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, Germany

Armin Wasicek
armin.wasicek@avast.com
Avast, USA

Christopher Gill
cdgill@wustl.edu
Washington Univ., St. Louis, USA

Marjan Sirjani
marjan.sirjani@mdh.se
Mälardalen Univ., Sweden

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

ABSTRACT

Abstract

CCS CONCEPTS

• **Computer systems organization → 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, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Precision timing plays an important role in a plethora of modern systems, ranging anywhere from embedded control systems that continually interact with concurrent physical processes (i.e., cyber-physical systems) to large-scale distributed systems requiring some measure of consistency. In order to effectively program these systems, there is a need for programming models with a semantics that includes time. In current-day general-purpose hardware and programming languages, timing properties of software are emergent–not specified. Therefore, the verification of timing properties of time-critical systems often relies on rigorous testing, but effectively testing software in the face of non-determinism is extremely challenging. We propose an actor-oriented programming model that reduces non-determinism by introducing a semantic notion of time, allowing programmers to specify timing properties, which, if executed on capable hardware, can be guaranteed statically.

In this paper, we propose an actor-oriented programming model for time-critical systems, based on a discrete event semantics, which ensures that messages between actors are handled in deterministic order unless nondeterminism is introduced explicitly as a desired property. We introduce *Lingua Franca* (LF), an interface definition language for the description of actors and their composition. LF allows programmers to implement the functionality of actors using a language of their choice. By default, all Lingua Franca programs will be deterministic. We use a semantic notion of time and time stamps to achieve this.

2 ACTORS

The actor model was introduced by Hewitt [9] in the early 70s. Since then, the use of actors has proliferated in programming languages [2, 4, 8], coordination languages [1, 18], distributed systems [10, 14], and simulation engines [17, 21]. Actors have much in common with objects—a paradigm focused on reducing code replication by means of inheritance and increasing modularity via data encapsulation—but unlike objects, actors also provide a model for *concurrency*. Indeed, each actor is presumed to operate concurrently alongside other actors that it may exchange messages with, asynchronously. These properties make actors an ideal for programming reactive systems. 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, albeit a less useful one for the specification of systems in which timely execution and determinism are important.

Extra machinery can be introduced for the formal specification and analysis of systems composed of Hewitt actors. For instance, Real-time Maude [16], a timed rewriting logic framework and (temporal) model checking tool, has been applied to actors in [5]. Similarly, the modeling language Rebeca performs analysis that uses a model checker to ensure that nondeterminism allowed in the model does not lead to behaviors that violate requirements of the simulated system [20]. Alternatively, constraints can be placed on actors' allowable behaviors so that they adhere to a well-defined model of computation (MoC), satisfying desirable properties such

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

as deadlock freedom, schedulability, bounded memory usage, and deterministic execution, by construction. It is the latter approach we follow.

3 LINGUA FRANCA

In Lingua Franca, rather than addressing each other directly, actors exchange messages via ports. This level of indirection allows actors to be entirely agnostic of the presence or absence of their receiving counterparts. The connections between actors are embedded in a level of hierarchy—a composite—that is tasked with relaying messages between contained actors. While this approach increases modularity, more importantly, it exposes dependencies between actors that can be used to devise a schedule that ensures i) actors observe produced messages in time stamp order, and ii) actors cannot produce messages with time stamp t until all antidependent inputs with time stamp t are known. This is the approach taken in the Discrete Event implementation of Ptolemy II [11], which is formally based on a synchronous reactive MoC TODO ▶*[Benveniste,Berry,1991]*◀, but, to increase efficiency, it averts performing a fixed-point computation by executing actors in topological order. That way, each actor is only executed once per tick of the logical clock. Because the schedule is based purely on the topology of the actor graph, no static analysis of the actors' internals is required—Ptolemy actors are treated as black boxes.

LF actors, on the other hand, are better characterized as *gray* boxes considering they reveal some key features of their internal structure, namely a closer approximation of the true dependencies between their ports. Other than actors in Ptolemy II, which implement an abstract semantics [22] that requires each actor to perform all of its computation in a so-called *fire* function, LF actors are broken down into smaller units we call *reactions*. This approach, inspired by the characterization of dataflow processes given in [12], exposes the dependencies between actors' inputs and output at a finer grain. We enforce simple lexical scoping rules, which virtually any programmer is already familiar with, to limit reactions' access to the actors' input and output ports, thereby eliminating dependencies between ports that are out of scope.

The interface definitions of reactions are much like function definitions. Consider the following example which defines a brake component as an LF-actor:

```
language ECMA6/Flow;
actor Brake {%
  input check:any
  input apply:number
  output on:boolean
  preamble {%
    state = false
    actuate() { ... }
  %}
  reaction(apply) -> {%
    pressed.set(true)
    if (apply > 5)
      actuate(apply)
    else
      state = false
  %}
  reaction(check) -> on {%
    on = state
  %}
%}
```

TODO ▶*Explain this, discuss polyglot*◀ TODO ▶*Precisely because no static analysis is required...*◀

Having this finer-grained dependency information available at the interface level also opens up avenues for scheduling optimizations, makes the detection of zero-delay feedback less conservative, and, we conjecture, will enable remote procedure calls (RPCs) between actors without compromising determinism or modularity. The same dependency analysis performed to devise a schedule for message delivery and rule out zero-delay feedback can be used to coordinate RPCs and guarantee deadlock freedom. We think that integrating RPCs into LF will be key to achieving wide adoption of our programming model, because it is such a omnipresent language feature, even in popular actor-based frameworks such as Ray [14].

Central to the LF programming model is the relationship between the time stamps associated with the receipt of messages, which denote *logical* time, and the passing of *physical* time, approximated by the system time of the execution platform. While logical time can be sped up or slowed down arbitrarily in a simulation, the fact that actors can interact with the physical world through sensors and actuators requires that logical time and physical time "keep up" with one another in order to have a temporal semantics that obeys the principle of causality (i.e., effects cannot precede causes) for all observers.

Before discussing a number of motivating examples, we summarize our model in terms of the following principles:

(1) Messages exchanged between actors are timestamped;
(2) The arrival of a message denotes a discrete event;
(3) Actors carry input ports, triggers, output ports, state, and a list of reactions, each of which:
   (a) is triggered by the presence of an event on an input port (originating from another actor) or trigger (originating from the actor itself);
   (b) sees events in time stamp order;
   (c) can read/modify the actor's state;
   (d) can set the values of output ports it has access to;
   (e) can schedule *future* events on triggers it has access to;
   (f) can read the values of input ports it has access to;
      (i) the time stamp associated with an observed event always denotes the current logical time; and
      (ii) if no message arrives at a given port at the current logical time, its value is considered *absent*.
(4) No logical time elapses during a reaction;
(5) Any two reactions of the same actor execute atomically with respect to one another; and
(6) If two reactions of the same actor are triggered simultaneously, they execute in a predefined order;

Martin ▶*Some more things for the list: reactions that are ready have a release order according to their definition order (in LF).*◀ Marten ▶*See item (6)*◀ Martin ▶ *We have discussed two different input ports: those that are only valid during the logical time of the time stamp (what did we call them?) and the ones that latch the value and are valid in future logical time as well.*◀ Marten ▶*I think we called them input ports and triggers, respectively. See item (3)(a). If unclear, we should definitely improve the wording. Same for the terminology. We could also go for external/internal input ports. Internal ports are only referenced by the actor itself; both can be considered triggers...*◀

4 TIMING AND DETERMINISM

TODO ▶*discuss timing and scheduling using actor graph examples (no code)*◀

4.1 Delays & Deadlines

$\mathcal{D}$ We have a notion of delay in two places: (1) as a delay actor to break up feedback loops and (2) at actuators to specify that an actuation shall happen before (or exactly at) the logical time of the input plus the delay.

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 sum 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.

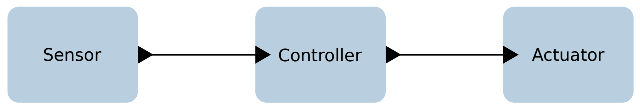

Figure 1: First example

4.2 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.

4.3 Causality

We do not 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."

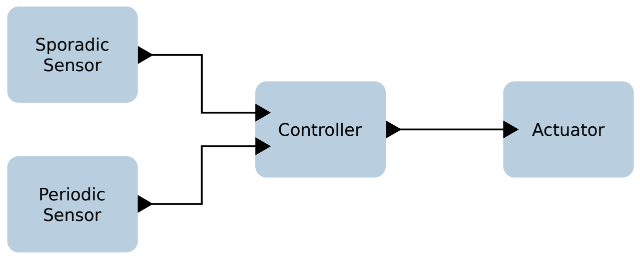

Figure 2: Second example

4.4 Example 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).

Marten ▶*In third example, discuss how persistent ports can be used.*◀

Marten ▶*Also, discuss optimized scheduler that runs ahead of physical time.*◀

[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.*◀

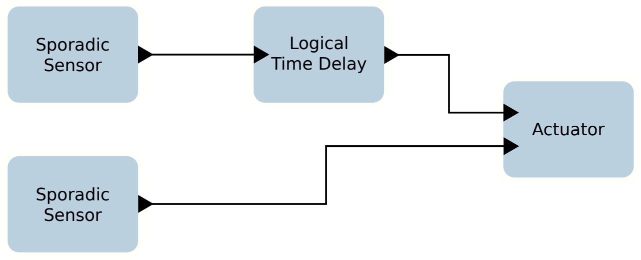

Figure 3: Third example

5 RELATED WORK

Towards a real-time coordination model for mobile computing [7] Method and tools for mixed-criticality real-time applications within PharOS [13] Accessors [3] S-Net [6] Guarded Atomic Actions [19] Timed C [15]

6 THE LINGUA FRANCA COMPILER TOOL CHAIN

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

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 than the configuration language LF. Therefore, our first iteration of dynamic reconfiguration is done in the host language JavaScript. 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.

7 CONCLUSIONS

Acknowledgments

The work in this paper 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 Avast, Camozzi Industries, Denso, Ford, Siemens, and Toyota

REFERENCES

[1] Farhad Arbab. 2004. Reo: A Channel-Based Coordination Model for Component Composition. *Mathematical Structures in Computer Science* 14, 3 (2004), 329–366.

[2] Joe Armstrong, Robert Virding, Claes Wikström, and Mike Williams. 1996. *Concurrent programming in Erlang* (second ed.). Prentice Hall.

[3] C. Brooks, C. Jerad, H. Kim, E. A. Lee, M. Lohstroh, V. Nouvellet, B. Osyk, and M. Weber. 2018. A Component Architecture for the Internet of Things. *Proc. IEEE* 106, 9 (September 2018), 1527–1542. DOI: http://dx.doi.org/10.1109/JPROC.2018.2812598

[4] Ankush Desai, Vivek Gupta, Ethan Jackson, Shaz Qadeer, Sriram Rajamani, and Damien Zufferey. 2013. P: safe asynchronous event-driven programming. *ACM SIGPLAN Notices* 48, 6 (2013), 321–332.

[5] Hui Ding, Can Zheng, Gul Agha, and Lui Sha. 2003. Automated Verification of the Dependability of Object-Oriented Real-Time Systems. In *2003 The Ninth IEEE International Workshop on Object-Oriented Real-Time Dependable Systems.* 171–171. DOI: http://dx.doi.org/10.1109/WORDS.2003.1267505

[6] Clemens Grelck, Sven-Bodo Scholz, and Alex Shafarenko. 2008. A gentle introduction to S-Net: Typed stream processing and declarative coordination of asynchronous components. *Parallel Processing Letters* 18, 02 (2008), 221–237.

[7] Gregory Hackmann, Christopher Gill, and Gruia-Catalin Roman. 2005. Towards a real-time coordination model for mobile computing. In *Monterey Workshop*. Springer, 184–202.

[8] Philipp Haller and Martin Odersky. 2009. Scala actors: Unifying thread-based and event-based programming. *Theoretical Computer Science* 410, 2-3 (2009), 202–220.

[9] Carl Hewitt. 1977. Viewing control structures as patterns of passing messages. *Journal of Artificial Intelligence* 8, 3 (1977), 323–363.

[10] John Hunt. 2018. *Further Akka Actors*. Springer International Publishing, Cham, 345–360. DOI:http://dx.doi.org/10.1007/978-3-319-75771-1_32

[11] Edward A. Lee, Jie Liu, Lukito Muliadi, and Haiyang Zheng. 2014. Discrete-Event Models. In *System Design, Modeling, and Simulation using Ptolemy II*, Claudius Ptolemaeus (Ed.). Ptolemy.org.

[12] Edward A. Lee and Eleftherios Matsikoudis. 2009. *The Semantics of Dataflow with Firing*. Cambridge University Press. http://ptolemy.eecs.berkeley.edu/publications/papers/08/DataflowWithFiring/ Draft version available at http://ptolemy.eecs.berkeley.edu/publications/papers/08/DataflowWithFiring/.

[13] Matthieu Lemerre, Emmanuel Ohayon, Damien Chabrol, Mathieu Jan, and Marie-Benedicte Jacques. 2011. Method and tools for mixed-criticality real-time applications within PharOS. In *2011 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*. IEEE, 41–48.

[14] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, William Paul, Michael I. Jordan, and Ion Stoica. 2017. Ray: A Distributed Framework for Emerging AI Applications. *CoRR* abs/1712.05889 (2017). arXiv:1712.05889 http://arxiv.org/abs/1712.05889

[15] S. Natarajan and D. Broman. 2018. Timed C: An Extension to the C Programming Language for Real-Time Systems. In *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 227–239. DOI:http://dx.doi.org/10.1109/RTAS.2018.00031

[16] Peter Csaba Ölveczky and José Meseguer. 2008. The real-time Maude tool. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 332–336.

[17] Claudius Ptolemaeus. 2014. *System Design, Modeling, and Simulation using Ptolemy II*. Ptolemy.org, Berkeley, CA. http://ptolemy.org/books/Systems

[18] Shangping Ren, Yue Yu, Nianen Chen, Kevin Marth, Pierre-Etienne Poirot, and Limin Shen. 2006. Actors, Roles and Coordinators — A Coordination Model for Open Distributed and Embedded Systems. In *Coordination Models and Languages*, Paolo Ciancarini and Herbert Wiklicky (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 247–265.

[19] Daniel L. Rosenband and Arvind. 2004. Modular Scheduling of Guarded Atomic Actions. In *Proceedings of the 41st Annual Design Automation Conference (DAC '04)*. ACM, New York, NY, USA, 55–60. DOI:http://dx.doi.org/10.1145/996566.996583

[20] Marjan Sirjani and Mohammad Mahdi Jaghoori. 2011. Ten Years of Analyzing Actors: Rebeca Experience. In *Formal Modeling: Actors, Open Systems, Biological Systems - Essays Dedicated to Carolyn Talcott on the Occasion of Her 70th Birthday*. 20–56.

[21] Marjan Sirjani, Ali Movaghar, Amin Shali, and Frank S. de Boer. 2004. Modeling and Verification of Reactive Systems using Rebeca. *Fundam. Inform.* 63, 4 (2004), 385–410.

[22] Stavros Tripakis, Christos Stergiou, Chris Shaver, and Edward A. Lee. 2012. A Modular Formal Semantics for Ptolemy. *Mathematical Structures in Computer Science Journal* to appear (2012). http://chess.eecs.berkeley.edu/pubs/877.html