A Distributed FRP Language with an Actor-based Execution Model

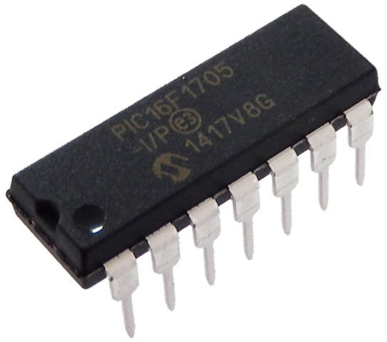
Kazuhiro Shibanai & Takuo Watanabe
Department of Computer Science, Tokyo Institute of Technology

May. 2019
Programming Languages for Distributed Systems (PL4DS)
Shonan Meeting #149

About This Work

- Distributed pure FRP language with an Actor-based execution model
- Goal
    - To provide high-level declarative abstraction for networked devices
        - coordination language / macroprogramming language for WSN
    - To support incremental development by providing a uniform way to express the whole (intra- & inter-device) behavior of a distributed system
    - (To provide a formal specification / verification framework for secure / reliable IoT systems)

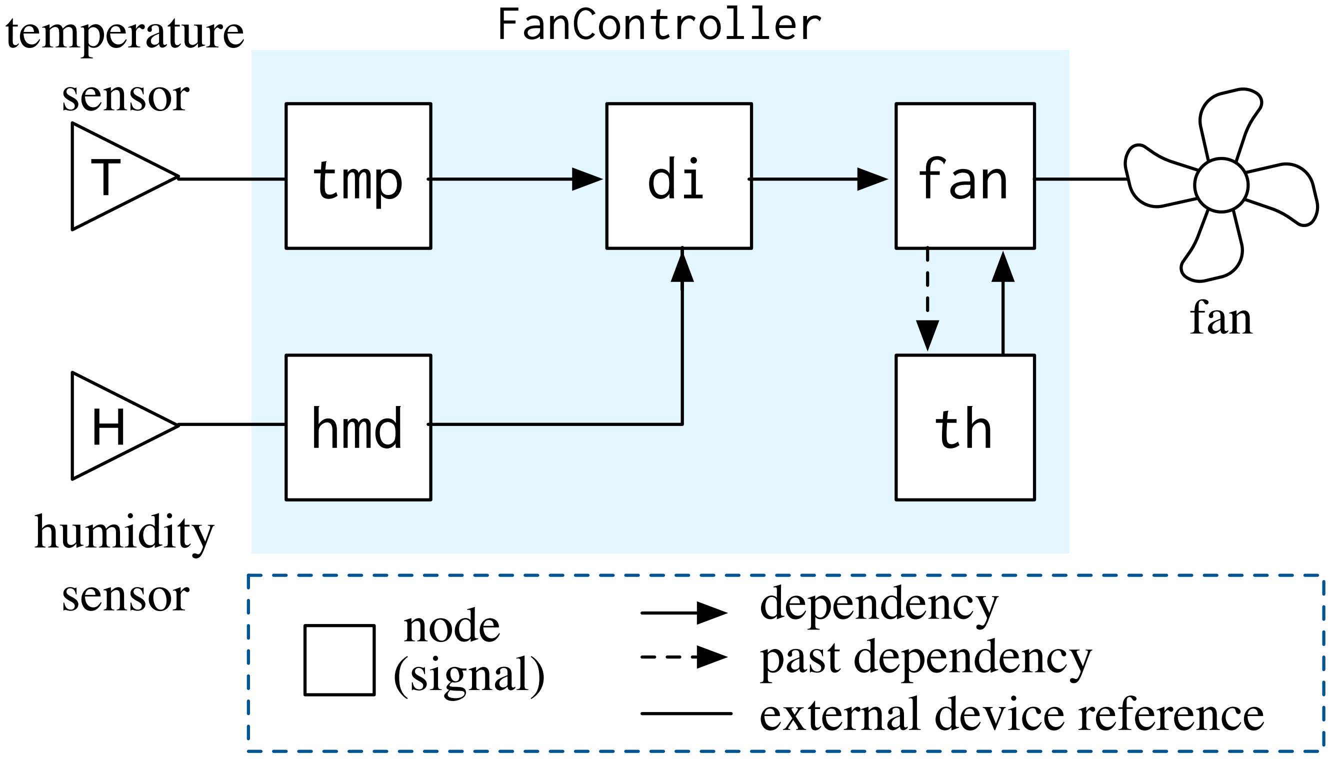
Emfrp [Sawada & Watanabe '16]



- FRP language for resource-constrained systems
  - Strongly-typed, purely functional
    - parametric polymorphism, type-inference, pattern matching
  - Simple abstraction for time-varying values (signals)
    - named, non-first-class representation, lifting-free
  - Small, statically bounded amount of runtime memory
    - syntactic restrictions & type system
- Implementation
  - Compiler to C
    - http://github.com/psg-titech/emfrp or "gem install emfrp"
  - Works for several microcontrollers
    - 8-32 bit MCU (16MHz-), RAM 2.5KB-, Flash 32KB-
    - ex) Microchip PIC/AVR, ARM Cortex-M, Xtensa LX6 (ESP32)

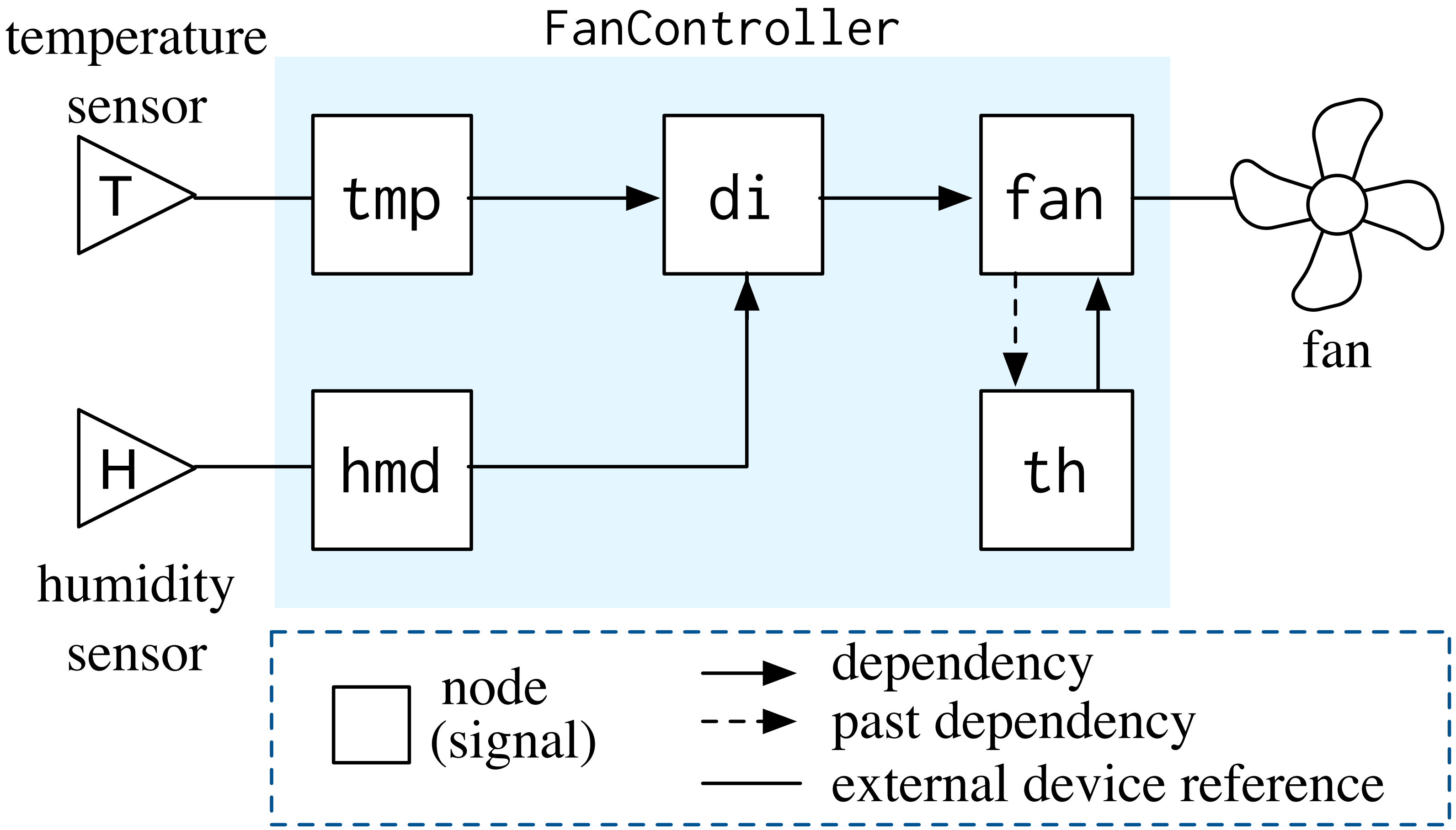
Simple Example

```
module FanController
in  tmp : Float, # temperature
    hmd : Float  # humidity
out fan : Bool   # fan switch

# discomfort index
node di = 0.81 ∗ tmp + 0.01 ∗ hmd
    ∗ (0.99 ∗ tmp − 14.3) + 46.3

# fan switch
node init[False] fan = di >= th

# threshold
node th = 75.0 +
  if fan@last then −0.5 else 0.5
```



- A fan controller with environmental sensors
- Turns a fan ON while the current discomfort index >= 75
- Does a simple hysteresis control to avoid frequent switching

Execution Model

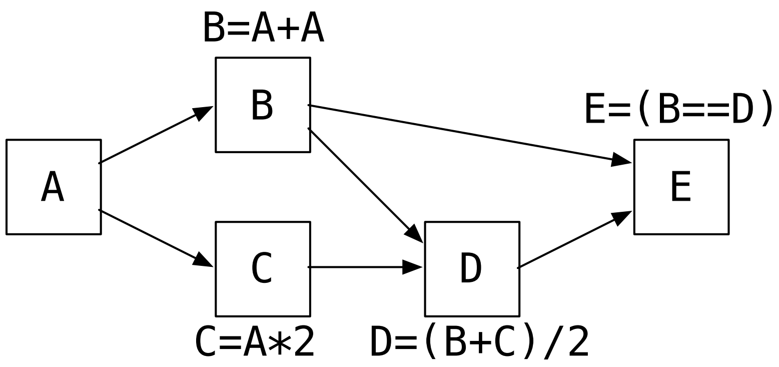


1. Read the temperature sensor and update tmp
2. Read the humidity sensor and update hmd
3. Update di
4. Update th using the past value of fan
5. Update fan and controls the fan switch

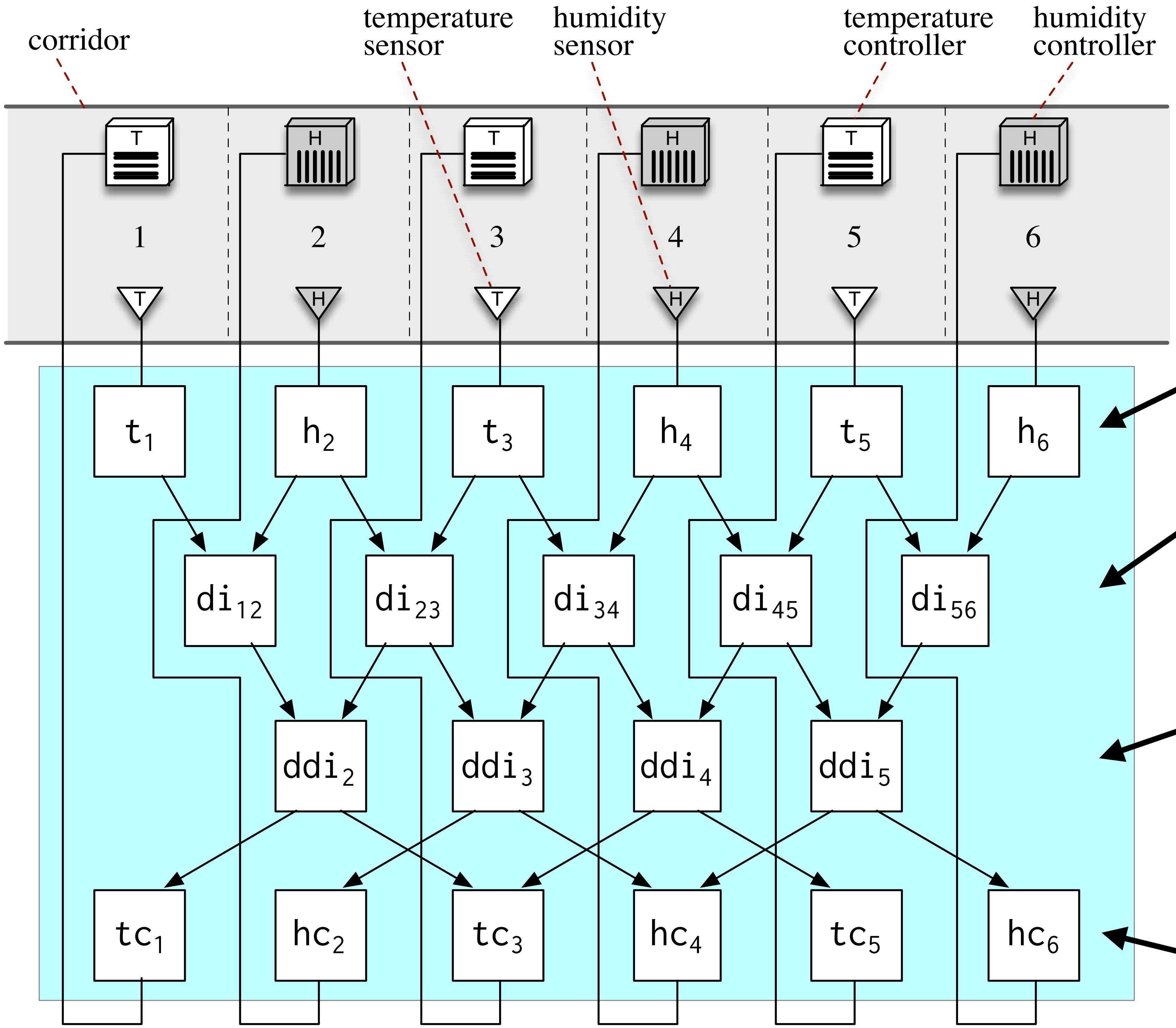
- Push-based sequential execution model
    - A program is represented as a DAG of signals and their dependencies
    - The runtime system repeatedly updates signals along the topologically-sorted DAG

Distributed-XFRP [Shibanai & Watanabe '18]

- A pure FRP language for networked devices
  - Syntax/semantics: similar to Emfrp
  - Actor-based execution model
    - A signal is represented by an actor
    - Provides asynchronous nodes (for heavy computation)
  - Signal updating algorithm
    - Guarantees single-source glitch-freedom
      - using a message versioning similar to DREAM
    - Handles out-of-order messages delivery
    - Handles simple message losses
  - Prototype compiler (to Erlang)
    - https://github.com/45deg/distributed-xfrp



Example: WSAN for Air-Regulation



t_i (or h_i) measures the air temperature (or humidity)

$di_{i\ i+1}$ calculates the discomfort index using t_i and h_{i+1} (or h_i and t_{i+1})

ddi_i checks the degree of imbalance between $di_{i-1\ i}$ and $di_{i\ i+1}$

tc_{i-1} (hc_{i-1}) or tc_{i+1} (hc_{i+1}) activates the temperature (humidity) controller according to ddi_i

Related Work

- DREAM [Margara et al '14, '18]
  - Support several glitch-freedom levels (none to complete)
- REScala [Salvaneschi et al '14][Mogk et al '18]
  - Guarantees GF by ACKs, Handles partial failures
- ScalaLoci [Weisenburger et al '18]
  - placement types, multitier reactive abstractions
- AmbientTalk/R [Carreton et al '10]
  - RP for unreliable networks
- QPROP, QPROP^d [Myter et al '19]
  - Decentralized complete GF, supports partial failures
- Actor-Reactor Model [Van den Vonder et al '17]
  - Separation of declarative reactor parts and imperative actor parts

Future Direction

- Fault-Tolerance
  - Fault-handling behaviors in declarative manner
- Dynamic Modification / Adaptation
  - Current model relies on the static construction of DAG
    - cf. COP for Emfrp [Watanabe '18]
- Support for incremental development
  - local to distributed, AOP-like abstraction
- Formal Specification / Verification
- Dealing with Time
- Relationships to Control Theory
- More efficient implementation for small devices
  - memory footprint, power consumption
- FRP for GPGPU, FRP for FPGA