Invited paper, presented at the 12th International Conference on Distributed Computing and Internet Technology (ICDCIT), Bhubaneswar, India, January 2016; published by Springer LNCS Vol 9581, pp. 19–29.

Trustworthy Self-Integrating Systems

John Rushby

Computer Science Laboratory
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025 USA

Abstract. Patients in intensive care often have a dozen or more medical devices and sensors attached to them. Each is a self-contained system that operates in ignorance of the others, and their integrated operation as a system of systems that delivers coherent therapy is performed by doctors and nurses. But we can easily imagine a scenario where the devices recognize each other and self-integrate (perhaps under the guidance of a master "therapy app") into a unified system. Similar scenarios can be (and are) envisaged for vehicles and roads, and for the devices and services in a home. These self-integrating systems have the potential for significant harm as well as benefit, so as they integrate they should adapt and configure themselves appropriately and should construct an "assurance case" for the utility and safety of the resulting system. Thus, trustworthy self-integration requires autonomous adaptation, synthesis, and verification at integration time, and this means that embedded automated deduction (i.e., theorem provers) will be the engine of integration.

1 Introduction

An invited paper provides an opportunity for more speculative inquiry than usual, and I will use this chance to sketch some of the challenges and opportunities in a class of systems that I think is just around the corner, but that does not seem to be widely recognized.

We are familiar with systems built from components, and are becoming so with systems of systems. Components are intended as parts of a larger whole and their interfaces and functionality are designed with that in mind. *Systems*, on the other hand, are intended to be self-sufficient and to serve a specific purpose, so they often cooperate awkwardly when combined as systems of systems. *Open* systems are intended to cooperate with others and they restore some of the characteristics of components while still operating as self-sufficient individually purposeful systems. Open *Adaptive* Systems (OAS), a popular topic of current interest, are open systems that are capable of adjusting their behavior to function better in systems of systems.

The systems that I am interested in here take this one step further: they do not merely adapt to cooperate better with their neighbors, but *self-integrate* to deliver some new capability or service, possibly under the direction of an application designed for that purpose. The integrating application or its surrogates

will seek useful capabilities among its peers, cause them to adapt or configure appropriately, and synthesize suitable wrappers, shims, or glue. This behavior has something in common with Service Oriented Architecture (SOA) as well as OAS, but what I want to posit is that the capabilities and services so constructed will be used for somewhat critical purposes where assurance of function and safety should be provided. Skeptics may concur that self-integrating systems are likely, but not their application as critical systems that require assurance. Accordingly, in the next section I sketch some scenarios proposed by others and then, in Section 3 outline some current and recent work. In Section 4, I explain why I believe that automated verification and synthesis are required at integration-time and I describe how modern developments in automated deduction can provide these. Section 5 summarizes and concludes that embedded automated deduction is the enabling engine for trustworthy self-integration.

2 Scenarios

We can build quite complex software-intensive systems that are individually safe: civil aircraft are a good example, as there have been no serious aircraft incidents due to faulty software.[1] There is a price to this record of safety, however, for current methods of safety assurance are based on strongly predictable, often deterministic, behavior. This prohibits even simple kinds of adaptive or intelligent behavior, such as control laws that optimize for the circumstances of each flight (e.g., weight and weight distribution, stiffness of actuators, etc.).

Furthermore, assurance applies to each system in isolation: there are no guarantees that the interaction of individually safe systems will be safe. Thus, although individual aircraft are safe, their interaction is managed by another, separate system for air traffic management. The move to NextGen, where individual aircraft take more responsibility for the safety of their interactions, as cars do on the road, is proving to be a challenging endeavor.

The future will be about interacting systems. Interaction will come about because, as systems become more ubiquitous and embedded in our lives, so they necessarily interact with each other through the "plant." For example, cars and traffic lights interact to create, or to avoid, traffic jams. Currently, there is (at least in the USA) essentially no management of this interaction, so cars waste gasoline accelerating and decelerating from one uncoordinated red light to another, or lights show green while traffic cannot move because it is backed up from the uncoordinated red light ahead. As systems become better able to communicate, so we expect them to become more open and to integrate in a productive manner, so that traffic lights and cars should communicate and respond to changing traffic and to the behavior of neighboring lights. Later, conventional traffic lights can be replaced or supplemented by virtual traffic lights that interact directly with car systems and can be deployed whenever and wherever needed.

[1] Due to faulty requirements, there have been incidents in systems implemented in software, but there have been no incidents due to software development.

Integration of large scale systems such as air and ground transportation presents formidable challenges, so it is not likely they will self-integrate any time soon, but it seems quite plausible for smaller systems. We already see this in the nascent "Internet of Things" (IoT). For example, I recently added a Chromecast to the large TV display in my living room, turned on the DLNA server built in to the Windows machine that stores my digital photos, and installed an app on my Android phone; I can now view my photos on the large display (transmitted over WiFi from the PC in another room) while using my phone to control everything from my couch. It is remarkable that this works at all, still more remarkable how easy it is to set up, but sadly it does not work very well. The photos are transmitted at full resolution despite the limited (1920x1080) resolution of the display. My WiFi is slow and the PC is in a room distant from the router; as a result one photo may still be in the process of transmission when the timer in the Android app calls for the next one; then everything hangs; sometimes a restart is sufficient, and sometimes a full reboot is needed. Viewing my photos is hardly a critical endeavor, but it is not uncommon for things that work "well enough" to undergo "mission creep" and become part of some critical enterprise (e.g., we could imagine my setup for home photos being used to display images needed during medical procedures), which is fine—until things fail.

Goldman and colleagues describe several intriguing applications for a more trustworthy IoT in a hypothesized "Operating Room of the Future" and "Intensive Care Unit of the Future" [1]. I sketch a few of these.

Some seriously ill patients are maintained on a heart-lung machine while undergoing surgery. And sometimes an X-ray is required during the procedure. Surgeons may temporarily turn off the heart-lung machine so the patient's chest is still while the X-ray is taken. They must then remember to turn it back on. We can posit a scenario where the heart-lung machine and the X-ray camera recognize each other and negotiate their safe interaction. In the simplest case, the camera could request a still period from the heart-lung machine; in a more attractive approach, the heart-lung machine could notify the camera of the null points during its inflation and deflation of the chest.

A patient under general anesthesia is generally provided an enriched oxygen supply. Some throat surgeries use a laser and this can cause burning (or even fire) in the presence of abundant oxygen, so the human anesthetist does not enrich the oxygen supply in this case. There is an obvious risk of human error in this scenario. Accordingly, it would surely be good if the laser and the anesthesia machine could recognize each other so that the laser could request reduced oxygen. Of course, we do not want other (possibly faulty) devices to cause the oxygen supply to be reduced. It is also possible that a faulty anesthesia machine may not reduce the oxygen, so we would like a safety interlock that does not allow the laser to light unless the oxygen has actually been reduced. Conversely, there may be emergency scenarios where the patient's health or survival urgently needs enriched oxygen (this may be detected by a "pulse oximeter," the third component of a larger integrated system) and we would like the combined sys-

tem to support this, either by an override that autonomously shuts off the laser and enriches the air supply, or by alerting the human operators.

Accurate blood pressure sensors can be inserted into an intravenous (IV) fluid supply. The reading needs to be corrected for the difference in height between the sensor (which can be standardized by the height of the IV pole) and the patient. Some hospital beds have a height sensor, but this is a fairly crude device to assist nurses in their activities. We can imagine an ICU where blood pressure data from IV sensors and height measurements from the beds are available on the local network and integrated by monitoring and alerting services. These services need to be sure the bed height and blood pressure sensor readings are from the same patient, and there needs to be an ontology that distinguishes height-corrected and uncorrected sensor readings. The noise- and fault-characteristics of the bed height sensor mean that alerts should be probably driven from changes in the uncorrected reading; alternatively, since bed height will seldom change, it is possible that a noise and fault-masking wrapper could be synthesized for this value.

A machine for Patient Controlled Analgesia (PCA) administers a pain-killing drug to patients on demand (when the patient presses a button). To prevent overdoses, PCA devices will not deliver the drug when thresholds in a built-in model (whose parameters can be programmed by a nurse) are exceeded. The thresholds are conservative, so patients may sometimes experience unrelieved pain unnecessarily. A pulse oximeter (PO) attached to the patient provides a specific indication of overdose, so the combination of a PCA and PO could provide safer and greater relief. We can imagine the combination of a standard PCA, a PO, and an application that manipulates the thresholds of the PCA based on data from the PO to allow this improved capability. As with the blood pressure example, we need to be sure that the two devices are attached to the same patient, and that all parties interpret the measurements consistently (i.e., there is a shared ontology). Furthermore, the integrating app and its assurance case must deal suitably with new hazards due to integration and possible faults therein. For example, if the app works by blocking button presses when an approaching overdose is indicated, then loss of communication could remove the safety function. If, on the other hand, it must approve each button press, then loss of communication may affect pain relief but not safety. In both cases, it is necessary to be sure that faults in the blocking or approval mechanism cannot generate spurious button presses.

Most will agree, I think, that the integrated systems sketched above could be readily constructed as bespoke systems by suitably skilled teams. But what I have in mind is that these systems "self assemble" from their separate component systems given little more than a sketch of the desired integrated function. The sketch might be a formal specification, or an idealized reference implementation that assumes fault-free and safe operation of the individual systems.

Beyond automation of this self assembly, the challenge is to provide assurance for safety of the integrated system. The state of the art in safety assurance is the assurance "case," which is an argument, based on evidence about the system and

its design and construction, that certain claims (usually about safety but possibly about other important properties such as security or performance) are true [2]. When systems interact, we would like the assurance case for their composition to be assembled in a modular or compositional way from the cases for the individual systems. This is difficult because, as we noted previously, safety generally does not compose: that is to say the composition of safe systems is not necessarily safe, primarily because their interaction may introduce new hazards. For example, the laser and anesthesia machine may be individually safe, but their integration has a new hazard (burning in an oxygen-enriched air supply). The construction of the joint assurance case is therefore a fairly difficult process, typically requiring human insight, which presupposes that system integration is a deliberate and planned procedure.

For the near term, I expect that system self-integration will be initiated by an application that embodies the purpose of the integration (e.g., a "safe analgesia app" that integrates a PCA and PO) but we can imagine that future systems will integrate spontaneously as they discover each other, similar to the way that human teams assemble to solve difficult problems (although even here there must be some agreed purpose for the teaming). And as with human teams, the integration may involve exchange of assumptions and claims, negotiation, and some relinquishing of autonomy and acceptance of constraints. Clearly this is an ambitious vision, but there is recent and current work that addresses many of the precursor challenges.

3 Recent Work

DEOS (Dependable Operating Systems for Embedded Systems Aiming at Practical Applications) was a large project in Japan that ran from 2008 to 2013. The overall ambition of DEOS was to ensure the dependability of open systems subject to change [3]; thus, it focuses on the evolution of systems rather than their self-integration. In DEOS, a system assurance case is maintained as part of the system and is used to guide its adaptation to failure and to changed requirements. For local adaptation (e.g., responding to failure), an online representation of the assurance case (called a D-Case) is used to guide adjustments to the system, potentially automatically, under the guidance of a scripting language call D-Script and associated tools [4]. Human intervention is required for larger adaptations, but this is assisted by, and maintains, the assurance case.

The Semantic Interoperability Logical Framework (SILF) was developed by NATO to facilitate dependable machine-to-machine information exchanges among Command and Control systems [5]. SILF employs an extensive ontology to describe the content of messages exchanged, and a mediation mechanism to translate messages as needed. The mediation can be performed by a centralized hub, or by wrappers at either the sender or receiver. ONISTT [6] is an SRI project that developed and prototyped many of the ideas in SILF; it was primarily employed to enable the integration of live and virtual simulation systems for military training. Using ontological descriptions, ONISTT is able automatically to synthesize adapters that allow incompatible message streams to be connected (e.g.,

different time representations, or different accuracies or units of measurement). It can also decide when incompatibilities are too great to meet the purpose of integration.

The Frauenhofer Institute for Experimental Software Engineering in Kaiserslautern, Germany, has done much work on the safety assurance of OAS, mostly in the context of farm tractors and the very sophisticated implements that attach to them. Trapp and Schneider provide a comprehensive survey of safety assurance for OAS [7]. They frame their survey in the context provided by Models@Runtime (M@RT), which is an emerging framework for organizing OAS. The idea is that if open systems are to adapt to each other, they need to know something about each other, and one way to do this is to exchange models for their individual behavior and assumptions. It is a matter for negotiation what constitutes a "model" in this context. In DEOS, for example, it is the D-CASE representation of an assurance case, in SILF and ONISTT it is ontologies in some standardized description logic (none of DEOS, SILF and ONISTT describe themselves in M@RT terms, but they fit the paradigm).

Trapp and Schneider extend the M@RT idea to *Safety* Models@Runtime (SM@RT) for self-integrating systems. They distinguish four levels of sophistication and difficulty according to how ambitious is the integration, and note that only the first two are feasible at present. The simplest class, "Safety Certificates at Runtime," applies when it is sufficient for system safety that each component system maintains its own local safety objective. Then come "Safety Cases at Runtime," where component system safety cases guide adaptation and are integrated dynamically to deliver a safe and assured assembly (e.g., one system may need to demonstrate that it delivers properties assumed by another). Next is "V&V at Runtime," where it may be that one system cannot deliver the assumptions required by another, so more extensive adjustment are needed (e.g., wrapping or runtime monitoring to exclude some class of faults). Finally, "Hazard Analysis and Risk Assessment at Runtime" applies when essentially the full procedure of safety assurance (e.g., identification and elimination or mitigation of hazards, and assurance that this has been done, accurately and completely) is performed on integration. Personally, I am not convinced the "at Runtime" appellation is suitable for all these notions; certainly runtime monitoring is one way to ensure certain properties, but the generic process employed here is analysis, adaptation, and assurance at *integration-time*.

4 Prospects

The recent work outlined above provides some impressive accomplishments and attractive frameworks in which to view the issues of self-integration. What I wish to propose is that automated deduction provides the capabilities needed to realize the more challenging classes of trustworthy self-integration, and that such capabilities (essentially, theorem provers) will be the engines of integration.

From the scenarios in Section 2, we see that ontology matching is an important capability: all parties need to be talking about the same things (patient identity, blood pressure, oxygen levels, drug dosage) in the same way and using

the same units. As mentioned in Section 3, SILF proposes that mediation mechanisms are employed to ensure this and ONISTT constructs such mechanisms automatically using ontological specifications for the data streams. The purpose of the integration is similarly represented in a task ontology. These ontological specifications are given in a description logic (the Web Ontology Language, OWL) and construction of the mediators is accomplished by a dedicated program written in Prolog.

Going beyond this, we can imagine that systems document not only the ontologies for their data streams, but specifications of their properties (e.g., those they assume and those they guarantee). Mediators may then have to provide more complex services than ontology matching: for example, they may need to enforce a certain invariant. A "runtime monitor" provides a simple way to do this: it is separate function that observes state variables of the system and evaluates a predicate representing the desired invariant. Should the predicate ever evaluate to false, then the monitor signals an alarm (to be handled by another part of the system, or by a human operator) or halts the system. If the invariant predicate is available in a suitably formal form (perhaps in the local assurance case) then software for the monitor can be synthesized automatically. Runtime monitors are the core of *Runtime Verification* and associated methods of assurance [8–10], whose guarantees can be very strong [11].

The ideas of runtime mediation and monitoring can be extended from ontology matching and invariant checking to more complex functions such as masking some kinds of faults or providing supervisory control. The software to accomplish these functions is more complex than simply implementing a given predicate, but modern methods of synthesis can often automate its construction. These methods are based on automated deduction (i.e., theorem proving).

Checking the satisfiability of propositional (i.e., Boolean) formulas is the simplest problem in automated deduction, and is also the quintessential NP-Complete problem, meaning its worst-case computational complexity is probably exponential. Yet modern satisfiability (SAT) solvers are remarkably efficient on realistic problems, often solving examples with thousands of variables in fractions of a second. Their efficiency, and the generality of the SAT problem, are such that the best way to solve many search problems is first to transform them to a SAT instance, then solve that, and finally transform back to the original problem. Satisfiability Modulo Theories (SMT) extends SAT by adding support for useful theories such as real and integer arithmetic, uninterpreted functions, arrays, and several other datatypes used in software. SMT solvers are at the core of many modern tools for automated program analysis and verification, including static analyzers, model checkers, and test generators.

Efficient automated verification methods open the door to effective automated synthesis: crudely, this is done by enumerating candidate solutions, applying automated verification to each in turn, and selecting the first that succeeds. Of course, this crude approach must be refined to yield a practical synthesis procedure. The first required refinement is a way for a human user to suggest the

space of likely solutions. An attractive way to do this is for the user to specify a sketch or template and leave the synthesis search procedure to fill in the details.

The "glue" elements needed in self-integration are generally straightline programs and single loops, and the formulas that correspond to these (e.g., invariants). A trivial example of the template for an invariant is $Ax + By < C$ for some parameters A, B, and C. Formally, this can be expressed as

$$\exists A, B, C : \forall x, y : Ax + By < C \tag{1}$$

where x and y are program variables, and the parameters A, B, C must be instantiated by the synthesis procedure. Variants on this formulation can be used to express assumption synthesis (find the weakest environment in which a given component meets its requirements), supervisory controller synthesis (design an algorithm to selectively disable component actions so that it satisfies some goal in the face of uncontrollable actions by the environment), and full synthesis (design an algorithm to achieve some goal).

The second refinement to the crude synthesis procedure sketched above is an efficient way to search for suitable values for the parameters A, B, and C. Observe the Exists-Forall (EF) two-level quantification in the formulation (1) above. Standard SMT solvers solve single-level Exists and Forall problems, but recent developments extend this to EF solving using refinements of the search procedure sketched above [12]. An EF-SMT solver uses an ordinary SMT solver as a component and works by iteratively performing two steps.

1. Guessing (cleverly) instantiations for the Exists variables and querying the SMT solver with the resulting Forall formula. If this succeeds, we are done.
2. If it fails, use the result (i.e., counterexample) of the Forall query to help in finding the next instantiation of the Exists variables.

The key in making this iteration efficient is to use (i.e., learn from) the result of failed verification (Forall) steps to prune the search space for subsequent synthesis (Exists) steps.

There is a tradeoff between generality and performance in formal synthesis. EF-SMT handles richer logics than the description logics of ONISTT, but the specialized synthesis procedure of the latter outperforms EF-SMT within its domain. Nonetheless, we can expect continued progress in EF and standard SMT solving and this will eventually lead to superior synthesis performance.[2]

Both SMT and EF-SMT solvers use learning from failed candidate solutions to optimize their search. Self-integrating systems can also use explicit learning methods to cope with noisy sensors, erratic components, or external attack [13]. The idea is to learn a "safety envelope" for the system under controlled, quiescent, conditions and then monitor this in operation. The safety envelope is an invariant (which may consist of sub-invariants for each learned "mode" of the system) and its violation can indicate an attack, noise, or erratic (faulty) behavior. The idea of a learned safety envelope (i.e., a conservative model) is quite

[2] We should note that the ONISTT technology also evolves, and the current system uses the Flora-2 reasoning system for F-logic.

different than the predictive models (e.g., Kalman filters) popular in control theory and serves a different and novel purpose. Depending on context, response to violation of a safety envelope could be to raise an alarm or to replace "bad" values by recent "good" ones, or by defaults.

Given the ingredients described above, we can propose that as self-integrating systems come together they exchange ontologies, models, specifications of their assumptions and guarantees, and assurance arguments. All of these are potentially formal descriptions; although an assurance case is generally expected to be a persuasive rather than deductively valid argument, a case can be made that its uncertainties should be restricted to the interpretation of evidence, not the application of argument to its claims [14]. We can then further propose that the mechanisms of automated formal verification, synthesis, and learning should go to work with the aim of establishing and maintaining local and global safety properties and their attendant assurance arguments.

This raises the question what is the purpose of the integrated system and what is the safety claim of the integrated system. As a first step (corresponding to Trapp and Schneider's "Safety Certificates at Runtime" and "Safety Cases at Runtime") we might suppose that the goal is for each component system to maintain its local safety claim despite the stresses of interaction. For example, a farm tractor must continue to brake safely when an implement is attached, and to do this it needs to know the weight and center of gravity of the implement, and its braking performance. Construction of an assurance case for the integrated system will employ automated deduction and synthesis: deduction to detect potential violations of individual safety claims in the integrated system (e.g., due to a property mismatch in an assume-guarantee argument), and synthesis to construct monitoring and recovery procedures to overcome these.

For more advanced integration, I propose that the purpose and safety claim of the integrated system will be associated with an "integration app" that guides the assembly. This might approach Trapp and Schneider's "V&V at Runtime," where the component systems not only maintain safety in each other's presence, but also deliver some new, integrated (i.e., positively emergent) behavior. For example, a standard pulse oximeter and PCA will not do anything interesting when integrated unless a suitable "safe analgesia app" takes the initiative to close the loop. We can imagine that the design and purpose of the "safe analgesia app" are indicated as formal templates or sketches, whose details are filled in using EF-SMT synthesis that uses the capabilities and properties announced by the attached PO and PCA.

5 Conclusion

I have described a view of self-integrating systems that draws on many current ideas in the Internet of Things, Service Oriented Architecture, Open Adaptive Systems, and Models@Runtime. I then argued that in many applications, these self-integrating systems need to be trustworthy and I outlined some examples in medical devices.

I hope then to have made the case that trustworthy self-integration requires automated verification and automated synthesis of monitors, adapters, and mediators at integration-time. I sketched how modern technology for automated deduction, specifically SMT and EF-SMT solvers, can perform these tasks.

Pulling these various threads together, I propose that a modest but useful class of trustworthy self-integrating systems is within reach, and that embedded automated deduction is the enabling engine for this development.

Looking further into the future, we can speculate on self-integration of systems that are individually highly adaptive or intelligent. I believe this will be a challenging endeavor: almost all crashes of modern aircraft are due to flawed interaction between automated aircraft systems and the intelligent human crew. Although usually ascribed to human error, a more nuanced assessment sees flaws in the manner of interaction, with each party having insufficient insight into the other's state and intentions. Thus, it may be that the information that must be exchanged between such advanced systems is of a more strategic character than the tactical information discussed here. For the most advanced kinds of systems, it may be that what is needed is agreement on a shared system of ethics.

Acknowledgments. This work was partially funded by SRI International. Many of the ideas assembled here originated in discussions with Dave Hanz of SRI, who also provided helpful comments on the paper.

References

1. Whitehead, S.F., Goldman, J.M.: Getting connected for patient safety: How medical device "plug-and-play" interoperability can make a difference. Patient Safety and Quality Healthcare (2008) Available at http://www.psqh.com/janfeb08/connected.html. 3
2. Rushby, J.: The interpretation and evaluation of assurance cases. Technical Report SRI-CSL-15-01, Computer Science Laboratory, SRI International, Menlo Park, CA (2015). 5
3. Tokoro, M.: Open Systems Dependability—Dependability Engineering for Ever-Changing Systems. CRC Press (2013) 5
4. Kuramtisu, K.: D-Script: Dependable scripting with DEOS process. In: 3rd International Workshop on Open Systems Dependability (WOSD), Pasadena, CA (2013) 326–330 Workshop held in association with ISSRE'13. 5
5. NATO Science and Technology Organization, Neuilly-Sur-Seine, France: Framework for Semantic Interoperability. (2014) STO Technical Report TR-IST-094. 5
6. Ford, R., Hanz, D., Elenius, D., Johnson, M.: Purpose-aware interoperability: The ONISTT ontologies and analyzer. In: Simulation Interoperability Workshop. Number 07F-SIW-088, Simulation Interoperability Standards Organization (2007) 5
7. Trapp, M., Schneider, D.: Safety assurance of open adaptive systems—a survey. In Bencomo, N., France, R., Cheng, B.H., Assmann, U., eds.: Models@Run.Time: Foundations, Applications, and Roadmaps. Volume 8378 of Lecture Notes in Computer Science. Springer-Verlag (2014) 279–318 6
8. Rushby, J.: Kernels for safety? In Anderson, T., ed.: Safe and Secure Computing Systems. Blackwell Scientific Publications (1989) 210–220 (Proceedings of a Symposium held in Glasgow, October 1986). 7

9. Rushby, J.: Runtime certification. In Leucker, M., ed.: Eighth Workshop on Runtime Verification: RV08. Volume 5289 of Lecture Notes in Computer Science., Budapest, Hungary, Springer-Verlag (2008) 21–35 7
10. Rushby, J.: The versatile synchronous observer. In Iida, S., Meseguer, J., Ogata, K., eds.: Specification, Algebra, and Software, A Festschrift Symposium in Honor of Kokichi Futatsugi. Volume 8373 of Lecture Notes in Computer Science., Kanazawa, Japan, Springer-Verlag (2014) 110–128 7
11. Littlewood, B., Rushby, J.: Reasoning about the reliability of diverse two-channel systems in which one channel is "possibly perfect". IEEE Transactions on Software Engineering **38** (2012) 1178–1194 7
12. Dutertre, B.: Solving Exists/Forall problems with Yices. In: SMT Workshop 2015 (held in association with CAV), San Francisco, CA (2015) 8
13. Tiwari, A., Dutertre, B., Jovanović, D., de Candia, T., Lincoln, P.D., Rushby, J., Sadigh, D., Seshia, S.: Safety envelope for security. In: Proceedings of the 3rd International Conference on High Confidence Networked Systems (HiCoNS), Berlin, Germany, ACM (2014) 85–94 8
14. Rushby, J.: On the interpretation of assurance case arguments. In: 2nd International Workshop on Argument for Agreement and Assurance (AAA 2015), Kanagawa, Japan (2015) 9