MESA:Message-based System Analysis User Manual

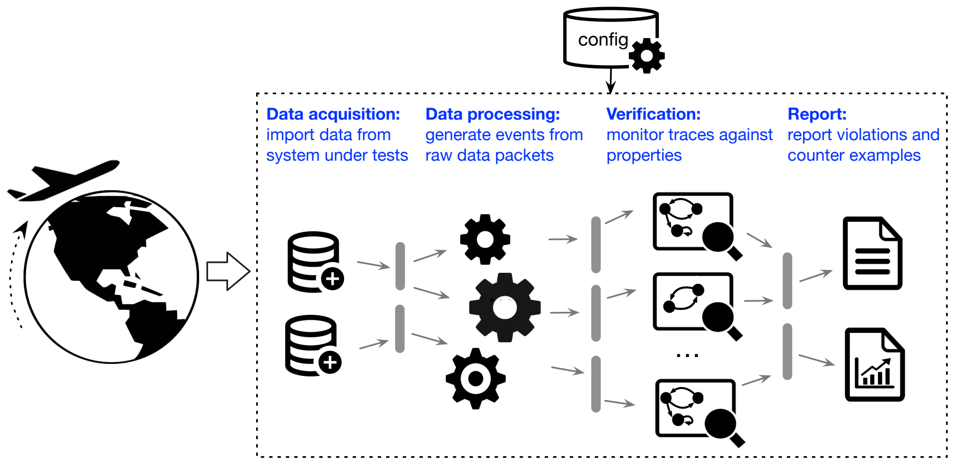
Nastaran Shafiei

SGT, Inc./NASA Ames Research Center, Moffett Field, USA

July 13, 2021

Contents

1 Framework Overview


config
Data acquisition:
import data from system under tests
Data processing:
generate events from raw data packets
Verification:
monitor traces against properties
Report:
report violations and counter examples
...

MESA [15] is a runtime verification framework that allows for building concurrent monitoring systems. It is used to check if a run of a system under observation (SUO) satisfies properties of interest. The figure above provides a high-level overview of MESA. It can be seen that the runtime verification approach used in MESA includes four different phases which are as follows.

1. The *data acquisition* phase imports raw data from the SUO into the MESA environment.

2. The *data processing* phase produces messages from a raw data packet retrieved by the previous phase.

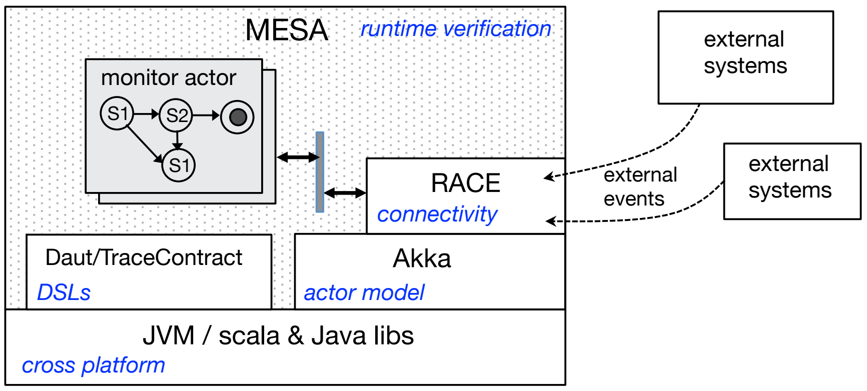
3. The *verification* phase evaluates the properties by applying them to each message produced in the previous phase. If a violation of a property occurs, the result will be reported by the next phase.

4. The *report* phase produces verification results which includes violations discovered by the previous phase and their corresponding counter examples.

MESA adopts the actor model employed by the Akka toolkit [14, 1]. Akka is an open source Scala project for building highly concurrent and distributed applications that scale. MESA applications are actor-based where each phase is implemented by a set of specialized and lightweight *actors* that run concurrently. Actors are isolated objects that communicate solely by exchanging asynchronous messages using point-to-point and publish-subscribe communication paradigms. The development of MESA is driven by our effort towards analyzing the runtime behavior of National Airspace System (NAS). However, MESA is not specific to this particular system or domain and it can be applied on a different context. For example, MESA is applied to the UxAS system (Unmanned Systems Autonomy Services) [8].

We refer to MESA as a framework since it is used to build actor-based monitoring systems. It offers building blocks to create such instances. Applying MESA requires a configuration file which is a text file in HOCON format which specifies actors in each verification phase and outlines the way they are connected. Often one needs to create application specific actors to extend MESA towards different domains. Some of the MESA key features include efficiency and scalability achieved by the Akka actor model. It can also provides the simultaneous check for different properties by allowing multiple concurrent actors in each phase of the verification.

1.1 External Systems



1.1.1 RACE

MESA leverages RACE (Runtime for Airspace Concept Evaluation) [13, 12] as an external library which is an open source Scala project [5]. RACE is a platform for generating airspace simulations, and used as an external library in MESA. RACE also employs the Akka toolkit, and extends it with some features that are used in MESA, such as synchronizing the execution of actor lifetime phases, and a mechanism to let remote actors communicate with local actors seamlessly using the same API. Moreover, as a framework to build airspace simulations, RACE already provides components to import, translate, filter, archive, replay, and visualize data from NAS which can be directly employed in MESA when checking for properties in the NAS domain.

1.1.2 Trace Analysis DSLs

MESA also incorporates the TraceContract [9, 7] and Daut [10, 11, 3] DSLs (Domain Specific Language) as external libraries. They provide support for property specification in linear temporal logic formulas and data-parameterized state machines.

2 Prerequisites

2.1 Hardware

At the moment, MESA has been tested on Ubuntu 18.04.3 LTS machine, Mac OS 10.14.6, and Mac OS 10.15.7. One of the key features of MESA is to provide distributed monitoring capability via concurrent monitor actors. Using machines with higher number of cores, one can take advantage of this capability.

2.2 Software

2.2.1 Java

MESA is written in Scala which is a VM language which relies on Java libraries. To use MESA, Java JKD (Java Development Kid) [4] version 12 or higher is needed.

2.2.2 SBT

SBT (Scala Build Tool) [6] is a build tool for Scala and Java applications. It includes native support for compiling Scala code, and provides dependency management using Ivy [2].

2.2.3 Git

Git is used as a distributed version control system used when developing MESA, and it is need to obtain MESA sources.

2.2.4 RACE

The RACE sources can be downloaded by running the following the Git command.

```
git clone https://github.com/NASARace/race.git
```

To build RACE, run the `sbt` command from the RACE root directory obtained by using `git` as follows. More details on how to build RACE can be found at http://nasarace.github.io/race/installation/build.html.

```
cd race
sbt

[info] welcome to sbt 1.5.1 (Oracle Corporation Java 14.0.1)
[info] loading settings for project global-plugins from ...
[info] loading global plugins from ...
...
[info] started sbt server
[race]>
```

After building RACE, use the SBT command `publishLocal` as follows to publish RACE to `/.ivy2/local/`, accessible by MESA.

```
cd race
sbt publishLocal
```

3 Download MESA

The MESA sources can be downloaded by running the following Git command.

```
git clone https://babelfish.arc.nasa.gov/git/mesa
```

The MESA root directory includes the following directory structure.

```
|-README.md
|-lib
|-mesa-core
|-mesa-nextgen
|-build.sbt
|-project
|-license.pdf
|-mesa-manual.pdf
```

4 Building MESA

To build MESA, first enter the directory `mesa` which is the MESA root directory obtained by using `git` command, and run the command `sbt` which takes you to the SBT interactive console and downloads all the external dependencies for running SBT. Note that at this stage, Daut and TraceContract sources are downloaded from `GitHub` and built automatically, as part of the MESA SBT built.

```
cd mesa
sbt

[info] Loading settings for project global-plugins from ...
[info] Loading global plugins from ...
[info] Loading project definition from ...
...
[info] sbt server started at local:///Users/nshafiei/...
sbt:mesa>
```

Then, execute the `compile` command from within the SBT console to compile the code.

```
sbt:mesa> compile

[info] Compiling ... Scala sources to...
...
[info] Done compiling.
[success] Total time: 8 s, completed Jul 12, 2021, 7:51:18 PM
```

To verify the build, run the command `test` from the SBT console which compiles and executes all the existing tests in MESA.

```
sbt:mesa> test

[info] Compiling 24 Scala sources to ...
[info] Compiling 3 Scala sources to ...
[info] Compiling 30 Scala sources to ...
...
[success] Total time: 8 s, completed Jul 13, 2021, 1:45:16 PM
```

To build a stand-alone script to run MESA from outside of the SBT environment, run the command `stage`. This creates an executable script in the directory `/mesa/target/universal/stage/bin` which is linked to a script `mesa` in the MESA root directory. Running MESA from outside the SBT environment is more efficient in terms of memory.

```
sbt:mesa> stage

[info] Wrote /Users/nshafiei/projects/rv/mesa/...
[info] Wrote /Users/nshafiei/.sbt/1.0/staging/...
...
[success] Total time: 5 s, completed Jul 13, 2021, 2:00:12 PM
```

5 Running MESA

Once MESA is compiled, you can run it from the SBT console using the command `run` which takes a MESA configuration file (`*.conf`) as an argument.

```
sbt:mesa> run ./config/mesa-example.conf

[info] Running gov.nasa.mesa.core.MesaMain ./config/mesa-example.
    conf
...
```

Alternatively, to run MESA outside of the SBT environment, you can use the `mesa` scripts in the MESA root directory generated by the SBT `stage` command.

```
nshafiei$./mesa ./config/mesa-example.conf
```

References

[1] Akka - scalable realtime transaction processing, http://doc.akka.io/docs/akka/current/scala.html

[2] Apache ivy, http://ant.apache.org/ivy/

[3] Daut DSL, https://github.com/havelund/daut

[4] Java Development Kit, https://www.oracle.com/java/

[5] RACE: Runtime for Airspace Concept Evaluation , https://github.com/NASARace/race

[6] Scala Build Tool, https://www.scala-sbt.org/index.html

[7] TraceContract DSL, https://github.com/havelund/tracecontract

[8] Unmanned Systems Autonomy Services, https://github.com/afrl-rq/OpenUxAS

[9] Barringer, H., Havelund, K.: Tracecontract: A Scala DSL for trace analysis. In: FM: Formal Methods - 17th International Symposium on Formal Methods. pp. 57–72 (2011)

[10] Havelund, K.: Data automata in scala. In: Theoretical Aspects of Software Engineering Conference. pp. 1–9 (2014)

[11] Havelund, K.: Monitoring with data automata. In: 6th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation. pp. 254–273 (2014)

[12] Mehlitz, P., Giannakopoulou, D., Shafiei, N.: Analyzing airspace data with race. In: DASC: Digital Avionics Systems Conference (2019)

[13] Mehlitz, P., Shafiei, N., Tkachuk, O., Davies, M.: Race: building airspace simulations faster and better with actors. In: DASC: Digital Avionics Systems Conference (2016)

[14] Roestenburg, R., Bakker, R., Williams, R.: Akka in Action. Manning Publications Co., Greenwich, CT, USA, 1st edn. (2015)

[15] Shafiei, N., Havelund, K., Mehlitz, P.: Actor-based runtime verification with mesa. In: Deshmukh, J., Ničković, D. (eds.) Runtime Verification. pp. 221–240. Springer International Publishing, Cham (2020), https://doi.org/10.1007/978-3-030-60508-7_12