CPS: Small:
Reconciling Safety with the Internet for Cyber-physical Systems

PI: Edward A. Lee, UC Berkeley
Student: Shaokai Lin
Postdoc: Marten Lohstroh

REACTORS

REACTORS are reactive software components that are composed out of reactions, which may be triggered by events produced internally (actions) or originating from other reactors (inputs). A reactor may contain other reactors and manage their connections. Connections define the flow of events, and two reactors can be connected only if they are contained by the same reactor.

EVENTS are timestamped, and reactions are triggered by them in timestamp order. Because reactions have to declare the ports they access, a deterministic execution schedule can be derived purely based on this readily available dependency information. Reactions are logically instantaneous.

DEADLINES require that when an input arrives at the last link in a chain of reactions triggered by an action, the difference between current physical time and the timestamp of the action is less than the specified maximum delay.

PHYSICAL ACTIONS are assigned a timestamp equal to the current physical time. Combined with deadlines, they allow for the specification and enforcement of end-to-end real-time constraints between sensors and actuators.

LINGUA FRANCA

LINGUA FRANCA (LF) is our coordination language for the definition and composition of reactors. LF is polyglot, and intended to be used with a variety of target languages. An LF program is deterministic unless the reactions (written in the target language) explicitly introduce nondeterminism, for instance, by reporting readings from some I/O device.

THE LF COMPILER generates target code that brings declared ports and actions into reaction scope. It constructs a precedence graph that governs the execution of reactions at any given time step. The toolchain is built using Xtext and features a syntax-directed editor that runs within Eclipse. Command-line tools are available as well.

TIME is a first-class citizen in LF; it allows for the specification of delays and deadlines.

FEDERATED EXECUTION allows reactors to interact through a network stack. We can either use a central coordinator or leverage fully distributed safe-to-process analysis known from PTIDES [1] and Spanner to preserve the deterministic reactor semantics (provided there are bounds on network latency and clock synchronization error).

EXAMPLE: DRIVE-BY-WIRE SYSTEM

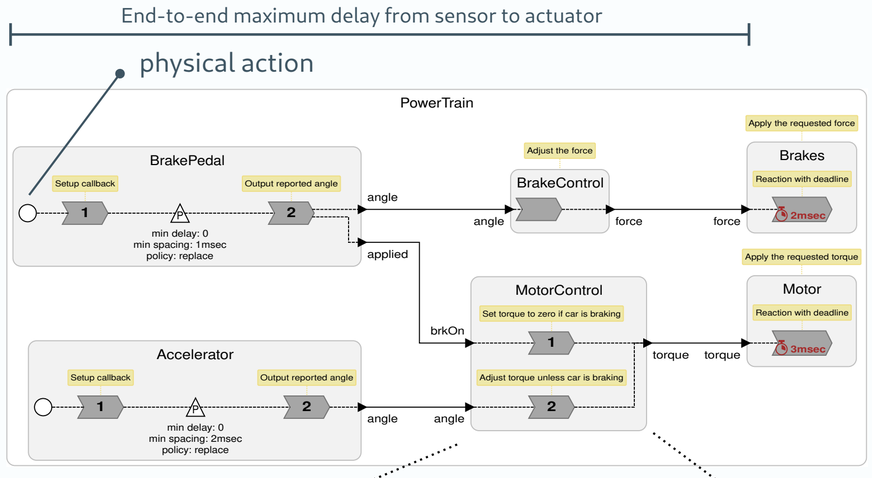

Fig. 1: Model of a Power Train

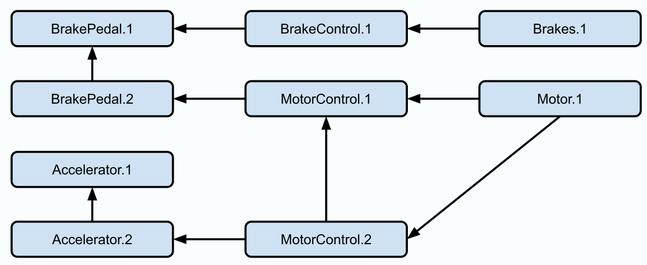

Fig. 2: Dependencies between reactions

- Dependencies between reactions are captured in the **reaction graph**, which must be acyclic.
- Scheduled events are ordered by timestamp; physical time must match the timestamp before reactions triggered by the event are loaded onto the reaction queue.
  - Pending reactions are ordered based on their location in the reaction graph. A reaction is not allowed to execute until all inputs that it depends on are known (i.e., preceding reactions have finished executing). Observing this constraint allows us to exploit parallelism exposed in the reaction graph, without relinquishing determinism.
  - A thread pool is used to efficiently handle rapidly succeeding reactions.

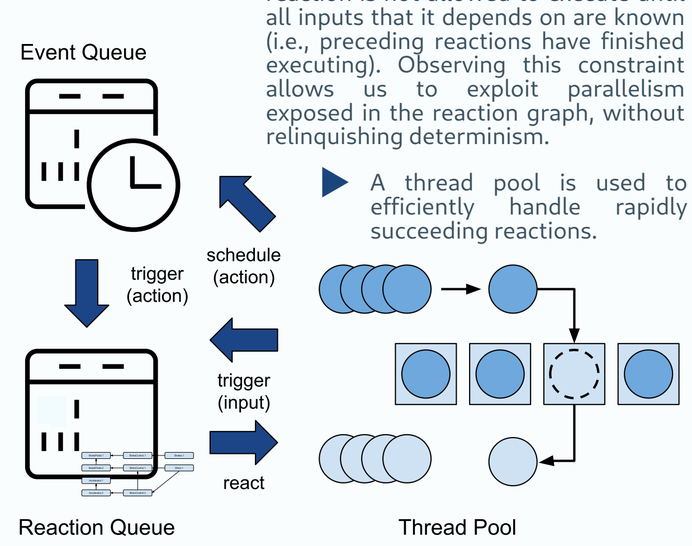

Fig. 4: Execution of reactor program

```
target C {threads: 1, keepalive: true, flags: "-lncurses"};

reactor MotorControl {
    input brkOn:bool;
    input angle:int;
    output torque:int;
    state braking:bool(true);

    //@label Set torque to zero if car is braking
    reaction(brkOn) -> torque {=
        self->braking = brkOn->value;
        SET(torque, 0);
    =}

    //@label Adjust torque unless car is braking
    reaction(angle) -> torque {=
        if (!self->braking) {
            SET(torque, calc_motor_torque(angle->value));
        } else if (angle->value > 0){
            printw("Cannot accelerate; release brake pedal first.\n");
        }
    =}
}

reactor Accelerator {
    state pedal:int(-1);
    physical action a(0, 2 msec, "replace"):int;
    output angle:int;
    state last:int(0);

    // @label Setup callback
    reaction(startup) -> a {=
        pedals.accelerate = a;
        init_sensors();
    =}

    // @label Output reported angle
    reaction(a) -> angle {=
        if (self->last != a->value) {
            SET(angle, a->value);
            self->last = a->value;
        }
    =}
}
```

Definition of a reactor class
Inputs and outputs, iff declared in the reaction signature, accessible through local variables in reaction code
State shared by all reactions are accessible through the `self` struct
Creation of event on output port
Triggers
Effects
Reaction bodies must be written in C; the LF compiler will invoke the C compiler after it has completed code generation
Types are inherited from the target language (LF has its own built-in type for time)
Reading the value of an input port
Start delimiter of target code
Function definitions and/or `#includes` can be put in preamble (not shown)
End delimiter of target code
State initialization
Any event scheduled on this action will acquire a tag relative to the current physical time
If min. spacing is violated, use the value of the last event
Enforce a minimum spacing between subsequent events
Process events without delay
If `startup` and a are present simultaneously, this reaction will execute first because it is declared first
Function declared in preamble

Fig. 3: LF implementation of `MotorControl` reactor

TARGETS

LF TARGETS can be added with moderate effort because target code in an LF file is not parsed or analyzed but embedded verbatim into the generated code. Supporting a new target only requires implementing a **reactor runtime** and a **code generator**:

Targets currently supported or under development:

- C;
- C++;
- TypeScript;
- Python; and
- Rust.

C, POSIX, PTHREADS

Our most mature target is C, and it uses POSIX primitives to obtain system time and manage threads of execution. The runtime is small (~3K LOC) and lightweight (up to 23 million reactions per second on a single core of a 2.6 GHz Intel Core i7). It features memory management for non-primitive event payloads (structs, arrays) and implements an earliest-deadline-first (EDF) scheduling policy.

Currently under development:

- support for run-time mutations;
- better syntax for expressing common patterns of parallel computation (e.g., map/reduce); and
- pluggable scheduling to tune the runtime engine to specific kinds of workloads; and
- bare-iron FlexPRET (RISC-V) support.

PRECISION-TIMED HARDWARE

- While reactors can guarantee determinism on conventional general-purpose hardware, ruling out the possibility of deadline violations requires a sound worst-case execution time (WCET) analysis of all reactions in the critical path of a deadline. This can be done much more accurately on platforms that are designed to yield predictable timing.
- With WCET carried out in the LF compiler, meeting timing constraints would be as simple as specifying them in the program. If the program compiles successfully, this means that the computed schedule is feasible, and execution of the program is guaranteed to satisfy the constraints.
- The first precision-timed hardware platform we're targeting is Patmos [2], which is well supported by several WCET tools. FlexPRET [3], which distinguishes between soft and hard real-time threads, is a particularly well-suited target for reactors.

[1] Y. Zhao, E. A. Lee, and J. Liu, A programming model for time synchronized distributed real-time systems in Real-Time and Embedded Technology and Applications Symposium (2007).
[2] Schoeberl, M., Puffitsch, W., Hepp, S., Huber, B., and Prokesch, D. Patmos: A time-predictable microprocessor. Real-Time Systems 54(2) (Apr 2018), 389–423.
[3] Zimmer, M., Broman, D., Shaver, C., and Lee, E. A. FlexPRET: A processor platform for mixed-criticality systems. In Real-Time and Embedded Technologyand Application Symposium (2014).

CHECK OUT OUR
GITHUB.COM /ICYPHY/LINGUA-FRANCA

NSF CPS PI Meeting, June 3-4, 2021
Award No: 1836601
iCyPhy.ORG

This work was supported in part by the National Science Foundation (NSF), award #CNS-1836601 (Reconciling Safety with the Internet) and the iCyPhy Research Center (Industrial Cyber-Physical Systems), supported by Denso, Ford, Siemens, and Toyota.