Actors, Time Stamps, and Determinism for Time-Critical Systems

Marten Lohstroh
marten@eecs.berkeley.edu
UC Berkeley
USA

Martin Schoeberl
masca@dtu.dk
TU Denmark
Denmark

Andrés Goens
andres.goens@tu-dresden.de
TU Dresden
USA

Armin Wasicek
armin.wasicek@avast.com
Avast
USA

Christopher Gill
cdgill@wustl.edu
Washington University in St. Louis
USA

Marjan Sirjani
marjan.sirjani@mdh.se
Mälardalen University
Sweden

Edward A. Lee
eal@eecs.berkeley.edu
UC Berkeley
USA

ABSTRACT

Abstract

CCS CONCEPTS

• Computer systems organization → Embedded systems; **Embedded systems**;

KEYWORDS

actor, real-time systems, worst-case execution time

ACM Reference Format:
Marten Lohstroh, Martin Schoeberl, Andrés Goens, Armin Wasicek, Christopher Gill, Marjan Sirjani, and Edward A. Lee. 2019. Actors, Time Stamps, and Determinism for Time-Critical Systems. In *Proceedings of The 56th ACM/ESDA/IEEE Design Automation Conference (DAC '19).* ACM, New York, NY, USA, **??** pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Since their introduction by Hewitt in the early 70s, the use of actors has proliferated in programming languages [? ? ?], coordination languages [? ?], distributed systems [? ?], and simulation engines [? ?]. Actors have much in common with objects—a paradigm focused on reducing code replication by means of inheritance and increasing modularity via data hiding—but unlike objects, actors also provide a model for *concurrency*. Indeed, in the original formulation of the actor model, each actor is presumed to operate concurrently alongside other actors that it may exchange messages with, asynchronously. The lack of any guarantees with respect to the ordering of messages, and the absence of a notion of time, makes this a very general model, but not a very useful model for the specification of systems in which timely execution and determinism are important.

DAC '19, June 2–6, 2019, Las Vegas, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 PRINCIPLES

A1 computate on private data A2 send events to output ports A3 request invocation of a procedure at t + delta A4 read any inputs & either get a value or null A5 access current time == time stamp of input A6 every actor sees events in time stamp order A6 request an async, callback of proc (what is current time?) A7 TR delay

3 LOGICAL TIME & PHYSICAL TIME

3.1 Deadlines

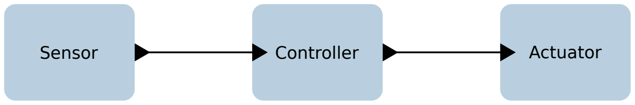

Figure 1: First example

3.2 Causality

We don't want to run ahead of realtime, because actors can produce spontaneous events that need to be stamped with pysical time, and we don't want these timestamps to be "in the past."

3.3

Shut off the lights some time after the switch has been flipped. Reason to have the deadline definition as stated: detectability. Suppose the start deadline cannot be met; the reaction should not be carried out (and the violation be reported on subsequently).

4 RELATED WORK

Martin: Is this the first Actor paper? Carl Hewitt; Peter Bishop and Richard Steiger (1973). "A Universal Modular Actor Formalism for Artificial Intelligence". IJCAI.

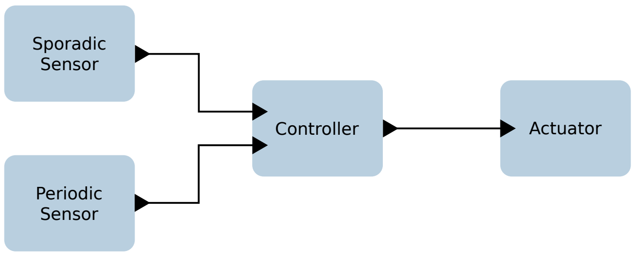

Figure 2: Second example

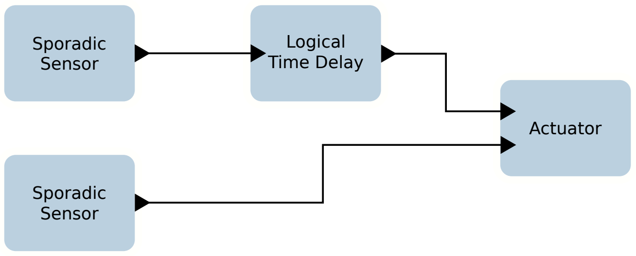

Figure 3: Third example

5 ACTORS

Several frameworks have been built based, loosely or strictly, on the original Hewitt actor model [? ?], including Akka [?], Ray [?], and Rebeca [?]. One property of the original Hewitt actor model is that messages received by an actor from distinct sources are handled in nondeterministic order. Rebeca performs analysis on the model using model checking to ensure that this nondeterminism does not lead to behaviors that violate requirements of the application [?]. In Lingua Franca, we take a different approach, which is to ensure that messages are handled in deterministic order unless the model explicitly requires nondeterminism. By default, all Lingua Franca programs will be deterministic. We use a semantic notion of time and time stamps to achieve this.

Actors communicate via ports. All data exchanged has a timestamp. An actor does not advance time, the reaction is considered instantaneous.

Reactions of one actor are atomic and are allowed to change state. Reactions are triggered by available inputs with timestamps at logical time (or older). When several reactions of an actor are enabled at the same time, the definition order in LF defines the execution order.

A reaction of an actor can also be released by a trigger. A trigger can be periodic triggers or a single shot timer.

Martin: We need to check this: Outputs generated by a time triggered reaction are timestamped. When the system is simulated, the timestamp is equal to the model time. When the system is used for execution, the logical time for the release of the reaction is synchronized to wall clock time and therefore the output message are synchronized to wall clock time.

Martin: Question: what happens when a reaction depends on two inputs, but those two inputs have a different logical time? I assume that the reaction is fired/released when all two inputs have values (which means logical time is at the younger message) and a produced output gets the younger timestamp.

Martin: We have (and maybe support) two options: (1) input is only valid when the timestamp equals logical time, absent at other times or (2) keeping the value of an input for future logical time and having a default value for system initialization time. (in the JS accessors this is the way it is specified, the default value gives the "keep" or persistent semantic).

5.1 Scribbles on Logical Time

Time is *logical* time. This logical time be used for simulation and then we call it model time. We can also use the network of actors for the implementation of the system. In that case wall clock time is used to timestamp input values at sensors. Logical time can never advance further than wall clock time. We have a notion of delay in two places: (1) as a delay actor to break up feedback loops (2) at actuators to specify that an actuation shall happen before (or exactly at) the logical time of the input plus the delay.

The scheduler invokes reactions of actors dependent in inputs with timestamps that are equal (or earlier–can this happen?) at logical time t_1. Logical time cannot advance further than wall clock time t_w. If next logical time $t_2 > t_w$ the scheduler waits till $t_w \geq t_2$.

5.2 Sporadic Events and Call Backs

Sporadic events and call backs can trigger a reaction. This reaction is allowed to observe and also change the state of the actor. Therefore, they are not allowed to preempt a reaction (reactions are atomic to each other). To enforce this atomicity a sporadic reaction is executed only between two logical timestamps t_1 and t_2. This can be enforced on a C based host by turning off interrupts when executing a reaction. Interrupts are enabled when all[1] reactions for timestamp t_1 have been executed and disabled again when logical time is advanced to t_2 where all reactions for t_2 are executed.

TODO: Describe that the single threaded execution of JavaScript enforces the restriction of the call back not interrupting the reaction.

5.3 More Scribbles

Delays deserve also the purpose to align events with wall clock time and to allow execution of reactions in their worst-case execution time (WCET). The summ of all reaction's WCET from one synchronization point (either sample of sensors or output of a delay actor) to another synchronization point (or output of an actuator) has to be less than the delay.

5.4 Dynamic Actors

For dynamic systems, e.g., an IoT that keeps running, but in different contexts (e.g., places), we support dynamic reconfiguration of the network of actors. This may be well supported when targeting a dynamic language such as JavaScript, but harder to implement in C.

To support reconfiguration we need a more complete language then the configuration language LF. Therefore, our first iteration of dynamic reconfiguration is done in the host language JavaScript.

[1] *TODO: All is a little bit restrictive, just for state atomicity we would not need to wait for all reaction, just for a single. But there is another issue that this shall happen only between two timestamps. I don't recall the details.*

In that case, the host language needs access to actors and ports. However, we can use scopes to having access to those parts of the framework only in well defined places.

5.5 Call of an External Service

An actor can call an external service that will deliver the result in the future and use a call back to deliver the result (Martin: Marten likes to use a Future). This call back is handled as described above between two model timestamps and the return value is timestamped with wall clock time.

An actor my implement a timeout mechanism by scheduling a trigger in the future. **This timeout TO need to be set relative to wall clock time, not logical time, as logical time will be behind wall clock time.**

6 CONCLUSION

Acknowledgment