Software Design for Cyber-Physical Systems

Edward A. Lee

Module 4: Time in Lingua Franca

Technical University of Vienna
Vienna, Austria, May 2022
Cyber-Physical Systems: A Huge Modeling Challenge

Focus on the role of time
What is Time?

- Change: Aristotle
- Smooth: Newton
- Relative: Einstein
- Discrete: Wheeler
What is Time?

Muller: Gives a theory of time that requires big black holes to collide somewhere near us to test it.

Rovelli: “The nature of time is perhaps the greatest remaining mystery.”
How can we build systems based on something we do not understand?
Deterministic Timing

Is our goal for our models to accurately reflect the timing of our implementation?

or

Is our goal for the implementation to accurately reflect the timing of the model?

If it’s the latter, then deterministic timing makes perfect sense!
Desirable Properties in a Model of Time

- A “present” that separates the past and future
- Support for causality
- A well-defined “observer”
- A notion of “simultaneity”

All are problematic in physics but useful in models.
There is no ground truth on the order in which events occur.

Einstein’s train: https://youtu.be/wteiuxyqtoM

Physically separated state machines.

Transition system model.
What is Time?

Aristotle  Newton  Einstein  Wheeler

Change  Smooth  Relative  Discrete

All of these are about scientific models, not engineering models.
The goal is for the implementation to accurately reflect the timing of the model.

So what should the model be?
Logical Time

```
main reactor Main {
    ramp = new Ramp();
    delay = new Delay();
    print = new Print();
    ramp.y -> delay.x;
    delay.y -> print.x;
}

reactor Ramp {
    timer t(0, 100 msec);
    output y:int;
    state count:int(0);
    reaction(t) -> y {
        SET(y, self->count);
        self->count++;
    }
}

reactor Print {
    input x:int;
    reaction(x) {
        printf("Logical time: %lld, Physical time %lld ", Value: %d\n",
            get_elapsed_logical_time(),
            get_elapsed_physical_time(), x->value);
    }
}

reactor Delay {
    logical action a(50 msec):int;
    input x:int;
    output y:int;
    reaction(a) -> y {
        SET(y, a->value);
    }
    reaction(x) -> a {
        schedule_int(a, 0, x->value);
    }
}
```

```
```

Diagram:
- Ramp
- Main
- Delay
- Print

Flow:
- Ramp
- Main
- Delay
- Print

Ramp (0, 100msec) → y → x → 2 → 50msec → 1 → y → x → Print
Logical and Physical Time

[marten@yoga Delay]$ lfc Delay.lf
 filename: Delay
 sourceFile: /home/marten/git/lingua-franca/example/Delay/Delay.lf
 directory: /home/marten/git/lingua-franca/example/Delay
 mode: STANDALONE
 Generating code for: file:/home/marten/git/lingua-franca/example/Delay/Delay.lf
 In directory: /home/marten/git/lingua-franca/example/Delay
 Executing command: gcc -O2 src-gen/Delay.c -o bin/Delay
 Code generation finished.
 [marten@yoga Delay]$ bin/Delay
 Start execution at time Mon Sep 14 14:18:59 2020
 plus 601126676 nanoseconds.
 Logical time: 500000000, Physical time 50096786, Value: 0
 Logical time: 1500000000, Physical time 150099592, Value: 1
 Logical time: 2500000000, Physical time 250123369, Value: 2
 Logical time: 3500000000, Physical time 350128015, Value: 3
 Logical time: 4500000000, Physical time 450088289, Value: 4
 Logical time: 5500000000, Physical time 550136789, Value: 5
 Logical time: 6500000000, Physical time 650144220, Value: 6
 Logical time: 7500000000, Physical time 750147670, Value: 7
 Logical time: 8500000000, Physical time 850124282, Value: 8
 Logical time: 9500000000, Physical time 950089670, Value: 9
 Elapsed logical time (in nsec): 1,000,000,000
 Elapsed physical time (in nsec): 1,000,130,940
 [marten@yoga Delay]$
Time and Timers

This page is showing examples in the target language C. You can change the target language in the left sidebar.

Logical Time

A key property of Lingua Franca is logical time. All events occur at an instant in logical time. By default, the runtime system does its best to align logical time with physical time, which is some measurement of time on the execution platform. The lag is defined to be physical time minus logical time, and the goal of the runtime system is maintain a small non-negative lag.

The lag is allowed to go negative only if the fast target property or the --fast is set to true. In that case, the program will execute as fast as possible with no regard to physical time.

In Lingua Franca, time is a data type. A parameter, state variable, port, or action may have type time. In the C target, time values internally have type instant_t or interval_t, both of which are (usually) equivalent to the C type long long.
Asynchronous External Events

```c
reaction(startup) -> response {=
  pthread_t thread_id;
  pthread_create(&thread_id, NULL,
                  &read_input, response
  )
  printf("Press Enter to produce a"
         "sensor value.\n");
} =
reaction(response) -> y =
  printf("Reacting to physical "
         "action at %lld\n",
          get_elapsed_logical_time());
  SET(y, true);
} =
```

```
reactor Sensor {
  preamble {=
    void* read input(void* response) { =
      //...
    }
  } =
  output y:boolean;
  physical action response;
}
```
Logical and Physical Actions

Physical Actions

A **physical action** is used to schedule reactions at logical times determined by the local physical clock. If a physical action with delay \( d \) is scheduled at physical time \( T \), then the logical time assigned to the event is \( T + d \). For example, the following reactor schedules the physical action \( p \) to trigger at a **logical time** equal to the **physical time** at which the input \( x \) arrives:

```c
#define T
#define d

reactor Physical {
    input x:int;
    physical action a;
    reaction(x) -> a { =
        lf_schedule(a, 0);
    }
    reaction(a) { =
        interval_t elapsed_time = lf_time_logical_elapsed();
        printf("Action triggered at logical time %ld nsec after start.\n", elapsed_time);
    }
}
```

If you drive this with a timer, using for example the following structure:

```
PhysicalTest
```

then running the program will yield an output something like this:

```
Action triggered at logical time 201491000 nsec after start.
Action triggered at logical time 403685000 nsec after start.
Action triggered at logical time 605969000 nsec after start.
...```

---

![ LF logo ](https://example.com/lf-logo.png)
Reflex Game

- Reaction Ordering
- Causality Loops
- Physical vs. Logical Time
Causality Loops

If you reverse the order of reactions in the RandomSource, you get a causality loop error.

Why?
Deadlines

Deadlines

\[ T < t + 500 \text{ usec} \]

\[ T > t + 500 \text{ usec} \]

WCET?
Preempting Execution

Sieve of Eratosthenes

Anytime computation
Possible Deadline Variants

Implemented:

• Lazy deadline
• Cooperative lazy deadline (lf_check_deadline)

Not implemented:

• Eager deadline (Issue #1006)
• Preemptive deadline (kill) (Issue #403)
• Lag trigger (Issue #1006)
target C;
main reactor {
    state count:int(1);
    logical action a;
    reaction(startup, a) -> a {=
        printf("%d. Logical time is %lld. Microstep is %d.\n", 
                self->count, lf_tag().time, lf_tag().microstep 
        );
        if (self->count++ < 5) {
            lf_schedule(a, 0);
        }
    } =}
1. Logical time is 1649607749415269000. Microstep is 0.
2. Logical time is 1649607749415269000. Microstep is 1.
3. Logical time is 1649607749415269000. Microstep is 2.
4. Logical time is 1649607749415269000. Microstep is 3.
5. Logical time is 1649607749415269000. Microstep is 4.
Logical simultaneity is a key concept in Lingua Franca.

Logical Simultaneity

Two events are **logically simultaneous** only if both the logical time and the microstep are equal. The following example illustrates this:

```c
target C;
reactor Destination {
    input x:int;
    input y:int;
    reaction(x, y) {
        printf("Time since start: %lld, microstep: %d\n", 
            lf_time_logical_elapsed(), lf_tag().microstep);
        if (x->is_present) {
            printf(" x is present.\n");
        }
        if (y->is_present) {
            printf(" y is present.\n")
        }
    }
}
main reactor {
    logical action repeat;
    d = new Destination();
    reaction(startup) -> d.x, repeat {
        lf_set(d.x, 1);
        lf_schedule(repeat, 0);
    }
    reaction(repeat) -> d.y {
        lf_set(d.y, 1);
    }
}
```
Teaser: Distributed Rhythm

EALMAC:- eal$ cd git/examples-lingua-franca/C
EALMAC:C eal$ bin/RhythmDistributed_player1
Federate 0: Connected to RTI at localhost:15045.
----- Start execution at time Sat May 7 16:05:51 2022
----- plus 570091000 nanoseconds.
Federate 0: ---- Using 19 workers.

EALMAC:- eal$ CD git/examples-lingua-franca/C
EALMAC:C eal$ bin/RhythmDistributed_player2

EALMAC:C eal$ RTI -n 2
RTI: Number of federates: 2
Starting RTI for 2 federates in federation ID Unidentified Federation
RTI using TCP port 15045 for federation Unidentified Federation.
RTI: Listening for federates.

Install the project with no errors.
Teaser: Distributed Rhythm

Basic control:
- x: quit
- +: speed up
- -: slow down

Instrument:
- 0: none
- 1: bass drum
- 2: bongo
- 3: claves
- 4: conga
- 5: cowbell
- 6: cuica
- 7: guiro
- 8: snare
- 9: tom

Rhythm:
- d: down beat
- m: merengue
- b: bossa nova
- s: samba

Federate 0: REMOTE: Changing rhythm to merengue.
Federate 0: Changing instrument to 3.

Federate 1: Changing instrument to 1.
Federate 1: Changing rhythm to merengue.

Starting RTI for 2 federates in federation ID Unidentified Federation
RTI using TCP port 15045 for federation Unidentified Federation.
RTI: Listening for federates.
RTI: All expected federates have connected. Starting execution.
RTI: Waiting for thread handling federate 0.
Conclusion

Concepts:

• Logical time vs. physical time(s)
• Multiple timelines
• Superdense time