# Lifestate: Event-Driven Protocols and Callback Control Flow (Artifact)

**Shawn Meier**
University of Colorado Boulder, USA
https://plv.colorado.edu/shawn/
shawn.meier@colorado.edu

**Sergio Mover**
École Polytechnique, Palaiseau, France
http://www.sergiomover.eu/
sergio.mover@lix.polytechnique.fr

**Bor-Yuh Evan Chang**
University of Colorado Boulder, USA
https://www.cs.colorado.edu/~bec/
evan.chang@colorado.edu

## Abstract

Developing interactive applications (apps) against event-driven software frameworks such as Android is notoriously difficult. To create apps that behave as expected, developers must follow complex and often implicit *asynchronous programming protocols*. Such protocols intertwine the proper registering of callbacks to receive control from the framework with appropriate application-programming interface (API) calls that in turn affect the set of possible future callbacks. An app violates the protocol when, for example, it calls a particular API method in a state of the framework where such a call is invalid. What makes automated reasoning hard in this domain is largely what makes programming apps against such frameworks hard: the specification of the protocol is unclear, and the control flow is complex, asynchronous, and higher-order. In this paper, we tackle the problem of specifying and modeling event-driven application-programming protocols. In particular, we formalize a core meta-model that captures the dialogue between event-driven frameworks and application callbacks. Based on this meta-model, we define a language called *lifestate* that permits precise and formal descriptions of application-programming protocols and the callback control flow imposed by the event-driven framework. Lifestate unifies modeling what app callbacks can expect of the framework with specifying rules the app must respect when calling into the framework. In this way, we effectively combine lifecycle constraints and typestate rules. To evaluate the effectiveness of lifestate modeling, we provide a dynamic verification algorithm that takes as input a trace of execution of an app and a lifestate protocol specification to either produce a trace witnessing a protocol violation or a proof that no such trace is realizable.

**2012 ACM Subject Classification** Software and its engineering → Software verification

**Keywords and phrases** domain-specific languages, event-based programming, language implementation, new programming models or languages, object-oriented programming, semantics, testing, verification - automated

**Digital Object Identifier** 10.4230/DARTS.5.2.13

**Related Article** Shawn Meier, Sergio Mover, and Bor-Yuh Evan Chang, “Lifestate: Event-Driven Protocols and Callback Control Flow”, in 33rd European Conference on Object-Oriented Programming (ECOOP 2019), LIPIcs, Vol. 134, pp. 1:1–1:29, 2019.
https://dx.doi.org/10.4230/LIPIcs.ECOOP.2019.1

**Related Conference** 33rd European Conference on Object-Oriented Programming (ECOOP 2019), July 15–19, 2019, London, United Kingdom

## 1 Scope

The accompanying scholarly paper [1] argues for a re-examination of the process of modeling callback control flow. Through this process, we identified some essential aspects of event-driven frameworks to arrive at a language called *lifestates* that simultaneously captures callback control flow and event-driven application-programming protocols at the app-framework interface. This re-examination leads to both a methodology for empirically validating such event-driven framework models against corpora of app-framework interaction traces and a technique for verifying trace rearrangements are absent of protocol violations. Overall, we evaluate empirically the capacity of lifestates to model a real, complex event-driven framework like Android and the necessity of validating such models (cf. Section 6 of the accompanying paper [1]). This artifact includes the software and inputs that we used in the evaluation section of our paper.

## 2 Content

The artifact package includes a virtual machine image (username `verivita` and password `verivita`) with the software, trace corpora, and callback control-flow models described above, along with the measurements produced to respond to the research questions described above.

## 3 Getting the artifact

The artifact endorsed by the Artifact Evaluation Committee is available free of charge on the Dagstuhl Research Online Publication Server (DROPS). In addition, the artifact is also available at: `https://drive.google.com/open?id=15DSRQCvuxgxhYKcA7L3ah9W1QNE_t4Wr`

## 4 Tested platforms

We provide the artifact as an Ubuntu virtual machine created with Virtual Box. We suggest using a machine with at least 16GB of RAM and dual core CPU.

## 5 License

Different parts of the artifact are under different licenses. Verivita, TraceRunner, and the trace data are under the Apache 2.0 license. Benchtools is licensed under GPL V3. NuXmV is under a proprietary license (see `https://es-static.fbk.eu/tools/nuxmv/downloads/LICENSE.txt`).

## 6 MD5 sum of the artifact

verivita.ova : d0ebbd97cb03e592b4d73ba57a633279

## 7 Size of the artifact

14 GB

## A Running the Virtual Machine

We suggest using VirtualBox which can be downloaded from `https://www.virtualbox.org/wiki/Downloads`. The virtual machine image may be downloaded from `https://drive.google.com/open?id=15DSRQCvuxgxhYKcA7L3ah9W1QNE_t4Wr`. Import the machine by clicking `file` → `import appliance` and select `verivita.ovf`.

## B Extended Artifact Description

An extended description of the artifact and how to reproduce results may be found on the desktop of the virtual machine in the file `artifact_description_extended.pdf`.

### References

1 Shawn Meier, Sergio Mover, and Bor-Yuh Evan Chang. Lifestate: Event-Driven Protocols and Callback Control Flow. In *ECOOP*, 2019.