Lotus@Runtime: A Tool for Runtime Monitoring and Verification of Self-adaptive Systems (Artifact)

Davi Monteiro Barbosa[1], Rómulo Gadelha de Moura Lima[2], Paulo Henrique Mendes Maia[3], and Evilásio Costa Junior[4]

1 **State University of Ceará (UECE), 1700 Dr. Silas Munguba Avenue, Fortaleza 60714-903, Brazil**
`http://orcid.org/0000-0002-7313-5825`
`davi.monteiro@aluno.uece.br`

2 **State University of Ceará (UECE), 1700 Dr. Silas Munguba Avenue, Fortaleza 60714-903, Brazil**
`http://orcid.org/0000-0001-6507-2396`
`romulo.gadelha@aluno.uece.br`

3 **State University of Ceará (UECE), 1700 Dr. Silas Munguba Avenue, Fortaleza 60714-903, Brazil**
`http://orcid.org/0000-0002-6683-6869`
`pauloh.maia@uece.br`

4 **State University of Ceará (UECE), 1700 Dr. Silas Munguba Avenue, Fortaleza 60714-903, Brazil**
`http://orcid.org/0000-0002-0281-2964`
`evilasio.junior@uece.br`

Abstract

This paper presents Lotus@Runtime, an extensible tool that uses models@runtime to monitor and verify self-adaptive systems. The tool monitors the execution traces generated by a self-adaptive system and annotates the probabilities of occurrence of each system action on their respective transition on the system model, which is created at design time in the tool as a Labelled Transition System (LTS). Then, runtime checks of a set of reachability properties are performed against the updated probabilistic model. If a property is violated, the self-adaptive system can be informed by a notification mechanism provided by Lotus@Runtime. The applicability of the proposed tool has been demonstrated by two service-based self-adaptive systems taken and adapted from the literature.

1998 ACM Subject Classification D.2.4 Software/Program Verification

Keywords and phrases Self-adaptive systems; Runtime models; Runtime verification; Tool; Framework

Digital Object Identifier 10.4230/DARTS.3.1.7

Related Article Davi Monteiro Barbosa, Rómulo Gadelha de Moura Lima, Paulo Henrique Mendes Maia, Evilásio Costa Junior, "Lotus@Runtime: A Tool for Runtime Monitoring and Verification of Self-adaptive Systems", in Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2017).
`http://dx.doi.org/10.1109/SEAMS.2017.18`

Related Conference 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2017), May 22–23, 2017, Buenos Aires, Argentina

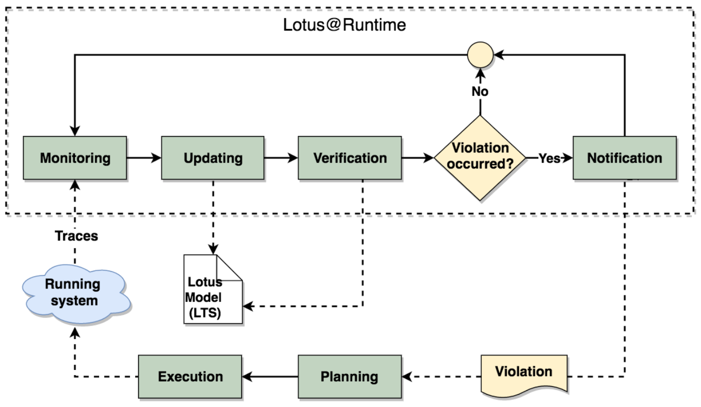

Figure 1 Lotus@Runtime Adaptation Flow.

1 Scope

Lotus@Runtime is an extensible tool that uses models@runtime to monitor and verify self-adaptive systems. The tool monitors the execution traces generated by a self-adaptive system and annotates the probabilities of occurrence of each system action on their respective transition on the system model, which is represented by a Probabilistic Labelled Transition System (PLTS). Then, runtime checks of a set of reachability properties are performed against the updated model. If a property is violated, the self-adaptive system can be informed by a notification mechanism provided by Lotus@Runtime.

In the Lotus@Runtime, an adaptation flow, illustrated in Figure 1, starts by monitoring the execution trace generated by a self-adaptive system. The traces are captured during the monitoring phase by the MonitorComponent. Then, during the model update process, the LotusModelComponent calculates the probabilities of each transition and updates the model with the new probabilities at runtime. After, the ModelCheckerComponent performs runtime checks of the user defined properties against the updated probabilistic software behavior model. If a property is violated, then NotifierComponent triggers a notification about that violation. Regardless of whether or not a violation occurs, the monitoring process should keep running continuously. To receive the violations that may occur, developers must implement the interface ViolationHandler at the planning phase. Otherwise, the notifications will be lost.

1.1 Tele Assistance System (TAS)

The Tele Assistance System is a service-based system that provides health support to chronic condition sufferers within the comfort of their homes [1]. In the original concept, the TAS uses a wearable device to take periodical measurements of the patient's vital parameters and analyze them through medical services. After the analysis, the result may trigger the invocation of an alarm service to call an ambulance, or the invocation of a pharmacy service to deliver a medication to the patient's house.

The system is composed by the following three abstract services:

- Alarm Service, which triggers the alarm and calls the first-aid squad (ambulance).
- Medical Analysis Service, which analyses the patient's vital parameters.
- Drug Service, which requests a medicine replacement or changes its dosage.

To configure the Lotus@Runtime in the TAS, we have adopted the strategy to monitor new traces of every change in the log file, and created the two following quality of service requirements:

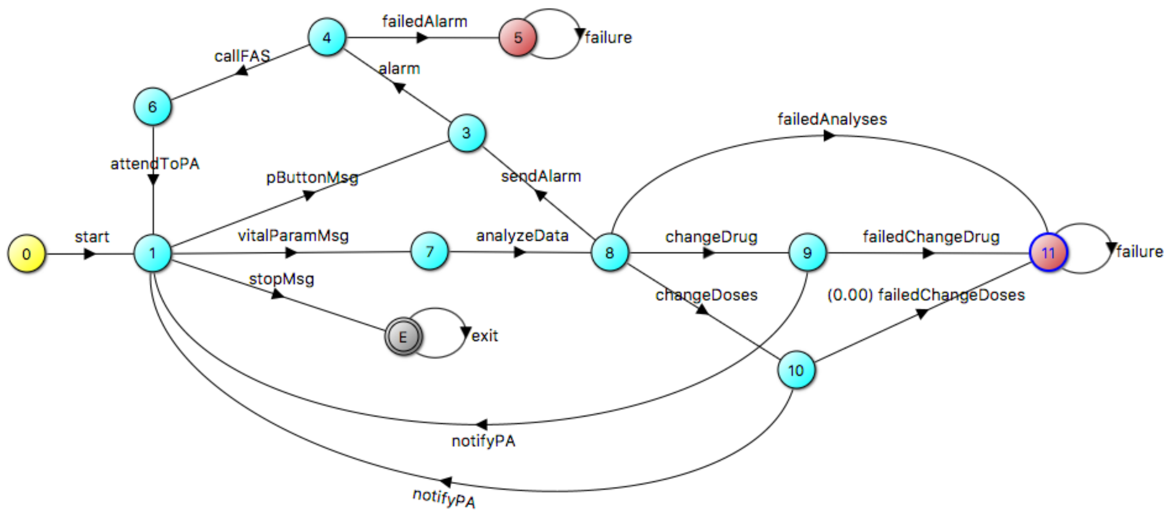

Figure 2 Tele Assistance System Model.

- (P1) the probability of a failure after the alarm service should be less than 20%, represented by reaching state 5 from state 0.
- (P2) the probability of a failure after the pharmacy service should be less than 10%, represented by reaching state 11 from state 0.

The LTS model, as illustrated in Figure 2, was created using the LoTuS tool and represents the system behavior with their services. The model is based on the DTMC model presented by Calinescu et al. [1].

1.2 Travel Planner Application (TPA)

The Travel Planner Application is a service-based system that provides services to search for flights, tourist attractions, accommodation arrangements, and rental cars or bicycles. The application is composed by the following five services:

- Attraction information service, which searches for tourist attractions.
- Flight reservation service, which searches for available flights.
- Hotel reservation service, which seeks hotels and makes reservations.
- Bicycle rental service, which allows renting bikes.
- Car rental service, which allows renting cars.

The TPA model is illustrated in the Figure 3, which was based on the DTMC model, available on COVE (COntinual VErification) framework site[1]. To configure Lotus@Runtime in the TPA, we have adopted the strategy to verify if there is a new trace at every 2 seconds, and created the three following quality of service requirements:

- (P1) the probability of a failure after the flight reservation service should be less than 15%, represented by reaching state 5 from state 0.
- (P2) the probability of a failure after the hotel reservation service should be less than 10%, represented by reaching state 7 from state 0.

[1] https://www-users.cs.york.ac.uk/raduc/COVE/

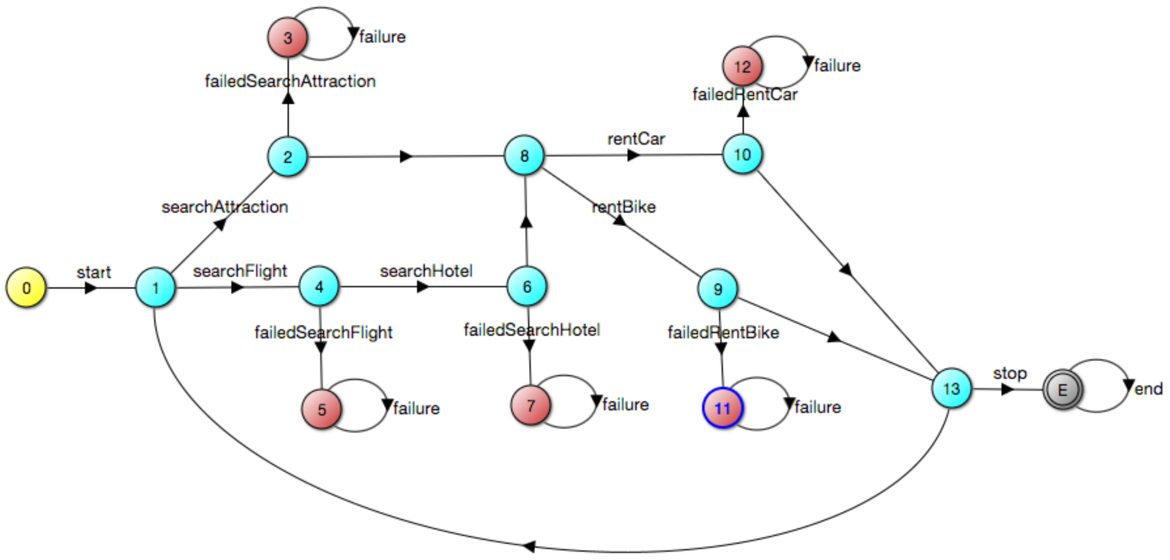

Figure 3 Travel Planner Application Model.

- (P3) the probability of a failure after the attraction information service should be less than 20%, represented by reaching state 3 from state 0.

2 Content

The artifact package includes:

- Lotus@Runtime and the exemplars workspace
- Lotus@Runtime source code
- TAS + Lotus@Runtime executable
- TPA + Lotus@Runtime executable

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:

- Lotus@Runtime source code: `https://github.com/davimonteiro/lotus-runtime`
- TAS GUI: `https://github.com/davimonteiro/travelapp_gui`
- TPA GUI: `https://github.com/davimonteiro/TAS_gui`

4 Tested platforms

4.1 Hardware Overview

MacBook Pro (Retina, 13-inch, Late 2013)
Processor: 2,4 GHz Intel Core i5
Memory: 8GB 1600 MHz DDR3
Graphics: Intel Iris 1536 MB

4.2 Software Overview

Operation System: macOS Sierra - version 10.12.3
Java(TM) SE Runtime Environment (build 1.8.0_45-b14)
Java HotSpot(TM) 64-Bit Server VM (build 25.45-b02, mixed mode)
Maven 3.2.5
Eclipse Java EE IDE for Web Developers - version: Neon.2 Release (4.6.2)

5 License

6 MD5 sum of the artifact

d3c770ee9a970bb96b152ba72dde6120

7 Size of the artifact

212.771.306 bytes (212,8 MB)

References

1 R. Calinescu, K. Johnson, and Y. Rafiq. Developing self-verifying service-based systems. In *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, pages 734–737, Nov 2013. `doi:10.1109/ASE.2013.6693145`.