COST Action IC 1402 ArVI: Runtime Verification Beyond Monitoring

Activity Report

Working Group 4 - Application areas

Abstract. This report presents a number of application areas which have been considered through the activities of the fourth working group of the COST Action ArVI, Runtime Verification beyond Monitoring: Financial transactions, medical devices, electronic contracts, security and privacy, and electrical energy storage.

1 Introduction

Whilst runtime verification (RV) [15,4] is typically considered to be a means of checking a system behaviour to detect divergences from some formally specified property, the application of RV techniques throughout their twenty years of history or so[1], have been far wider reaching. In its most basic form, RV simply attempts to answer the language membership query for a word which is typically available progressively. Consequently, this gives rise to a number of related concerns which usually include performance, means of specifying the language, and ways of extracting the word from the concerned system.

Over and above the vanilla RV described so far, RV frequently supports corrective actions to be specified upon violation detection, giving rise to the areas of runtime enforcement, and more generally, monitor-oriented programming. The former is concerned with not allowing a violation to happen in the first place, while the latter uses language membership queries to trigger pieces of code to construct the functionality of a system.

While RV mainly focuses on the single run under observation, the theory which has been growing around RV has also been used to provide insights into the potential behaviour of other (unseen) runs. This information is highly useful for example to detect potential race conditions even though they may never be observed at runtime.

Finally, the techniques surrounding RV - particularly those related to non-intrusively extracting events from a system - also make it useful as a tool for extracting and processing software runtime information which would otherwise be difficult to observe without cluttering the system logic. Such techniques are useful for example during testing to provide information to oracles enabling them to give their verdict.

In the rest of the document a number of domain areas, which have been explored the ARVI COST action through organised workshops and sessions, are reviewed from the runtime verification point of view:

Section 2: Financial transactions — led by Christian Colombo

[1] See http://www.runtime-verification.org/

Section 3: Medical devices — led by Martin Leucker and Ezio Bartocci
Section 4: Electronic contracts — led by Gerardo Schneider and Gordon Pace
Section 5: Security and privacy — led by Gerardo Schneider
Section 6: Electrical energy storage — led by Martin Sachenbacher

2 Financial Transactions

Following the workshop organised in April 2015 in Malta as part of the ARVI COST action[2], this section attempts to synthesise a number of challenges and recommends a way forward for RV in the context of the financial transaction industry.

2.1 Challenges

Property specification language A frequent bone of contention between teams working on RV is the choice of specification language which users will be provided with to express correctness properties. One of the mostly debated aspects is whether the language should be logic-based or automata-based. Logic-based notations provide succinct descriptions and can benefit from simplification rules and other means of manipulation. On the other hand, automata-based notations are typically less structured and thus harder to perform certain operations on. From experience [11], the team at the University of Malta (UoM) has found that automata are significantly less intimidating to technical people than logics.

Technical issues with existing tools A number of technical challenges remain an issue for industry to adopt RV, mainly stemming from the fact that most tools are implemented by students and not built for industry. This means that industry finds it hard to trust such tools, particularly because code generated by them would be running alongside their software. Apart from this generic problem of trust, there are also other concerns such as the following:

Overheads The issue of overheads may not be particularly problematic in cases where there are no or little time constraints. However, in application such as financial transactions, where deadlines are very tight, adding overheads might mean that service level agreements are not adhered to.

Integration A major concern for industry is the modification of the process: If the way software is produced and deployed is disrupted due to the introduction of RV, this might be a deal breaker for industry. The problem is exacerbated since the integration of RV into the software development life cycle has not been studied much with only far in between publications in the area (e.g., [14,1]). From past experience with the financial transaction industry [11], it was found preferable to introduce RV to the quality assurance team (QA) rather than the developers who already have to write tests for their code. For the QA, RV provides a high level view of the system, allowing them to assess the quality of service customers are getting. However, these observations do not address how RV should be incorporated within the life cycle. Not only is it currently unclear how best to incorporate RV in the software development process, but RV tools have not yet been integrated in frequently used build tools used by industry.

[2] See https://www.cost-arvi.eu/?page_id=166 for more details. Since the ARVI COST action workshop, substantial parts of the material presented in this section has been published [11,10]

Instrumentation A considerable number of RV tools use instrumentation as their preferred method of extracting events from the monitored system. Instrumentation is a major hurdle in the introduction of runtime verification mainly due to its inherent intrusive nature (with a single line of code possibly altering the behaviour of the system significantly) and the fact that industry might not need instrumentation support for other aspects of the system apart from RV. This issue is particularly pertinent when applying RV to legacy systems rather than when designing systems from scratch with RV in mind. This is because in the case of the latter, support may be planned upfront and the need for instrumentation may be eliminated altogether (by for example having an explicit system for event generation and communication).

Instrumentation also frequently presents a number of challenges for example when the source code is not available (making it difficult to identify the events of interest) or when some language features are not properly supported by the technology, e.g., intercepting calls to overloaded methods.

Scalability and reliability issues Existing RV tools are also limited in their handling of scalability. For example in an industrial setting, it would typically be useful to aggregate events from across multiple servers. While this has been theoretically studied (e.g., [13,5]), implementations supporting this element are still largely missing. Similarly, most RV tools currently keep monitor data in main memory with two serious problems: firstly, if the system crashes, monitor data is lost and secondly, if the number of monitors or their size grows too large, it might not be sustainable.

Flexibility issues Due to the highly dynamic nature of industry, one would typically expect monitored properties to change from time to time, possibly introducing new ones. Similarly, one might want to switch properties on and off depending on various considerations including the overhead each property introduces versus the available excess resources which can be dedicated for such purposes. While such flexibility can easily be afforded to stateless properties (i.e., ones which do not consider the history), for stateful properties this is a significant challenge: namely, how to efficiently update the monitor state when a monitor starts monitoring from some arbitrary point in the system's lifetime. This issue is also linked with that of reliability since if the system crashes, monitors would also need to somehow recover their state. While these problems have been studied to some extent [9], the current solution depends on significant human intervention.

Visualisation and debugging tools Existing RV tools do not generally provide visualisation and debugging support such as computing the smallest counterexample which would violate the monitored property. Helping developers identify the source of a violation in the code, and explaining how the property in question is violated, would add significant value to RV use. Existing work in this direction exists (e.g., [6]), but once again, the tools available are not ready for industrial use.

Sales pitch and scope Showing the usefulness of RV to industry is not an easy task. The experience of UoM with the financial transaction industry has shown that usually industry sees RV as an extra effort to check for problems which would already have been covered by testing. Therefore, from this perspective, RV was at first considered to be duplication of work with no apparent gain. At this point, UoM was targeting developers as the users of RV. However, developers focus on particular modules and are well-covered

with unit testing. The convincing benefits of RV became clearer when UoM presented RV as a tool for the quality assurance team [11] to have a way of checking the correctness of the system without the need for customers to report errors. Detecting problems ahead of the customers is of utmost priority to the financial transaction industry, as a vulnerability might be exploited if undetected and customers might lose their trust in the services provided.

2.2 Way Forward

This section attempts to give a number of suggestions as a way forward for the RV community towards a wider use of its techniques.

Low-hanging fruit While attempting to take academic knowledge to industry use might seem a hard task with no clear way forward, this section tries to identify a number of ways which might help academia exploit its current resources and achievements to reach industry with less effort:

Asynchronous/Offline monitoring Attempting to provide an RV tool which is able to generate monitors reliable enough to work in a live environment is a very challenging task. However, if monitors are used for offline or asynchronous processing, a number of issues are eliminated: overheads are not an issue, monitor crashes are not that serious, etc. From experience with the financial transaction industry, this has been a way of starting to introduce runtime verification [10]. Since then, it has been the company itself which implemented runtime verification modules within their systems.

Exploit and piggyback on existing popular technologies Trying to sell a standalone RV tool might be quite hard. However, if incorporated with already popular technologies, the task might become easier: For example jUnit[3] and Cucumber[4] are both popular testing tools which have been proposed for explored in this regard. The tool jUnitRV[12] provides a means of using RV monitors within tests instead of assertions. This gives the possibility of significantly more advanced checks on test executions rather than traditional assertions. Another approach currently being explored at UoM under nationally-funded project GOMTA[5], is to translate system level tests written in Gherkin (the language Cucumber uses to define test cases) into monitors[3]. This can be done by applying the checks carried out during testing on the live system if the same testing behaviour is observed at runtime.

Domain specific and controlled natural languages A lot of previous work tasked with identifying an appropriate specification language has assumed that RV end users would be developers. However, from experience with industry [11], developers are typically more concerned with particular submodules of the system rather than the full view of the system. Furthermore, developers are not the ones who usually deal directly with problems occurring after deployment. These two aspects suggest that people such as QA personnel, who have a "user-view" of the system, might be more willing RV users. Naturally, this also means that the kind of properties of interest would for example be more focused on measuring the user experience quality rather than ensuring that iterators are used correctly in the code. Similarly, the notation which appeals to

[3] https://junit.org
[4] https://cucumber.io/
[5] https://www.um.edu.mt/ict/cs/research/projects/gomta

non-technical people is also significantly different from what a developer might find comfortable.

Taking this idea further, the UoM has recently carried out two projects which provide RV to non-technical people by providing a controlled natural language focused on the particular domain. The first project aids non-technical public relations officers keep track of several Facebook pages [7], e.g., getting alerts if a negative comment is posted and this gets more than ten likes in five minutes. The second project is aimed at aiding tax auditors create queries to analyse their databases [2], e.g., show me all tax payers who have declared an income of less than €5,000 for any three years out of the previous five.

Suggestions from the financial transaction industry During the first ARVI COST action meeting which took place in Malta[6], a number of software architects from Ixaris Systems Ltd who attended and presented during the meeting, gave a number of practical suggestions to the RV community based on their experience of collaboration with the UoM. Similarly, SDL Fredhopper has made a number of proposals for the community's consideration. Below we list and expand on these:

Open source software In response to the problem academic institutions face with wanting to develop high quality software for industrial use without having the means of doing so, it was suggested that we open source our tools and try to build a community around them which would both help in developing the tools, as well as use them and provide feedback.

Dynamic switching on and off of properties Given that overheads are particularly problematic for the financial industry, it was suggested that generated monitors provide performance measurements for each running property as well as the possibility of turning on/off such properties on demand.

Choice of property language In view of the difficulty in identifying the right property specification language, it was suggested that specification languages are as close as possible to what industry is already familiar to. For example industry is familiar to Service Level Agreements (SLAs), e.g., the uptime for core elements of the system is 99.9%. Using such format for specifying runtime verification properties might ease adoption by industry.

3 Medical devices

3.1 Runtime Verification for Debugging the Interconnection of Medical Devices

Most of the medical devices are certified and developed following very strict rules. However, it is often desirable for economical reasons to interconnect devices from different manufacturers, and there could be situations where safety-critical properties for the patients cannot be guaranteed at design time of the single devices. In recent years, there has been a great effort on studying medical device interoperability. Several important projects have developed technology and standardisation. Examples include the US-funded multi-institutional community "Medical Device Plug-and-Play Interoperability Program" (MD PNP) and the German flagship project OR.NET project (http://ornet.org/en/) that aims to provide an open standard protocol for the

[6] https://www.cost-arvi.eu/?page_id=138

interconnection of medical devices. These initiatives have promoted the development of many open communication standards based on service-oriented architectures (SOAs) – i.e., Devices Profile for Web Services (DPWS). and the Open Surgical Communication Bus (OSCB) developed within the smartOR project – or on a publish-subscribe architectures such as the Data Distribution Service (DDS). Frameworks such as the Integrated Clinical Environment (OpenICE) enable the connection of several different medical devices and clinical applications together. There is also a dedicated IEEE 11073 family of standards on "Health informatics - Point of Care (PoC) medical device communication / Personal health device communication". In this scenario it is very important to provide a framework that (i) rigorously specifies the communication interfaces between medical devices and between a human and a medical device, and (ii) enforces compliance of the communication to the specification.

Within the OR.NET project, one of the partners of this COST action was involved in building up a communication stack allowing the communication of different medical devices. Runtime Verification techniques have been used to support such developments. Runtime Verification techniques allow the simple to use but precise analysis of communication patterns, typically correctness properties, and thus spot errors early in the implementation phase. Moreover, monitors synthesised from high-level specification may also act as stubs during the testing process. These ideas have be pursued and discussed within the COST action community and published [17,16].

3.2 Quantitative Regular Expressions for Arrhythmia Discrimination Algorithm

In modern medical devices, signal processing (SP) algorithms are tightly integrated with decision making algorithms, so that the performance and correctness of the latter critically depends on that of the former. Thus, analysing the device's decision making in isolation of its SP would provide an incomplete picture of the device's overall behaviour.

For example in Implantable Cardioverter Defibrillators (ICDs) where a Peak Detection (PD) algorithm is first executed on the input voltage signal, known as an electrogram. The PD algorithm generates, as output, a timed boolean signal where a value 1 represents a peak (local extremum) caused by a heartbeat. This boolean output signal is then used by the downstream discrimination algorithms to differentiate between fatal and non-fatal rhythms.

Around 10% of an ICD's erroneous decisions are due to over-sensing (too many false peaks detected) and under-sensing (too many true peaks missed). This leads to an inaccurate estimate of both the heart rate and to an imprecise calculation of the timing relations between the contractions of the heart's chambers.

One of the main challenges is how to formally verify the properties of ICD algorithms for cardiac arrhythmia discrimination. In particular there is a need for a unified language to express and analyse both the PD and discrimination tasks that are currently at the heart of modern ICD technology.

The desired language should be formal (not ambiguous, with a well-defined semantics), expressive enough to be easy to use, and it should guarantee runtime and memory consumption bounds for the resulting programs. One common approach from the computing literature would be to view the ICD tasks as specification-based monitoring problems. Namely, one can try to express the PD and discrimination tasks as require-

ments in some temporal logics: i.e., write a specification which is true exactly when the signal (in a given window) has a peak, and another one which is true exactly when the rhythm (in that window) displays an arrhythmia. Monitor synthesis then automatically translates these specifications into detection algorithms and code.

However, this approach is impractical for the peak detection and discrimination tasks we are concerned with. Despite the increasingly sophisticated variety of temporal logics proposed in the literature, they turn out to be inadequate to express, in one concise formalism, both PD algorithms and arrhythmia discriminators. It is worth noting that PD is a very common signal processing primitive that is employed in many domains beyond medical applications (i.e., earthquake detection), and that arrhythmia discriminators are present in cardiac devices other than ICDs, such as Implantable Loop Recorders and pacemakers. Thus, the observed limitations of temporal logics extend beyond ICD algorithms.

PD and discrimination require performing a wide range of numerical operations over data streams, where the data stream is the incoming cardiac voltage signal (electrogram) observed in real-time.

Motivated by the need for powerful formal methods to reason about the performance of arrhythmia detection algorithms, in, we show how to specify all these algorithms using Quantitative Regular Expressions (QREs). QRE is a formal language to express complex numerical queries over data streams, with provable runtime and memory consumption guarantees. We show that QREs are more suitable than classical temporal logics to express in a concise and easy way a range of peak detectors (in both the time and wavelet domains) and various discriminators at the heart of today's arrhythmia detection devices. The proposed formalisation also opens the way to formal analysis and rigorous testing of these detectors' correctness and performance, alleviating the regulatory burden on device developers when modifying their algorithms. We demonstrate the effectiveness of our approach by executing QRE-based monitors on real patient data on which they yield results on par with the results reported in the medical literature.

3.3 Workshop on Medical Cyber-Physical Systems

In 2016 Martin Leucker and Ezio Bartocci have organised a workshop on Medical CPS at CPS Week 2016[7].

The Medical CPS workshop provides a forum for the presentation of research and development covering all aspects of High Confidence Medical Devices, Software, and Systems (HCMDSS), which is essential to support innovative, networked Medical Device (MD) systems to improve safety and efficiency in health care. The past five workshops have enjoyed a healthy participation of 35-40 attendees, and have provided a working forum for medical device specialists, including researchers, developers, and caregivers from clinical environments, industry, research laboratories, academia, and government with the goal of advancing science, technology, and practice to overcome crucial issues with medical devices, software, and systems and challenges facing the design, manufacture, certification, and use of medical devices. This workshop features medical device and clinical experts from all over the world. The topics covered in the

[7] http://mlab-upenn.github.io/medcps_workshop/

Fig. 1: MedCPS Workshop at CPS Week 2016. From right to left: Martin Leucker, Ezio Bartocci (co-Chairs), Pietro Valdastri (invited speaker), Julian M. Goldman (invited speaker) and Insup Lee.

workshop ranged across all aspects of medical device software modelling and synthesis for safety, assurance, security and control, including but not limited to:

High Confidence Medical Device Software Development and Assurance: Component-based methodologies; design and implementation, verification, validation and testing; robustness and fault-tolerance; system integration and interoperability of heterogeneous systems; closed-loop control; systems of systems; requirements solicitation and capture; clinical data management, and data security.

Modelling and Simulation of operational scenarios: modelling of failures in medical devices; caregivers, and patients' behaviour modelling; high fidelity organ/patient models for design and testing; Pharmacokinetic and Pharmacodynamic (PK-PD) models; and machine learning models.

Embedded, Real-Time, Networked HCMDSS: Architecture, platform, middleware, resource management, QoS (Quality of Service) in HCMDSS, dynamic interoperation in HCMDSS, including MD PnP (Plug-and-Play) operation, and distributed control.

Enabling Technologies for Future Medical Devices: telemedicine, biosensor technologies, implantable devices, energy harvesting and remote powering devices, medical ultrasound systems, robotic surgery, and physiologic signal QoS (Quality of Service).

Medical practice: User-centric design; use and misuse of MD; risk understanding, and management of failures; medical guidelines and regulations.

Certification of HCMDSS and MD Interoperability: Regulatory fundamentals, laws, CE marking, FDA approval, pre-clinical testing, clinical evaluation, regulations applicability, incremental certification, role of design tools, approval of non-deterministic and self-adaptive MD systems.

4 Electronic contracts

Contracts are defined to be agreements between two or more parties, which determine their rights and obligations with respect to each other. Electronic contracts have been used in the literature to refer to contracts which are processed through computer systems, whether they are used to support the drafting, negotiation, monitoring or management of the contracts. Over the years, the term has come to refer to a wide range

of artefacts, from legal contracts which are stored electronically, but not so processed, to executable software specifications. In this section we focus on the more legal side of this spectrum, in particular since it can be argued that runtime verification has, from its birth, dealt with the other end of the spectrum —- contracts which coincide with decidable specifications.

4.1 Contract Representation

With the recent rise of Regtech (the use of new technology to facilitate the delivery and treatment of regulatory requirements), the industry is increasingly showing interest in finding means of supporting the regulatory process and legal document processing through automated means. Alas, it has become increasingly clear that one of the major challenges is that almost all legal texts are written in natural language, and although typically it is heavily codified in style, automated techniques to extract a machine-processable form of such texts, to enable applying semantic (or even syntactic) analysis techniques is still a huge challenge. Contract representation is thus one of the major challenges in the field.

One can broadly categorise the different approaches taken in the industry into three: (i) an extract-what-you-can approach, in which natural language processing techniques are used to extract partial information from legal texts for automated analysis; (ii) template-based legal texts and contracts which, thanks to the limited syntactic form of the texts can be given an exact semantics; and (iii) specialised controlled languages, natural language or logic-based to represent more general legal texts for complete syntactic and semantic analysis. The first approach is the one which can be directly used on existing legal texts and contracts, but suffers from the information extracted being limited. The situation in which this approach has proved to be particularly useful is that of large repositories of existing contracts written in natural language, and which may require far too much legal expertise to process manually. Although the approach allows for smarter filtering of contracts (e.g. allowing for searches for obligations on the company related to a particular service) and surface analysis and categorisation, the partial representations obtained are rarely sufficiently complete to enable their use as oracles in automated monitoring and adjudication. The second template-based approach is particularly useful in domains where one typically adopts a number of clauses from a family of standard forms with little variation apart from parameterisation. For instance, in the domain of lease agreements, or simple house-sale agreements, the clauses of the contracts vary little, and range over a relatively small number of possibilities. Such forms contracts can easily be given an exact semantics and their execution or verification to be automated. Finally, the third form of controlled languages lies somewhere in between the two previous approaches —- while one can express a wider variety of contracts, it requires a greater effort to do so. What one gains in expressivity, one loses in ease of transcribing contracts in such languages, limiting the applications for such an approach.

Although there is much work in academia on the use of deontic logics to express the semantics of legal texts, the challenges real-world systems face go beyond the use of a clean compositional semantics, particularly since legislation and contracts are typically organised in articles or clauses which may refer to each other and to their breaching. For instance, one frequently finds clauses in contracts of the form “In case of clauses x to y being violated...”, which make compositional analysis more challenging.

4.2 RV and Contracts in the Industry

Given the challenges which exist in contract representation, and given the need for concrete semantics to perform runtime verification, few industrial systems provide some form of monitoring of behaviour and flagging contract violations. Many existing systems focus on manual transcription of legal texts into an executable monitor, for instance in the case of service-level agreements, especially parts which pertain to quantitative quality-of-service constraints. Another domain which has been increasingly given attention recently is that of automated legal compliance checking, particularly for electronic services in domains which are tightly regulated, such as financial transactions and online gambling. Unlike the typical usage pattern of most academic runtime verification systems which tend to attempt to separate the monitoring and verification parts from the main system, in these cases mentioned, the dynamic analysis is typically seen as an integral part of the system and its functionality. Although the architecture of the system may separate the logic managing transactions from that checking for compliance, it is typically engineered as an enforcement engine rather than a verification one – in that it adds functionality to the rest of the system to identify and act on legal violations based on the transactions performed, rather than verifying that the system is working correctly.

In recent years, the notion of smart contracts has taken traction. Although proposed at least as far back as the 90s, the notion of contracts being expressed as executable code which enforces behaviour, rather than verifying it has shot up in use in industry since the advent of blockchain and other distributed ledger technologies (DLT). The ability to enforce and guarantee contracts over digital assets in an immutable manner has deep implications in the domain of legal contracts. As long as one remains within the DLT domain, managing digital assets which reside in the underlying DLT can be regulated directly using smart contracts – a process akin to runtime enforcement. However, when interfacing with the external world, one typically has to introduce external points-of-trust (e.g. sensors to be able to measure whether a physical product has been used), a number of industry projects have sprouted which implement real world contracts to verify compliance (or otherwise) of the parties involved, which is more akin to the scenario typically found in runtime verification. The current challenge is that of linking legal contracts with smart ones – not just translating the former into the latter, but also in having the two interact together, for instance by having smart contracts acting as runtime verifiers on the behaviour of the parties, and with a legal contract asserting consequences of the violation being flagged.

4.3 The Future

The use of runtime verification techniques for electronic contracts is still limited, but with improvement of natural language techniques, and with contracts guaranteed to be executed on DLT-like systems, there is still much to be achieved in the coming years. For instance, in the domain of contract negotiation, runtime verification can play an important role in monitoring the process and highlighting patterns and plan responses.

One of the main challenges today is how to extract a runtime monitor from a declarative legal contract in natural language. Given the complexity of this task, the hope is then that more and more (law) companies work towards a standardisation of a controlled natural language, close enough to the way lawyers write contracts but with a

formal grammar, in order to breach this gap. Once this is in place, we believe that much of the existing research could be applied to a certain extent in order to automate the analysis and enforcement of electronic contracts.

Concerning smart contracts in particular (as defined on top of blockchain) the area is still in its infancy and more investment is needed (both from academy and industry) in order to make contract enforcement a reality.

5 Security and Privacy

In the last years there has been an explosion in the availability of large volumes of data. Large integrated datasets can potentially provide a much deeper understanding of both nature and society and open up many new avenues of research. They are critical for addressing key societal problems, from offering personalised services, improving public health and managing natural resources intelligently, to designing better cities and coping with climate change. Applications are deployed in our smart devices and used by our browsers in order to offer better services. Everything has a price though: on one side most services are offered in exchange of personal data, on the other side the complexity of the interaction of such applications and services makes it difficult to understand and track what they have access to, and what they do with our data. Privacy and security are thus at stake.

Given the breadth of the Security & Privacy domain, we will not be able to be exhaustive on the analysis the different application areas. We have thus deliberately focused our attention only on a very small subset of the whole area, mainly privacy concerns from the EU General Data Protection Regulation (GDPR) and browser extensions. Even within those specific areas, we have only been able to touch the surface and only mention a small part of a much bigger picture.

5.1 GDPR (General Data Protection Regulation)

The European privacy regulation, the General Data Protection Regulation (GDPR - EU-2016/679, adopted on 27 April 2016 and that entered into application on 25 May 2018), subject companies, governmental organisations and any other data collector to stringent obligations when it comes to user privacy in their digital products and services. Systems will need to be designed having privacy in mind (privacy-by-design), and existing systems will have to provide evidence about their compliance with the coming rules. This will be mandatory, and sanctions for data breaches will get tougher.

As an example, we mention here Article 5 of GDPR, related to the so-called data minimisation principle: “Personal data must be adequate, relevant, and limited to the minimum necessary in relation to the purposes for which they are processed”. While determining what is “adequate” and “relevant” might seem difficult given the inherent imprecision of the terms, identifying what is “minimum necessary in relation to the purpose” seems to be easier.

One way to understand minimisation is on how the data is used, that is we could consider ways to identify how the input data is used in the program, and for which purposes. In this case we would need to look inside the program and track the usage of the data by performing static analysis techniques like tainting, def-use, information flow, etc. For that we need a precise definition of what “purpose” means and a way to

check that the intended purpose matches with the real purpose under which data will be processed.

Another aspect of minimisation is when and how the data is collected and only allow the collection of data that is actually needed to compute what is required to achieve the given purpose. In this case we could consider that the "purpose" is given by the specification of the program. This is the approach followed by Antignac et al. Data minimisation might be enforced technically, at least in what concerns some of its aspects. Other privacy principles might not be that easy.

For the runtime verification community one of the main challenges is to identify which privacy principles might be verified or enforced by using monitors. As the regulation is quite extensive, we suggest to start with the principle of data minimisation as an example of the kind of challenges the community might face.

When considering how the data is used, a challenge is that we will not be able to do runtime verification in a black box manner. Getting access to proprietary code might be an issue. Concerning when and how the data is collected we could do runtime verification in a black box manner, but data minimisation is not monitorable in general. For the more general notion of distributed data minimisation, the property is not monitorable in general, therefore new techniques using grey box runtime verification might be needed.

In this area, there has not been much work of RV in practice, given that it is a relatively new area.

5.2 Browser Extensions

Browser extensions are small applications executed in a browser context in order to provide additional capabilities and enrich the user experience while surfing the web. The acceptance of extensions in current browsers is unquestionable. For instance, Chrome's official extension repository has more than 140,000 applications, with some of these extensions having more than 10 million users. When an extension is installed, the browser often pops up a message showing the permissions that this new extension requests and, upon user approval, the extension is then installed and integrated within the browser. Extensions run through the JavaScript event listener system. An extension can subscribe to a set of events associated with the browser (e.g., when a new tab is opened or a new bookmark is added) or the content (e.g., when a user clicks on an HTML element or when the page is loaded). When a JavaScript event is triggered, the event is captured by the browser engine and all extensions subscribed to this event are executed.

Research on the understanding of browser extensions, detecting possible privacy and security threats, and mitigating them is still in infancy. The potential danger of extensions had been highlighted for instance by Heule et al. where extensions have been identified to be "the most dangerous code to user privacy" in today's browsers. Some recent works have focused on tracking the provenance of web content at the level of DOM (Document Object Model) elements.

Another issue is the order in which extensions are executed. Indeed, when installed, extensions are pushed to an internal stack within the browser. This implies that the last installed extension is the last one that will be executed. In recent work, some experiments have been performed to demonstrate that this order could be exploited by an unprivileged malicious extension (i.e., one with no more permissions than those already assigned when accessing web content) to get access to any private information

that other extensions have previously introduced. To the best of our knowledge, there is no solution to date to this problem (we have not seen any paper even identifying this as a problem).

Last but not least, there is the problem of collusion attacks, that is when two or more extensions may collaborate to get even more information from the user based on the individual permissions each extension has (in isolation they cannot do any harm, but this could be done by collaborating and combining their privileges). This is a relatively unexplored area, with the exception of very few papers.

Given that extensions may subscribe to events after they have been installed (i.e., at runtime), there is no way to statically detect potential attacks. One of the few works providing a runtime solution to information flow in browsers (Chromium in particular) is by Bauer et al.

Overall, there still is a lot to do in what concerns studying the effect of browser extensions regarding security and privacy, and given the limitations of what may be done statically, solutions to mitigate such issues could significantly benefit from runtime monitoring techniques.

One challenge is on the enforcement side: how to ensure that malicious extensions do not expose private information from a user's homepage. This private leakage might be done by an external entity or by another extension which may aggregate this information with the information it already has eventually performed a collusion attack.

A related issue has to do with implementation: a robust runtime enforcement mechanism might need to modify the core of the browser (e.g., Chromium). This is quite invasive and might require a high level of expertise.

5.3 Workshop on Security and Privacy at ARVI

As part of our activities in this action, we have organised a workshop on Security and Privacy (S&P) in Siracusa (Sicily, Italy). The main objective of the workshop was to explore the interplay between RV and S&P. This was achieved by asking researchers in S&P, both inside and outside the RV community, to talk about challenges where RV could be applied.

The expected outcome was to have new insights on their domain (and vice-versa) hoping to establish new contacts and collaboration. Abstracts of the talks are available at the workshop homepage[8].

6 Electrical Energy Storage

The ongoing transition towards a more sustainable, "green" economy — including the shift to renewable but fluctuating energy sources like wind and solar power, the development of self-sustaining smart homes, and locally emission-free electromobility — critically depends on technologies to efficiently and safely store electrical energy.

Lithium-ion batteries are currently amongst the most advanced and efficient technologies for electrochemical energy storage.

However, lithium-ion battery cells have a relatively narrow window in terms of voltage, current, and temperature in which the battery cells can be safely operated. Operation outside the safe margins can lead to permanent battery damage of the cells and

[8] https://www.cost-arvi.eu/?page_id=1431

potentially dangerous situations. For instance, for the wide-spread nickel-manganese-cobalt (NMC) lithium-ion cell chemistry with graphite electrodes, discharging the cells to a too low voltage (below 2.5 Volt) may lead to dissolving of copper from the cell's anode and subsequent short-circuits, whereas charging cells to a too high cell voltage (above 4.2 Volt) may lead — especially at low temperatures — to deposition of metallic lithium and a permanent capacity reduction of the cell. A too high cell temperature (above 60 degrees celsius) may lead to a so-called thermal runaway, in which the release of gas can lead to fire and even an explosion of the battery.

To avoid the potential hazards, each lithium-ion battery cell therefore needs to be closely monitored and the overall battery system needs to be equipped with electric safety switches, which disconnect the circuit in an emergency situation before the system leaves the safe operating regions. In addition, a multitude of regulatory and legal requirements have to be met, for instance functional safety norms like IEC 61508 for electrical/electronic systems, or ISO 26262 for road vehicles.

A lithium-ion battery pack therefore consists of several individual battery cells together with an electronic battery management system (BMS). The main task of the battery management system is to safely manage the charging and discharging of the battery cells, while ensuring a maximum availability of the energy stored in the system. The battery management system may monitor the state of the battery as represented by various sensor measurements, such as voltage (total voltage, voltages of individual cells, or minimum and maximum cell voltage), temperature (average temperature, coolant intake temperature, coolant output temperature, or temperatures of individual cells), and current (current in or out of the battery). In addition to monitoring these observable values, the battery management system has to provide an estimate of the battery's internal, unobservable states, in particular the state of charge (SOC), state of health (SOH), and available power (SOF).

Typically, battery packs that are used in automotive or stationary storage applications consist of several hundreds or thousands of individual cells, which are connected in series and/or parallel to achieve the required current and voltage output level. This complexity of battery packs puts high demands on battery management systems. Various topologies for battery management systems exist and among the most widely used is a distributed master-slave architecture with several dedicated measurement boards (slaves) and a central processing board (master), which communicate data internally through a bus system.

While previous generations of battery systems operated with large safety margins, a current trend in battery systems is to widen the operating windows as far a possible to maximise for instance the driving range of electric vehicles. For example, in the on-going Horizon2020 project EVERLASTING (Electric Vehicle Enhanced Range, Lifetime And Safety Through INGenious battery management), novel on-line state-estimation methods based on physico-chemical models are developed to predict the internal behaviour of the battery cells more precisely and therefore to increase the usable capacity of the cells without compromising safety and reliability. However, this requires more frequent and accurate sensor readings, increasing the required bandwidth and speed of communication, and also requires increased computing power of the battery management system to run these models.

Especially in automotive applications, there is therefore a trend to shift from traditional vehicle bus systems like CAN bus to high-speed bus standards like industrial ethernet. Another trend is to replace wire-based communication buses by optical or wireless communication networks, which can reduce weight and increase flexibility but may on the other hand introduce new reliability and security problems.

An overall trend for electric vehicles, especially with the simultaneous advent of computing-intensive autonomous driving functions, is the consolidation of the electronic hardware and software architecture. This means that the functions of various vehicle sub-systems, including the battery management, are virtualised on a few centralised, powerful processors rather than running on their own dedicated control units. It is then necessary that these platforms provide guarantees on the execution time in order not to compromise the behaviour of safety-relevant battery management functions.

Finally, the increasing proliferation of battery systems creates a need for reliably deciding when the battery has reached its end of useful life. For instance, for electric cars this is typically defined by the battery capacity having dropped to around 80% of its original capacity. The battery then either has to be recycled or it can be used in less demanding second-life applications like stationary storage or peak-shaving. In this context, the battery management system should securely provide key parameters for this decision, for instance in the form of a non-modifiable battery gauge.

While runtime verification methods could be applied in the area of electrical energy storage, some of the challenges are:

- large amount of data (for example, voltage measurements are typically sampled every 10 milliseconds for each battery cell, which can easily amount to data streams of several megabits in large battery systems)
- intermittent connectivity (in mobile applications) and limited computational resources (in terms of runtime and memory), which also leads to the challenge of finding the right balance between performing computations on a connected, centralised platform versus locally in the embedded system itself (edge computing)
- coupling of a monitoring system with more higher-level applications and tasks, like diagnosis and planning (for instance, pertaining to the overall control strategy of a vehicle drive train or a home stationary storage system)
- safety-relevant applications, where norms like ISO 26262 require, depending on the safety integrity level, the restriction of the safety-critical control and monitoring software to certain coding constructs and language subsets like MISRA-C, which are typically not in the focus of RV research

7 Way Forward

This section attempts to give a number of suggestions as a way forward for the RV community towards a wider use of its techniques.

7.1 Engineer high quality tools

Whether by going for open source (as suggested by industry) or by putting in more resources, industrial quality RV tools have to be built to facilitate and encourage use. This can be done by incorporating standard quality processes in the development of RV tools and using benchmarks and test frameworks to show their robustness. Positive

developments in this regard have been made through the RV competition [18] which compares the tools against each other and naturally benchmarks them with respect to a number of systems. Another tool SMock [8] provides a testing infrastructure for RV tools through the use of a scripting language to quickly generate test systems which can generate custom streams of events.

7.2 Establishing better relations with industry

To facilitate the interaction between the RV community and industry, a number of efforts may be undertaken:

Joint projects with industry One way of improving the communication between academia and industry is to increase the number of jointly undertaken projects, possibly first simply having student interns, to establish a working relationship and a common language which would enable the parties to understand each other.

Selling RV the right way The RV tools currently available are generally generic tools, offering basic RV capabilities. While such tools provide most RV functionality industry might need, in reality they are possibly an overkill. Packaging RV tools into focused sub-tools with specific purposes might facilitate adoption by industry, e.g., packaging an RV tool as a stream processor, a debugging tool, a querying tool, etc.

7.3 Case study database

Keeping a database of case studies can help the community remain up to date with the latest developments, as well as having the possibility of sharing case studies where this is possible. The ARVI COST website is now hosting a database of such case studies at https://www.cost-arvi.eu/?page_id=241. Plans are being considered to take this forward and have a dedicated platform for RV-related resources.

References

1. Baresi, L., Ghezzi, C.: The disappearing boundary between development-time and run-time. In: Roman, G., Sullivan, K.J. (eds.) Proceedings of the Workshop on Future of Software Engineering Research, FoSER 2010, at the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2010, Santa Fe, NM, USA, November 7-11, 2010. pp. 17–22. ACM (2010), https://doi.org/10.1145/1882362.1882367
2. Calafato, A., Colombo, C., Pace, G.J.: A controlled natural language for tax fraud detection. In: Davis, B., Pace, G.J., Wyner, A.Z. (eds.) Controlled Natural Language - 5th International Workshop, CNL 2016, Aberdeen, UK, July 25-27, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9767, pp. 1–12. Springer (2016), https://doi.org/10.1007/978-3-319-41498-0_1
3. Cauchi, A., Colombo, C., Francalanza, A., Micallef, M., Pace, G.J.: Using gherkin to extract tests and monitors for safer medical device interaction design. In: Luyten, K., Palanque, P.A. (eds.) Proceedings of the 8th ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS 2016, Brussels, Belgium, June 21-24, 2016. pp. 275–280. ACM (2016), https://doi.org/10.1145/2933242.2935868
4. Colin, S., Mariani, L.: Run-time verification. In: Model-Based Testing of Reactive Systems. Lecture Notes in Computer Science, vol. 3472, pp. 525–555 (2004)
5. Colombo, C., Falcone, Y.: Organising LTL monitors over distributed systems with a global clock. Formal Methods in System Design 49(1-2), 109–158 (2016), https://doi.org/10.1007/s10703-016-0251-x

6. Colombo, C., Francalanza, A., Grima, I.: Simplifying contract-violating traces. In: Pace, G.J., Ravn, A.P. (eds.) Proceedings Sixth Workshop on Formal Languages and Analysis of Contract-Oriented Software, FLACOS 2012, Bertinoro, Italy, 19 September 2012. EPTCS, vol. 94, pp. 11–20 (2012), https://doi.org/10.4204/EPTCS.94.2
7. Colombo, C., Grech, J., Pace, G.J.: A controlled natural language for business intelligence monitoring. In: Biemann, C., Handschuh, S., Freitas, A., Meziane, F., Métais, E. (eds.) Natural Language Processing and Information Systems - 20th International Conference on Applications of Natural Language to Information Systems, NLDB 2015 Passau, Germany, June 17-19, 2015 Proceedings. Lecture Notes in Computer Science, vol. 9103, pp. 300–306. Springer (2015), https://doi.org/10.1007/978-3-319-19581-0_27
8. Colombo, C., Mizzi, R., Pace, G.J.: Smock - A test platform for monitoring tools. In: Legay, A., Bensalem, S. (eds.) Runtime Verification - 4th International Conference, RV 2013, Rennes, France, September 24-27, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8174, pp. 352–357. Springer (2013), https://doi.org/10.1007/978-3-642-40787-1_24
9. Colombo, C., Pace, G.J.: Fast-forward runtime monitoring - an industrial case study. In: Qadeer, S., Tasiran, S. (eds.) Runtime Verification, Third International Conference, RV 2012, Istanbul, Turkey, September 25-28, 2012, Revised Selected Papers. Lecture Notes in Computer Science, vol. 7687, pp. 214–228. Springer (2012), https://doi.org/10.1007/978-3-642-35632-2_22
10. Colombo, C., Pace, G.J.: Considering academia-industry projects meta-characteristics in runtime verification design. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice - 8th International Symposium, ISoLA 2018, Limassol, Cyprus, November 5-9, 2018, Proceedings, Part IV. Lecture Notes in Computer Science, vol. 11247, pp. 32–41. Springer (2018), https://doi.org/10.1007/978-3-030-03427-6_5
11. Colombo, C., Pace, G.J.: Industrial experiences with runtime verification of financial transaction systems: Lessons learnt and standing challenges. In: Bartocci, E., Falcone, Y. (eds.) Lectures on Runtime Verification - Introductory and Advanced Topics, Lecture Notes in Computer Science, vol. 10457, pp. 211–232. Springer (2018), https://doi.org/10.1007/978-3-319-75632-5_7
12. Decker, N., Leucker, M., Thoma, D.: junit^{rv}-adding runtime verification to junit. In: Brat, G., Rungta, N., Venet, A. (eds.) NASA Formal Methods, 5th International Symposium, NFM 2013, Moffett Field, CA, USA, May 14-16, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7871, pp. 459–464. Springer (2013), https://doi.org/10.1007/978-3-642-38088-4_34
13. Francalanza, A., Gauci, A., Pace, G.J.: Distributed system contract monitoring. J. Log. Algebr. Program. 82(5-7), 186–215 (2013), https://doi.org/10.1016/j.jlap.2013.04.001
14. Giannakopoulou, D., Pasareanu, C., Lowry, M., Washington, R.: Lifecycle verification of the nasa ames k9 rover executive. In: ICAPS'05 Workshop on Verification and Validation of Model-Based Planning and Scheduling Systems (2005)
15. Leucker, M., Schallhart, C.: A brief account of runtime verification. The Journal of Logic and Algebraic Programming 78(5), 293 – 303 (2009), http://www.sciencedirect.com/science/article/pii/S1567832608000775
16. Leucker, M., Schmitz, M.: Secured SOA for the safe interconnection of medical devices (position paper). In: Zimmermann, W., Böhm, W., Grelck, C., Heinrich, R., Jung, R., Konersmann, M., Schlaefer, A., Schmieders, E., Schupp, S., y Widemann, B.T., Weyer, T. (eds.) Gemeinsamer Tagungsband der Workshops der Tagung Software Engineering 2015, Dres-

den, Germany, 17.-18. März 2015. CEUR Workshop Proceedings, vol. 1337, pp. 11–14. CEUR-WS.org (2015), http://ceur-ws.org/Vol-1337/paper3.pdf

17. Leucker, M., Schmitz, M., à Tellinghusen, D.: Runtime verification for interconnected medical devices. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications - 7th International Symposium, ISoLA 2016, Imperial, Corfu, Greece, October 10-14, 2016, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9953, pp. 380–387 (2016), https://doi.org/10.1007/978-3-319-47169-3_29
18. Reger, G., Hallé, S., Falcone, Y.: Third international competition on runtime verification - CRV 2016. In: Falcone, Y., Sánchez, C. (eds.) Runtime Verification - 16th International Conference, RV 2016, Madrid, Spain, September 23-30, 2016, Proceedings. Lecture Notes in Computer Science, vol. 10012, pp. 21–37. Springer (2016), https://doi.org/10.1007/978-3-319-46982-9_3