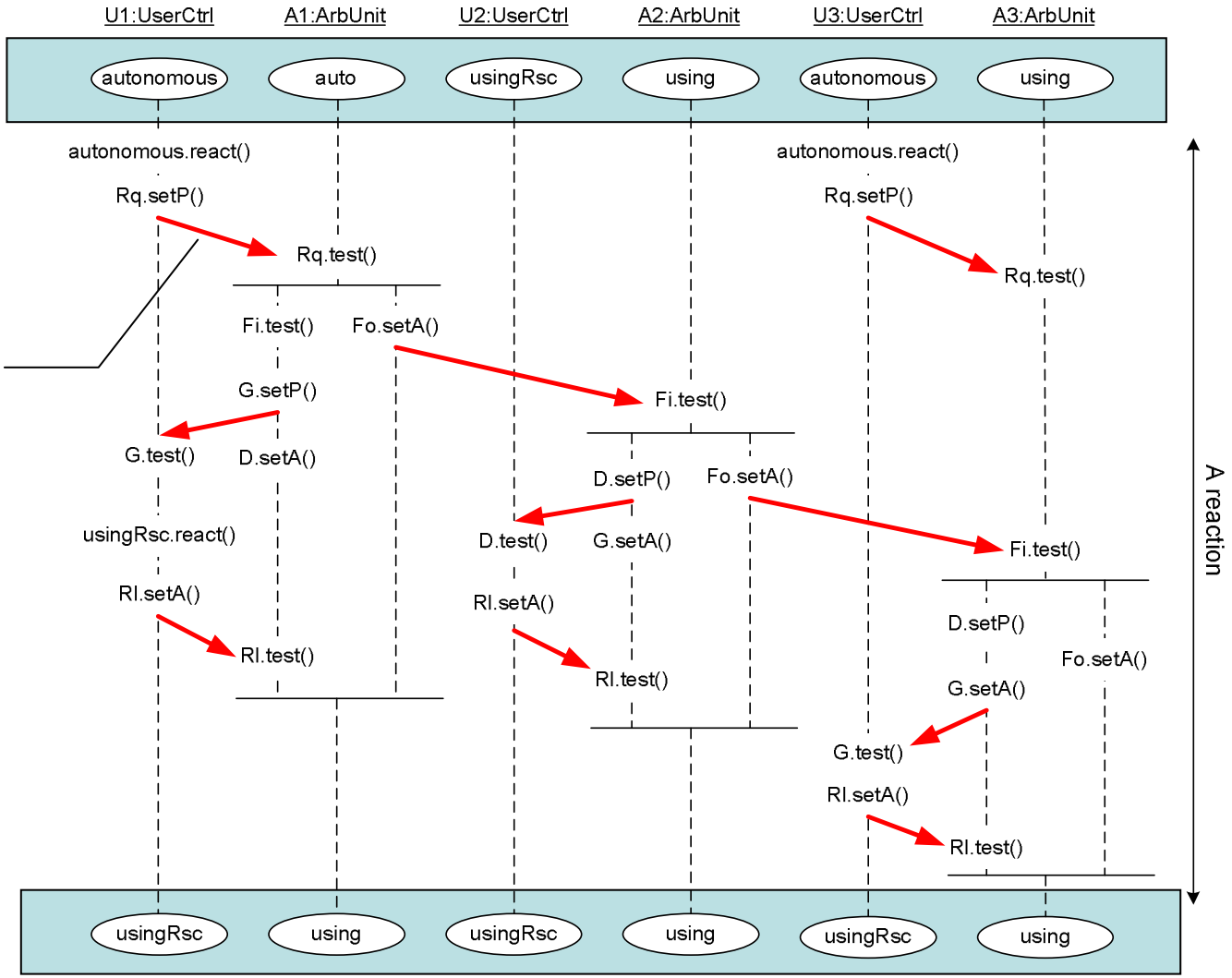
Emergent behavior

Finite and acyclic at each instant

Causality relation

A signal may be tested only after being set

Stable configuration



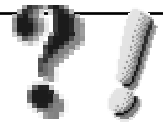
Limitations of the UML State Machines

- Object-based variant of statecharts
- Semantics described in terms of operations of a hypothetical machine
- Event queue + dispatcher
- Events are dispatched and processed one at a time
- Run-to-completion assumption
- Poor support for concurrency

Synchronous hypotheses

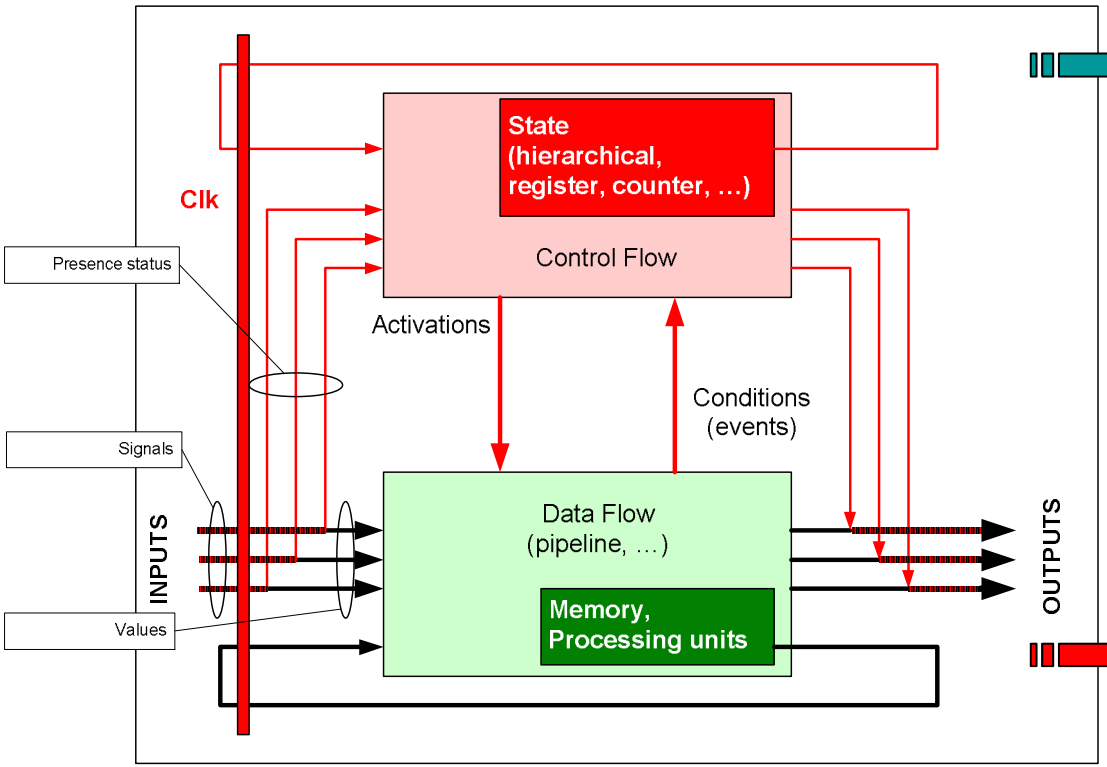
S/R stands for Synchronous Reactive

- Logical division of time into instants
- At each instant: execute a synchronous cycle (a reaction)
  - Acquisition
  - Compute (a **global** run-to-completion)
  - Actuate
- Signals are the unique support for communication

	UML	S/R
Events	•One at a time	•Several •Combination
Execution Semantics	•(Local) run-to-completion	•Distributed consensus (instant)
Parallelism	•Serialized concurrency	•Genuine simultaneity
Reaction to the absence		•Usual

Needs: a clear notion of instant

Control-flow / Data-flow



Esterel/SyncCharts:

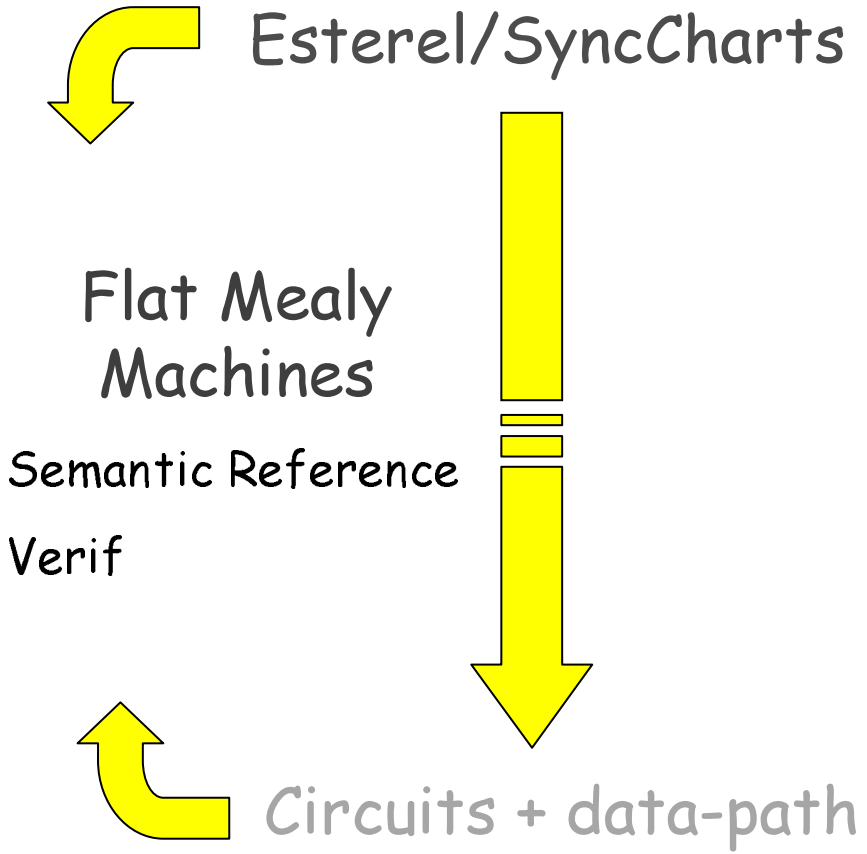
state-oriented

Combination ?

Lustre/Scade:

activity-oriented ?

Control-flow <-> Data-flow



Preponderant control-flow

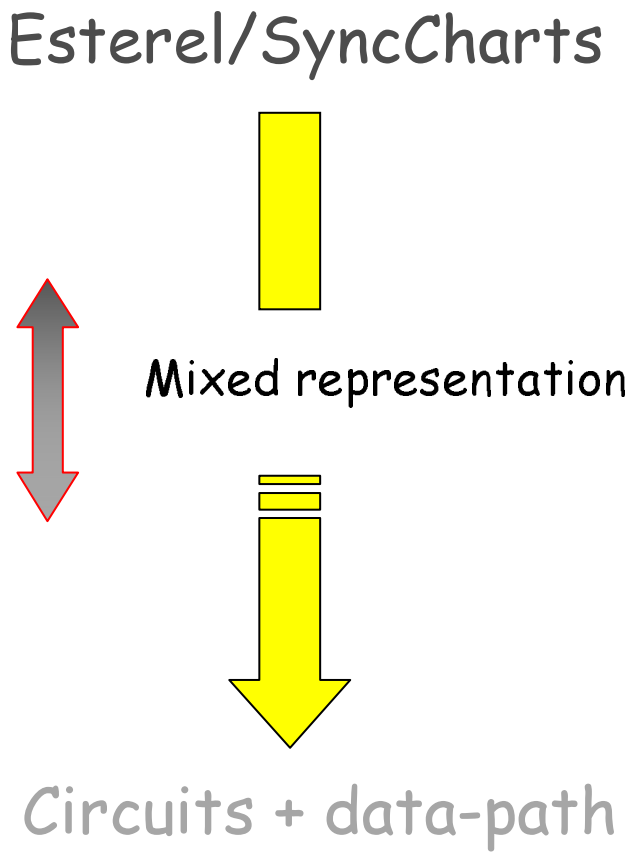
Imperative algorithmic style

Data-path: implicit & dynamic

Flat block-diagrams

Control= 0/1 Boolean values on wires

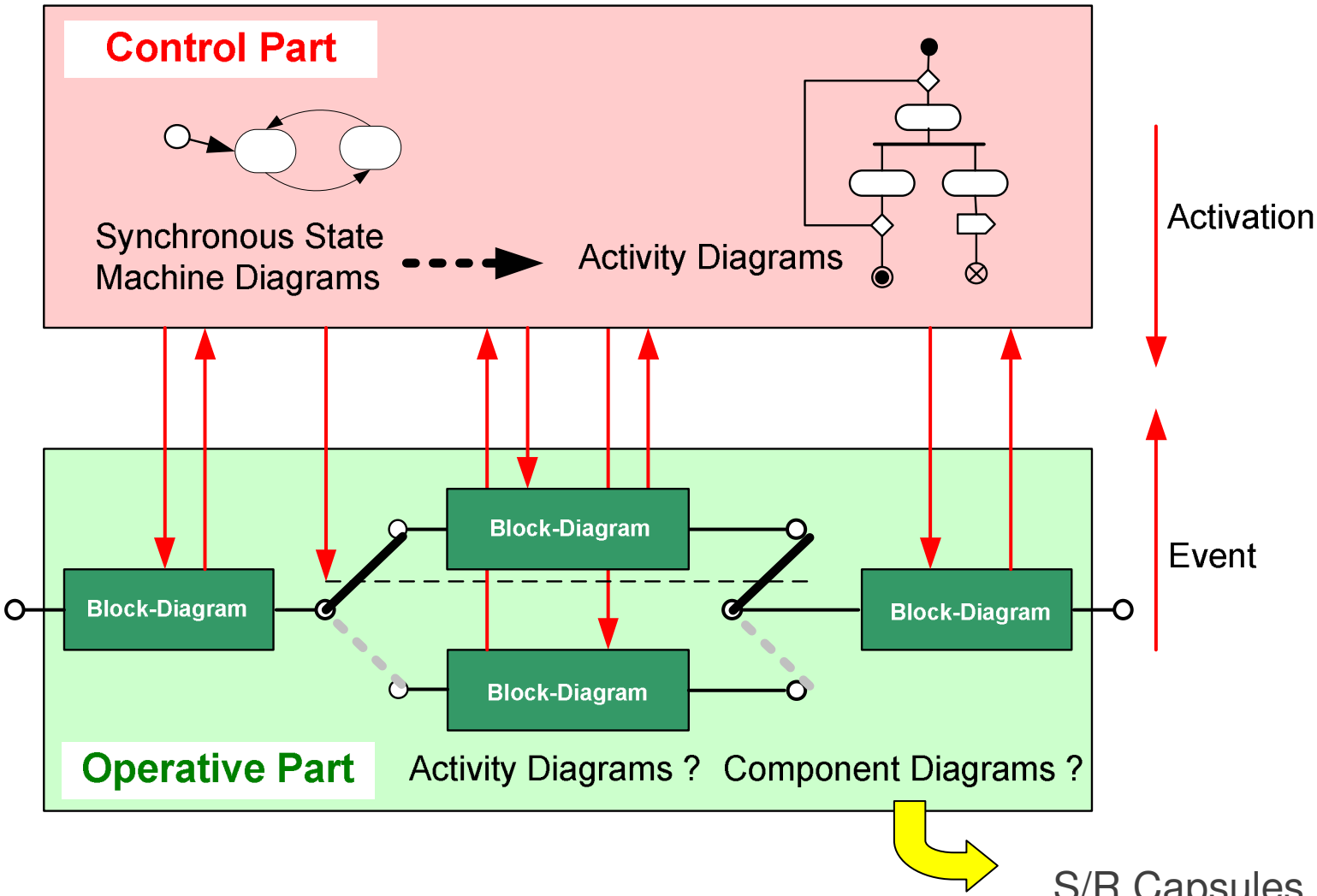
Control-flow <-> Data-flow



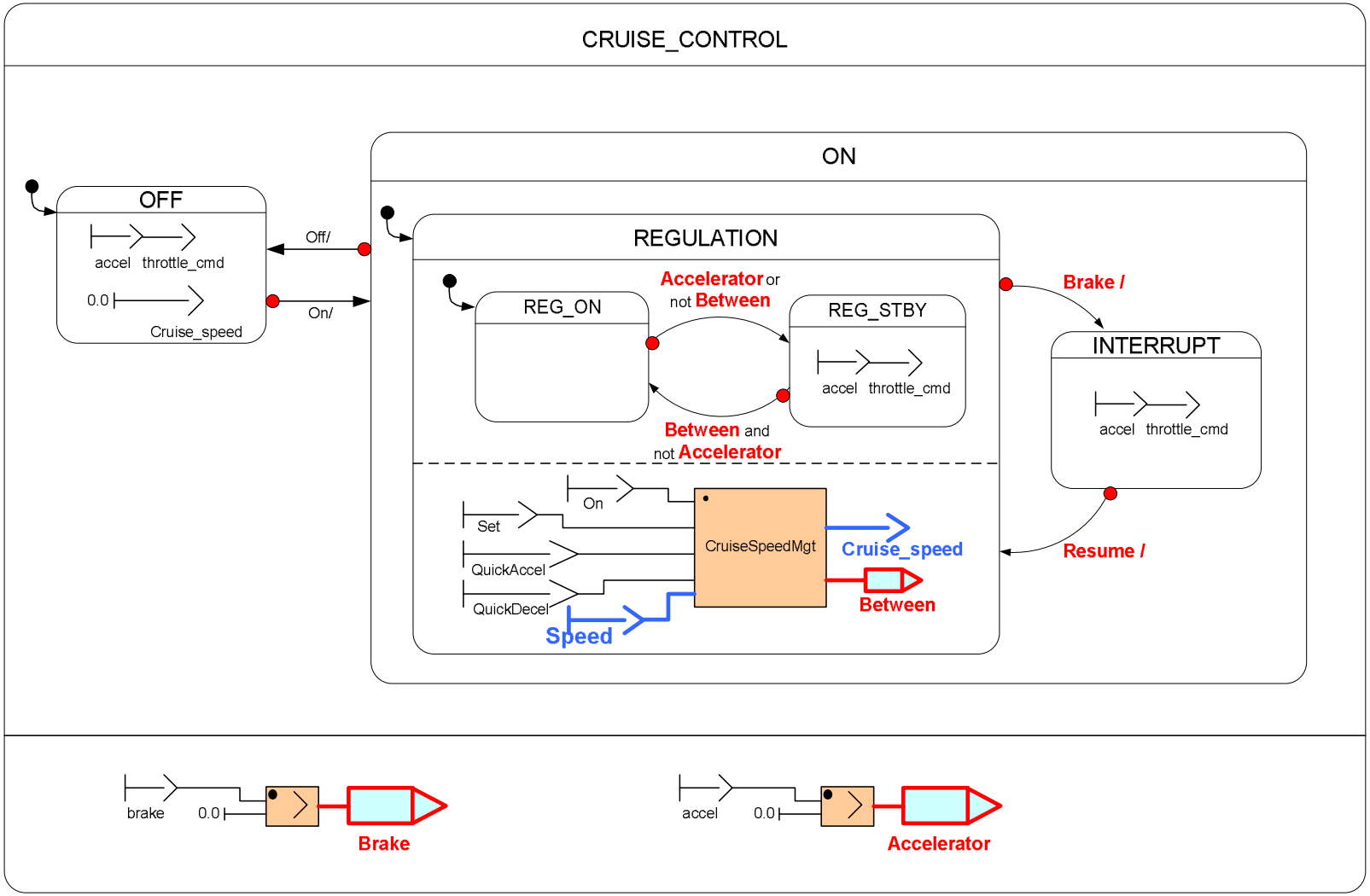
Explicit state registers (Control + Data)

Hierarchy: control network drives block-diagram networks

S/R Mixed Architecture



Dream Diagrams



Courtesy of J-L Colaço (E.T.)

Conclusion

- "Control system engineering" has specific needs, not all addressed by the UML which focuses on asynchronous threads and middleware
- The S/R approach answers some of them
- Try to subvert UML semantics for this domain.
- Formalisms closely linked to mathematical models: natural MDA approach (not new for us!)