Runtime Verification for Stream Processing Applications

Christian Colombo[1], Gordon J. Pace[1],
Luke Camilleri[2], Claire Dimech[1], Reuben Farrugia[3], Jean Paul Grech[1],
Alessio Magro[4], Andrew C. Sammut[3], and Kristian Zarb Adami[4]

[1] Department of Computer Science, University of Malta
[2] Ixaris System Ltd, Malta
[3] Department of Communications & Computer Engineering, University of Malta
[4] Institute of Space Sciences and Astronomy, University of Malta

Abstract. Runtime verification (RV) has long been applied beyond its strict delineation of *verification*, through the notion of monitor-oriented programming. In this paper we present a portfolio of real-life case studies where RV is used to program stream-processing systems directly— where all the logic of the implemented system is defined in terms of monitors. The systems include the processing of Facebook events for business intelligence, analysing users' activity log for detecting UI usability issues, video frame analysis for human movement detection, and telescope signals processing for pulsar identification.

1 Introduction

Runtime verification (RV) [1] is a lightweight formal methods technique which allows users to specify formal properties and through instrumentation and automatically synthesised monitors, check that a system's behaviour adheres to the properties. However, RV has long been applied beyond this strict definition. In particular, the notion of verification is not the sole application of monitoring, especially with the rise of the notion of Monitor-Oriented Programming (MOP) [9] which takes the approach further — advocating how a system's functionality can be extended through the use of monitors. The architecture of a system built in such a manner is shown in Fig. 1(a): The system,

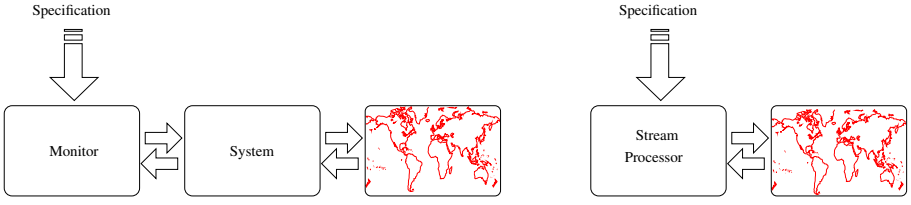

Fig. 1. (a) MOP architecture; (b) Stream-processing architecture

interacting with the real world, generates events which are captured by the specification-synthesised monitor, which in turn reacts to particular traces of events according to the specification. However, the use of runtime monitoring techniques and tools has been pushed even further by programming stream processing systems directly as monitors, thus having the monitors generated from the specifications interact directly with events generated from a real-world event sensor or event generator (which might also be a computer system). The architecture of such a system is shown in Fig. 1(b), with the monitor processing the stream of events directly. The key feature of this approach is that if we see a stream as a total function from the discrete time domain $\mathbb{N}$ to the type of values carried on the stream, a stream processing system is one which takes an input stream, and produces an output stream[5]. In general, while the output stream does not need to be produced in sync with the input stream, we expect that the values are produced in sequence and not out of order[6].

While RV can be seen as providing a convenient layer of abstraction for stream processing applications, stream processing techniques can provide support in two challenging aspects of RV: *reactivity* and *efficient computability*.

Reactivity. Stream processing applications, such as sound or video processing, frequently tolerate little delay between the input stream and the corresponding points in the output stream. This is also a desirable property in several RV applications. The problem, however, is that not all stream processors can have the output corresponding to a particular point in time to be produced at that same time instant. Consider a processor which encodes: $o(t) = i(t + 2)$ — the output at time t is the input at time $t + 2$. Clearly, without some form of temporal look-ahead, the output can only be produced two time units late. This is an issue when future time logics are used to specify properties or behaviour of stream processors. For example, the interpretation of the LTL formula XXa on a stream starting at time t, can only be known when the value of a at time $t + 2$ is known. Similarly, when matching a pattern such as a fraudulent set of transactions, the pattern may only be detectable after a significant part of fraudulent action has been observed. This is the case with the intelligent video surveillance as well as the radio telescope signal processing applications presented in the paper (Sec. 4 and 5). Conversely in the Facebook event processing application (Sec. 2), we consciously choose a specification language which enables reactivity, since we would like to have notifications triggered immediately.

Even if the specification language or logic allows for full reactivity (being able to calculate the value of $o(t)$ knowing the values of $i(0)$ till $i(t)$), one may still adopt an implementation which is not reactive. For instance, if we are given the specification $o(t) = E(i(t))$, where E is a calculation increasingly expensive as the input parameter grows, and also that large values of the input are statistically rare, we might adopt a solution which, upon receiving a large input, will continue reading inputs while computing the output in the background. This would result in a non-reactive implementation

[5] A tuple of streams can be converted into a stream of tuples, which allows this view to cater for multiple inputs and outputs.

[6] Out of order generation can still be catered for, by caching the calculated outputs but outputting them only once their turn comes. Needless to say, this might induce additional space requirements, though.

even if a reactive one was possible — with the advantage of allowing a more frequent input sampling rate.

Finally, in the case of outputs whose calculation is an approximation which can be refined as more inputs are received, the loss of reactivity corresponds directly to the level of accuracy one desires.

In view of these issues, non-reactive stream processing can range from ones which (a) compute their output as soon as possible (*'best effort'* systems), e.g., video surveillance (Sec. 4) benefits from reporting matched patterns as soon as it is possible; and ones which (b) can take even longer than strictly required by the specification language (*'late'* systems) e.g., telescope signal processing and user profiling (Sec. 3) do not require the verdict with particular urgency.

Efficient computability. In the context of stream processing, we say an application is *efficiently computable* if computing the output stream never requires more than a fixed length of history to be recorded. Contrast the stream processor $o(t) = i(t-1) + i(t)$ which requires a history buffer of size 1, with the stream processor $o(t) = i(t \operatorname{div} 2)$ which requires increasing memory as time progresses. This is crucial given the amount of data one would expect to process in the applications presented below, particularly if reactivity is required without slowing down input reading rates. In the case of the telescope signal processing system (Sec. 5), we used statistical methods which do not need to store history. For the Facebook event processing application (Sec. 2), the length of the history depends on the user-defined properties being monitored, but with the guarantee that for any given property, one can statically decide the (maximum) buffer size required. Finally, in the case of the video surveillance system (Sec. 4), the history bound depends on the maximum size of a continuous video sequence. In our implementation, the video sequence resets whenever no persons are observed for a number of frames, or if the number of frames exceed a user-specified bound. While ideal, efficient computability might not be necessary for a number of RV applications: e.g., the user web interface profiling system (Sec. 3). This is particularly so, given that it is effectively creating a statistics database based on the observed input.

2 Facebook Events Processing

With ever increasing information available in social networks, the number of businesses attempting to exploit it is on the rise, particularly by keeping track of their customers' posts and likes on social media sites like Facebook. Whilst APIs can be used to automate the tracking process, writing scripts to extract information and processing it requires considerable technical skill and is thus not an option for non-technical business analysts. On the other hand, off-the-shelf business intelligence solutions do not provide the desired flexibility for the specific needs of particular businesses.

One way of allowing a high degree of flexibility while providing an off-the-shelf solution would be to present a simple interface based on a controlled natural language (CNL) [8] which would allow a business intelligence analyst the flexibility to express the desired events for notification. These would in turn be automatically compiled into Facebook monitors without further human intervention.

Based on interviews with two business analysts, such a CNL should allow the user to specify patterns such as: (i) *Create an alert when the service page has a post and the post contains the keywords fridge, heater, or freezer.* (ii) *Create an alert when my page has a post and the post is negative and the post has 10 likes.*

Once a prototype CNL was designed, it could have potentially been compiled into any executable programming language. With the aim of keeping the translation as simple as possible, we translated the CNL into an intermediary specification from the RV domain. Translating our CNL into the formal specification accepted by the RV tool Larva [5] and using a simple adapter to present relevant Facebook events as method calls in the control flow of a program, we were able to detect Facebook behaviour through RV software. Results [4] suggest that users indeed found the CNL manageable although UI support could facilitate writing CNL sentences further.

3 Profiling User Web Interfaces

User interface designers try their best to improve the user experience to facilitate user productivity. However, the ways in which users end up using the product (e.g., the sequence in which a number of features are used) might be difficult to predict in practice. Furthermore, new features are regularly deployed with the possibility of unforeseen effects such as performance degradation. Adding profiling logic within the system code to gather such statistics would typically lead to cluttered code. In such a scenario, RV was convenient due to its separation of concerns: having the profiling logic handled by the monitor without affecting the live system.

This approach has been successfully used in the context of Ixaris System Ltd[7], a transaction processing software company, where the effectiveness with which the users were able to use the interface needed to be evaluated. A database was available with several months of logs of user activity: which users are logged in, which activities are being carried out, which currency is being used, whether the user is a first time visitor (and if so whether through a referral or a particular campaign), etc. Subsequently, we defined the possible paths of user activities in terms of a finite state machine (FSM) and by running the FSM over the logs, statistics and information could be gathered that otherwise would have had to be implemented into the production code.

The statistics gathered through RV help us identify paths in the system that need performance improvements before clients do and RV reports serve as proof that service level agreements are met. In the future, we aim to investigate ways of processing and presenting statistics in real-time without compromising performance.

4 Intelligent Video Surveillance

Video analytics has become an important feature in security systems particularly in public areas or buildings with controlled entry or exit, or where disturbances can be caused. For example motion tracking of a crowd in a football stadium may detect the start of a commotion, enabling the security personnel to act fast.

[7] `http://www.ixaris.com`

For these purposes, image processing and computer vision techniques [11, 2] have been developed with success, typically starting by identifying human body parts such as the head, the hands, the legs and so on, then tracking these parts, and subsequently attempting to identify human activity based on the movement of the individual parts. However, these techniques do not perform well in low frame rate which is the case in our case study — a prominent public building in Malta (which cannot be named). In this context, we are employing RV to specify high-level rules complementing the information provided by low-level features to suppress false positives: In the area of image processing, it is well known that it is virtually impossible to correctly identify body parts at 100% accuracy — particularly, if images are taken from poor-quality videos. RV has thus been used to specify high level rules which consider multiple subsequent frames at a time: filtering out any detections which are not found in more than one frame or propagating detections within a frame which would otherwise have been missed. This has been achieved by specifying a number of rules such as *"humans can only start or stop appearing near one of the doors"* or a *"human can only move a limited distance from one frame to the next"* and applying them on frames through an RV-synthesised monitor.

The work is still ongoing but the results achieved so far are promising: when applying the monitor to remove false positives, we obtained a 100% recall and 91% precision. In the future we hope to specify a domain-specific language to enable non-technical end users to express custom surveillance rules.

5 Radio Telescope Signal Processing

Whilst for many years the visible light was the only source of information used to discover the universe through optic telescopes, the discovery of the electromagnetic spectrum has provided a wide range of waves which can shed more light about our universe. Amongst these are the radio waves, which through the use of radio telescopes have enabled us to learn more about the universe.

The LOw Frequency ARay (LOFAR) is a radio telescope built and operated by the Netherlands Institute for Radio Astronomy (ASTRON) able to deliver around 3Gb/s of data. To detect pulsars[8] within this data on-the-fly, we have employed RV to process radio telescope signals. Due to the pulsars' periodic nature, we could detect the beam of light by keeping track of the standard deviation of the signal. The monitoring tool used was LarvaStat [3], since it provides direct support for gathering statistics. In particular, we made extensive use of LarvaStat's notion of *point statistics* to provide a context, maintain the running standard deviation while simultaneously evaluating the next value.

To evaluate the precision of our approach, we compared our results to those of a standard Fast Fourier Transform (FFT) [10] technique. We observed that our approach is not as precise: 1.01% error as opposed to 0.02%. However, the advantage of using the runtime monitoring technique is that while the time complexity of the FFT is $O(n \cdot log_2(n))$, ours is $O(n)$. Furthermore, the FFT processes data in chunks, requiring a suitably-sized buffer and a corresponding delay for detection. This is not the case with our approach which is able to process the data on-the-fly. In terms of performance, the

[8] Pulsars are rapidly spinning neutron stars which emit regular electromagnetic radiation beams.

monitoring approach did one order of magnitude worse than the FFT but this may be mainly due to the fact that the former is implemented in Java while the latter is in C.

6 Conclusion

This is not the first time that the connection between stream processing and RV has been shown with tools such as LOLA [7] and a Larva flavour [6] which accepts Lustre as a property specification language. The contribution of this paper is to highlight different case studies in which RV has proved useful in alleviating the challenges of the domain through the abstraction it provides. Returning to the stream processing categories introduced in Sec. 1, the presented applications are categorised below:

	Reactivity	**Non-reactive**	
		best effort	*late*
Efficient	Facebook	Surveillance	Telescope
Non-efficient			Profiling

We note that with the exception of profiling where the monitor is used to essentially populate a database, efficiency is a common property of monitoring applications dealing with large volumes of data. Another observation is that if monitoring is to be reactive, then it would be undesirable to have non-efficiency. Therefore, one would not typically expect to have applications which fall in the bottom-left quadrant.

We hope that this study serves as an inspiration to the wide ranging usefulness of RV techniques and, consequently, further take-up in industrial settings.

References

1. Runtime Verification conference, yearly LNCS proceedings since 2010
2. Benfold, B., Reid, I.: Stable multi-target tracking in real-time surveillance video. In: Computer Vision and Pattern Recognition (CVPR). pp. 3457–3464. IEEE (2011)
3. Colombo, C., Gauci, A., Pace, G.J.: Larvastat: Monitoring of statistical properties. In: RV. LNCS, vol. 6418, pp. 480–484. Springer (2010)
4. Colombo, C., Grech, J.P., Pace, G.: A controlled natural language for business intelligence monitoring. In: NLDB (2015), to appear
5. Colombo, C., Pace, G.J., Schneider, G.: Larva — safer monitoring of real-time java programs (tool paper). In: Seventh IEEE International Conference on Software Engineering and Formal Methods (SEFM). pp. 33–37. IEEE (2009)
6. Colombo, C., Pace, G.J., Schneider, G.: Resource-bounded runtime verification of java programs with real-time properties. Tech. Rep. CS2009-01, Department of Computer Science, University of Malta (2009), available from `http://www.cs.um.edu.mt/~reports`
7. D'Angelo, B., Sankaranarayanan, S., Sánchez, C., Robinson, W., Finkbeiner, B., Sipma, H.B., Mehrotra, S., Manna, Z.: LOLA: runtime monitoring of synchronous systems. In: TIME. pp. 166–174. IEEE (2005)
8. Kuhn, T.: A survey and classification of controlled natural languages. Computational Linguistics 40(1), 121–170 (2014)

9. Meredith, P.O., Jin, D., Griffith, D., Chen, F., Roşu, G.: An overview of the MOP runtime verification framework. STTT 14, 249–289 (2012)
10. Rao, K.R., Kim, D.N., Hwang, J.J.: Fast Fourier Transform - Algorithms and Applications. Springer, 1st edn. (2010)
11. Rodriguez, M., Laptev, I., Sivic, J., Audibert, J.Y.: Density-aware person detection and tracking in crowds. In: Computer Vision (ICCV). pp. 2423–2430. IEEE (2011)