Offline Runtime Verification with Real-Time Properties: A Case Study

Christian Colombo
Dept. of Computer Science
University of Malta
christian.colombo@um.edu.mt

Gordon J. Pace
Dept. of Computer Science
University of Malta
gordon.pace@um.edu.mt

Patrick Abela
Ixaris Ltd
Malta
patrick.abela@ixaris.com

Abstract

Online monitoring of software systems has lately been gaining popularity. Yet, such monitoring interferes with the system under observation, possibly causing unexpected repercussions such as system slowdown. In various cases, such as the financial sector, it is still worth detecting a problem late. The awareness of the problem might be useful to avert similar problems from reoccurring in the future. The LARVA *monitoring framework has been adapted to support offline monitoring of systems and has been applied to an industrial case study — a financial transaction system. Various interesting issues emerge: initialisation of monitors, and the implementation of timestamp-driven clocks.*

Index Terms

runtime verification, real-time properties, duration calculus

1. Introduction

To date, the most widely used technique for software quality assurance is testing. Yet, with testing in place, we have grown accustomed to expect a newly released software to contain bugs. This situation might be acceptable for a word processor but not ideal for a system which handles financial transactions. Model checking [4] techniques, which guarantee that a system adheres to its properties no matter what the inputs are, is the golden grail. Unfortunately, it does not scale up well for large systems. A possible solution to this dilemma is runtime monitoring [5] where a system is continuously being monitored to detect any problems which might arise and possibly take some remedy action. The idea of monitoring is not new and has been used for decades to monitor critical systems such as power plants. In our approach, instead of installing sensors to detect malfunction of machinery, we insert monitors in software detecting system calls and variable changes. Instead of checking against fluid leaks, we check that no data is given to unauthorised users. Instead of shutting down part of the plant in case of a fault, we block a malicious user from the system. There are two modes of monitoring: online and offline. Online monitoring is when monitoring is done in synch with the observed system, i.e. the system waits for the monitor to complete its verification before progressing. In the case of offline monitoring, a log of events is monitored, possibly much later than the actual time the events occurred. This means that the monitor cannot take recovery actions at the time when the problem occurs. The advantage is that monitoring does not interfere with the system except for the logging of events, which is typically performed in any case.

In this paper, we present the adaptation of the runtime monitoring system LARVA applied to an industrial case study from Ixaris Ltd. The contributions of this work are two-fold: (i) an offline monitoring architecture supporting a highly expressive logic, initialisation of monitors, and timestamp-driven clocks; (ii) the application of the architecture on an industrial case study.

This paper is structured as follows: next we give background information about LARVA and the Ixaris system (on which the case study was carried out). In Section 4 we give an overview of the architecture — going into detail in certain interesting aspects. An account of the properties of the case study is given in the Section 5. Subsequently, we relate the work to the literature in Section 6 and conclude, giving future directions.

This is intended as an internal report.

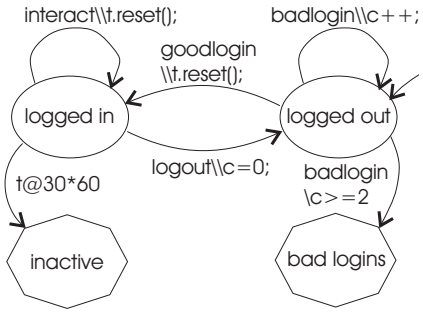
2. Background

2.1. LARVA

A runtime verification architecture normally involves the following five components: (i) a system to be monitored; (ii) a set of specifications written in some formal notation; (iii) a stream of events extracted from the system in (i); (iv) a monitoring system which receives the events and verifies them according to the specification in (ii); and (v) a feedback loop through which the monitor may influence the system in case of property violation. The LARVA architecture is no exception and has the above five components. However, for the purposes of this paper, we will ignore the feedback loop capability of LARVA.

A user who wants to monitor a system using LARVA must supply the system itself — a Java program — and a set of specifications in the form of a LARVA script — a textual representation of DATEs [6], similar to timed-automata enriched with stopwatches. Using the LARVA compiler the specification is transformed into the equivalent monitoring code, together with a number of aspects which extract the events from the system. Aspects are generated in AspectJ, one of the aspect-oriented implementations for Java, enabling automatic code injection without directly altering the actual code of the system. If events are available in a database, (as in the case of our case study), a simple Java program is used to extract events from the database and replay them for LARVA to detect and check.

As an example, consider a system where one needs to monitor bad logins and the activity of a logged in user. By having access to *badlogin*, *goodlogin* and *interact* events (each of which corresponds to an entry in the log database), one can keep a successive bad-login counter and a clock to measure the time a user is inactive. Fig. 1 shows the specification of a property stating that there are no more than two successive bad logins and 30 minutes of inactivity when logged in, expressed as a DATE automaton [6]. Transitions have three (backslash separated) labels: (i) the event triggering it; (ii) the condition which is checked before taking it; and (iii) the action performed when it is taken. A total ordering on the transitions is used to ensure determinism.




```
GLOBAL {
 VARIABLES {
   int c = 0;
   Clock t;
 }
 EVENTS {
   badlogin() = { User.badlogin() }
   clk()      = { t@30*60 }
   ...
 }
 PROPERTY users {
   STATES {
     BAD      { badlogins inactive }
     NORMAL   { loggedin }
     STARTING { loggedout }
   }
   TRANSITIONS {
     loggedout -> badlogins [badlogin\c>=2\]
     loggedout -> loggedin  [goodlogin\\t.reset();]
     ...
     loggedout -> loggedout [badlogin\\c++;]
     loggedin -> inactive   [clk\\]
   }
 }
}
```

Figure 1. The automaton and LARVA code of the bad logins scenario.

3. Technology

Ixaris Systems Ltd is a privately owned company that operates EntroPay, an online prepaid payment service. Entropay users deposit funds through funding instruments types (such as their own personal credit card or through a bank transfer mechanism) and spend such funds through spending instruments (such as a virtual VISA card or a Plastic Mastercard). The service is used worldwide and thousands of transactions are processed on a daily basis.

EntroPay is built on SmartPay, a technology framework developed and maintained by Ixaris Systems (Malta) Ltd. This framework addresses various data integrity considerations which need to be taken in consideration when dealing with financial transactions. Through the work presented in this paper, Ixaris aims to extend the Smartpay platform with a run-time verification mechanism which, apart from monitoring against errors due the system code, monitors also the integrity of the system across the system's runtime execution environment which would include elements ranging from the hardware and middleware configuration to day-to-day manual processes run by operations staff.

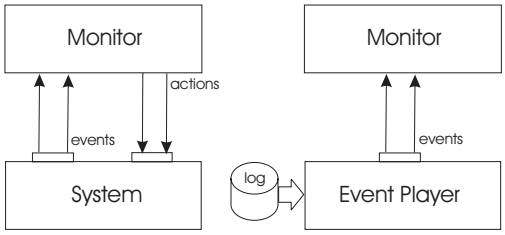


Figure 2. An online monitoring system (left) and an offline monitoring system (right).

4. The Monitoring Architecture

The offline monitoring architecture is very similar to the online counterpart. The main difference is that rather than interfacing with the system, an additional component is required to interface with the system log (a database). In the case of online system, the monitor receives events directly from the system in synch, i.e. the system waits for the monitor to process events before progressing further. In the case of an offline system, an event player extracts events from the system log and feeds them to the monitor, waiting for the monitor to process each event. Therefore there is no difference in the monitor, except that in the offline mode it cannot take actions to remedy problems. Fig. 2 shows the difference between both architectures.

4.1. Generating Events

For our intents and purposes, an event can be considered as any interesting action which occurred within a system. In a log database, an event is usually a record in a table. However, in some cases, to locate all the event details (parameters), several records need to be consulted. Inversely, a record might feature in several events. Considering the need for such flexibility, an appropriate way of defining an event is by using a *select* statement. Each *select* statement returns a number of records — considered to be a list of events of a particular type — where each record has a number of fields considered to be the event parameters. It is up to the designer of the properties to define an event in terms of an SQL *select* statement. Once a *select* statement has been drawn up for each event type, they are executed and all the events are sorted in chronological order. Due to sorting, it is necessary that every event has a timestamp field.

5. Properties

There are various properties which are currently being monitored. These are by far not exhaustive but the focus was on what is most sensitive and worth monitoring. In general, the properties fall under two categories: (i) checking that something occurs within the expected time frame; and (ii) checking that something does not occur when not expected to (commonly known as bounded liveness and safety properties). To decide whether something is expected or not, the system requires some kind of context to take the decision. Although a lot of properties overlap in the type of context they use, we can roughly classify the properties under four kinds of context (listed in order of the number of properties):

- **Life cycle** A lot of properties depend on which phase of the life-cycle an entity is. For example, the current user state is crucial in determining the subsequent user state.
- **Real-time** A good number of properties are real-time properties. In most cases this is due to some kind of expiration which triggers some activity in the system.
- **Rights** User rights are a very important aspect of the system's security. A number of transactions require the user to have particular rights before the transaction is permitted.
- **Amounts** There are various limits (for security reasons) on the frequency of certain transactions and the total amount of money which these transactions constitute. For example, a user who is unexpectedly loading a lot of money and spending it, might imply that the user account has been hacked.

6. Related Work

In principle, any algorithm used for online monitoring can be used for offline monitoring as long as all the information available at runtime is also available offline. The inverse, however, is not always true because some offline algorithms [10], [11] assume that the complete trace is available at the time of checking.

There are numerous algorithms and tools [7], [8], [1], [3], [2], [10], [9] which support offline monitoring — sometimes also known as trace checking. However, we are particularly interested in log analysers, i.e. analysers which obtain the trace directly from the log file. This technique is mostly used for the evaluation of testing results. There are three works which are closest to our approach [1], [3], [2], two of which come from the area of testing [3], [2]. The Test Behaviour Language (TBL) [3] allows a user to write specifications in terms of patterns which are automatically translated into executable scripts. TBL supports parametrisation

of events and a timeout construct. The use of the timeout requires the validating script to run at the same time of the system. Thus, although extracting information from the logs, the monitoring is not purely offline. The Log File Analysis Language (LFAL) [2], like TBL, is a specification language which can be translated to verify test results. LFAL does not support real-time and is purely offline. Apart from parametrisation of events, LFAL also supports parametrisation of states and uses a state-machine-like notation. Both TBL and LFAL are applied directly to textual logs. This approach is highly susceptible to changes in the format of the log. LOGSCOPE[1], on the other hand, base its specification on an abstraction of the log. Although LOGSCOPE uses uses a user friendly language, it is quite restrictive for the general case. Properties for LOGSCOPE can also be defined in terms of $\forall$-automata which are much more expressive. The latter is by far the work closest to our approach. The main difference is that LOGSCOPE does not offer parametrised monitors, and explicit support for real-time properties.

7. Conclusions and Future Work

In this paper we presented an offline monitoring architecture where the log of the system is verified asynchronously with respect to a specification written in LARVA. Various interesting issues emerge such as the implementation of timestamp-driven clocks (TDCs), and the initialisation and maintenance of the monitors. The architecture has been applied to an industrial case study and the preliminary results are encouraging.
The work presented here is intended to evolve into a novel architecture where the online and the offline approach can be combined using the notion of compensations where actions of a system can be 'undone' to somewhat restore a previous state. Thus, even if an error is detected late, (due to the non-synchronous nature of offline monitoring) compensations can be used to correct the state of the system.

References

[1] M. S. H. B. A. Groce, K. Havelund. Let's look at the logs: Low-impact runtime verification. Submitted to the The Computer Journal special issue on the COMPASS'09 ETAPS workshop.

[2] J. H. Andrews and Y. Zhang. General test result checking with log file analysis. *IEEE Trans. Softw. Eng.*, 29(7):634–648, 2003.

[3] F. Chang and J. Ren. Validating system properties exhibited in execution traces. In *ASE '07: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 517–520. ACM, 2007.

[4] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model checking*. MIT Press, 2000.

[5] S. Colin and L. Mariani. *Model-Based Testing of Reactive Systems*, volume 3472. Springer Berlin / Heidelberg, 2005.

[6] C. Colombo, G. J. Pace, and G. Schneider. Dynamic event-based runtime monitoring of real-time and contextual properties. In *FMICS'08*. To appear in LNCS, 2008.

[7] B. D'Angelo, S. Sankaranarayanan, C. Sánchez, W. Robinson, B. Finkbeiner, H. B. Sipma, S. Mehrotra, and Z. Manna. Lola: Runtime monitoring of synchronous systems. In *12^{th} International Symposium on Temporal Representation and Reasoning (TIME'05)*, pages 166–174. IEEE Computer Society Press, June 2005.

[8] S. A. Ezust and G. v. Bochmann. An automatic trace analysis tool generator for estelle specifications. In *SIGCOMM '95: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, pages 175–184, 1995.

[9] K. Havelund and G. Rosu. Synthesizing monitors for safety properties. In *TACAS '02: Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 342–356, London, UK, 2002. Springer-Verlag.

[10] G. Roşu and K. Havelund. Rewriting-based techniques for runtime verification. *Automated Software Engg.*, 12(2):151–197, 2005.

[11] G. Rosu and K. Havelund. Synthesizing dynamic programming algorithms fromlinear temporal logic formulae. Technical report, 2001.

Acknowledgements

The research work disclosed in this publication is funded by the Malta National Research and Innovation (R&I) Programme 2008 project number 052.