Combining MILS with Contract-Based Design for Safety and Security Requirements

Alessandro Cimatti[1], Rance DeLong[2], Davide Marcantonio[1], and Stefano Tonetta[1]

[1] FBK-irst {cimatti,marcantonio,tonettas}@fbk.eu
[2] The Open Group r.delong@opengroup.org

Abstract. The distributed MILS (D-MILS) approach to high-assurance systems is based on an architecture-driven end-to-end methodology that encompasses techniques and tools for modeling the system architecture, contract-based analysis of the architecture, automatic configuration of the platform, and assurance case generation from patterns. Following the MILS[3] paradigm, the architecture is pivotal to define the security policy that is to be enforced by the platform, and to design safety mechanisms such as redundancies or failures monitoring. In D-MILS we enriched these security guarantees with formal reasoning to show that the global system requirements are met provided local policies are guaranteed by application components. We consider both safety-related and security-related requirements and we analyze the decomposition also taking into account the possibility of component failures. In this paper, we give an overview of our approach and we exemplify the architecture-driven paradigm for design and verification with an example of a fail-secure design pattern.

Keywords: MILS, contract-based design, safety and security, formal verification

1 Introduction

The *MILS architectural approach* [6] to the design and implementation of critical systems involves two principal phases: the development of an abstract architecture intended to achieve the stated purpose, and the implementation of that architecture on a robust technology platform. During the first phase, essential properties are identified that the system is expected to exhibit, and the contributions to the achievement of those properties by the architectural structure and by the behavioural attributes of key components are analyzed and justified.

Safety and security are more and more intertwined problems. The potential impact of security threats on safety-critical systems is increasing due to the interconnections of systems. Safety, security, and dependability are emergent behavioural properties of a system interacting with its environment. The MILS

[3] "MILS" was originally an acronym for "Multiple Independent Levels of Security". Today, we use "MILS" as a proper name for an architectural approach and an implementation framework, promulgated by a community of interested parties, and elaborated by ongoing MILS research and development efforts.

approach leverages system architecture to support vital system-level properties. The architecture reflects an intended pattern of information flow and causality referred to as the *policy architecture*, while key components of the architecture enforce *local policies* through specific behavioural properties. By reasoning compositionally over the components about the policy architecture and the local policies, many useful system-level properties may be established.

The *MILS platform* provides the technology for the concrete realisation of an abstract system architecture. A *separation kernel* [33,31], the underlying foundational component of the MILS platform, is used to establish and enforce the system architecture according to its configuration data.

The assurance of a system's properties depends not only on the analysis of its design but on the correct implementation and deployment of that design. The configuration of the separation kernel must faithfully implement the specified architecture. This is guaranteed by the *MILS platform configuration compiler* that is driven by a model of the architecture and the constraints of the target platform to synthesize viable and semantically correct configuration data corresponding to the specified architecture.

In this paper, we give an overview of the integration of the MILS approach with contract-based reasoning developed in the D-MILS project [1]. The approach relies on the OCRA tool [13] to formally prove that the global system requirements are met, provided local policies are guaranteed by application components. We consider both safety-related and security-related requirements and we analyze the decomposition also taking into account the possibility of component failures. We exemplify the architecture-driven approach on the Starlight Interactive Link example [5], extended with a safety mechanism in order to take into account the possibility of component failures.

The rest of the paper is organized as follows: in Section 2, we give an overview of D-MILS project; in Section 3, we detail how the MILS approach has been extended with a contract-based design of the architecture and the related tool support; in Section 4, we describe how we extended the Starlight example and the related analysis of contract refinement; in Section 5, we give an overview of the related work, while we conclude in Section 6.

2 Overview of D-MILS

The D-MILS concept extends the capacity of MILS to implement a single unified policy architecture to a network of separation kernels [29,28]. To accomplish this, each separation kernel is combined with a new MILS foundational component, the *MILS networking system* (MNS), producing the effect of a distributed separation kernel. In the D-MILS Project [1] we have employed *Time-Triggered Ethernet* (TTE) [32] as the MILS "backplane", permitting us to extend the robustness and determinism benefits of a single *MILS node* to the network of *D-MILS nodes*, referred to as the *distributed MILS platform*[4] [26,27].

[4] Our D-MILS Platform is composed of the LynxSecure Separation Kernel from Lynx Software Technologies, France, and TTE from TTTech, Austria.

Since D-MILS systems are intended for critical applications, assurance of the system's critical properties is a necessary byproduct of its development. In such applications, evidence supporting the claimed properties must often be presented for consideration by objective third-party system certifiers. To achieve assurance requires diligence at all phases of design, development, and deployment; and, at all levels of abstraction: from the abstract architecture to the details of configuration and scheduling of physical resources within each separation kernel and within the TTE interfaces and switches. Correct operation of the deployed system depends upon the correctness of the configuration details, of the component composition, of key system components[5], and of the D-MILS platform itself. Configuration is particularly challenging, because the scalability that D-MILS is intended to provide causes the magnitude of the configuration problem to scale as well. The concrete configuration data and scheduling details of the numerous separation kernels and of the TTE are at a very fine level of granularity, and must be complete, correct, and coherent.

The only reasonable prospect of achieving these various aspects of correctness, separately and jointly, is through pervasive and coordinated automation as embodied in the D-MILS tool chain. Inputs to the tool chain include, a declarative model of the system expressed in our own MILS dialect of the Architecture Analysis and Design Language (AADL) [18], facts about the target hardware platform, properties of separately developed system components, designer-imposed constraints and system property specifications, and human guidance to the construction of the assurance case. Components of the tool chain perform parsing of the languages [19], transformations among the various internal forms [20,21], analysis and verification [24], configuration data synthesis and rendering [25], and pattern-based assurance case construction [22,23]. Outputs of the tool chain include, proofs of specified system properties, configuration data for the D-MILS platform, and an assurance case expressed in Goal Structuring Notation (GSN) [2]. We say that D-MILS provides not only a robust and predictable platform for system implementation, but also an end-to-end and top-to-bottom method supported by extensive automation.

3 Architecture-driven integration of the MILS approach and contract-based design

In this paper we focus on the integration of the MILS architectural approach with contract-based design and analysis. Both MILS and contract-based approaches focus on architecture, and do so in a complementary way. MILS regards information flow policy as an abstraction of architecture, and seeks to maximize the correspondence between architectural structure and the desired information flow policy of a system, which may rely on the behavior of key components to enforce local policies that further restrict the maximal information flow permitted by the

[5] The D-MILS Project regards proof of component correctness to a specification as a "solved problem" and focusses on the correctness of the composition of components' specifications, and of the configuration of the D-MILS platform.

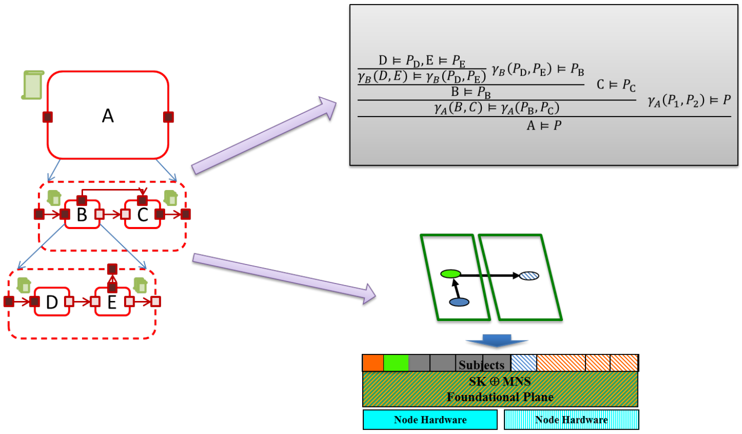

Fig. 1. The architecture is used for 1) formal reasoning to prove that the system requirements are assured by the local policies, 2) configuration of the platform to ensure the global information flow policy and the integrity of the architecture.

architecture. The contract-based approach employs formalization and a method to prove that the architecture decomposition represented in the set of contracts of the components is a proper refinement of the system requirements. Contracts specify the properties that the components individually, and the system as a whole, are expected to guarantee, and the assumptions that their respective environments must meet. Formal verification techniques are used to check that the derivation of the local policies from the system requirements is correct.

An architecture is only as valuable as the integrity of its components and connections. Recognizing the importance of integrity, MILS provides an implementation platform that can be configured to the "shape" of the architecture by initializing it with specific configuration data compiled to embody the global information flow policy.

The two methods are complementary and their combination yields a strong result. The contract-based method proves that the composition of components that satisfy their contracts will meet the system requirements, provided that their integrity is protected. The MILS platform guarantees the integrity of components and their configured connections, preventing interference that could cause a verified component to fail to satisfy its contract.[6]

In Fig. 1, we show the approach applied to an abstract example. The system A is decomposed into subsystems B and C, and B in turn is decomposed into D and E. Each component is enriched with a contract (represented here by green scrolls). If the contract refinement is correct, we have associated with the architecture a formal proof that the system is correct provided that the leaf

[6] For the purpose of our work we assume that components can be constructed and verified to satisfy their contracts.

components (D, E, and C) satisfy their contracts. Namely, if D and E satisfy their contracts ($D \models P_D, E \models P_E$) and the contract refinement of B is correct ($\gamma_B(P_D, P_E) \preceq P_B$), then the composition of D and E satisfies the contract of B ($\gamma_B(D, E) \models P_B$). Moreover, if C satisfies its contract ($C \models P_C$) and the contract refinement of A is correct ($\gamma_A(P_B, P_C) \preceq P_A$), then the composition of B and C satisfies the contract of A ($\gamma_A(B, C) \models P_A$).

In MILS terms, the architecture defines three subjects (D, E and C) and prescribes that the only allowed communications must be the ones between D and E and between E and C. This is translated into a configuration for the D-MILS platform (taking into account other deployment constraints in terms of available resources), which in this example encompasses two MILS nodes.

3.1 Tool support for contract-based reasoning

In D-MILS, the architecture is specified in a variant of AADL, called MILS-AADL, similar to the SLIM language developed in the COMPASS project [7]. The COMPASS tool set has been extended in order to support the new language and to enrich the components with annotations that specify different verification properties such as contracts. The language used to specify the component contracts is the one provided by the OCRA tool [13]. It consists of a textual human-readable version of a First-Order Linear-time Temporal Logic. The logic has been extended in D-MILS to support uninterpreted functions, i.e. functional symbols that do not have a specific interpretation but are used to abstract procedures and the related results (such as CRC checksum or encryption), or to label data with user-defined tags (such as "is_high" or "low-level", etc.).

Such a very expressive language required the development of effective techniques to reason about the contracts. To this purpose the engine undertakes to prove the contract refinement. The refinement is first translated by OCRA into a set of entailment problems in temporal logic. nuXmv [11] translates this into a liveness model-checking problem with a classic automata-theoretic approach [37]. The resulting problem requires proving that a certain liveness condition can be visited only finitely many times along an (infinite) execution. This problem is in turn reduced to proving an invariant on the reachable states with the K-liveness techniques described in [17]. This has been extended to infinite-state systems and to take into account real-time aspects in [15]. Finally, the invariant is proved with an efficient combination of induction-based reasoning, explicit-state search, and predicate abstraction, extending the IC3 algorithm [9] to the infinite-state case, as described in [14].

4 Starlight example

4.1 Architecture

In this section, we exemplify the approach on an example taken from the literature [5,12]. The Starlight Interactive Link is a dispatching device developed

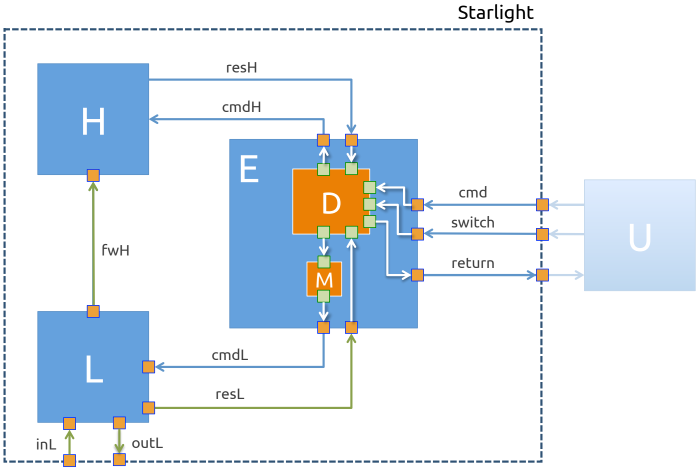

Fig. 2. Architecture of the D-MILS Starlight example.

by the Australian Defense Science and Technology Organization to allow users to establish simultaneous connections to high-level (classified) and low-level networks. The idea is that the device acts as a switch that the user can control to dispatch the keyboard output to either a high-level server or to a low-level server. The user can use the low-level server to browse the external world, send messages, or have data sent to the high-level server for later use.

Fig. 2 shows the architecture of the Starlight Interactive Link as formalized in D-MILS. The components H and L represent respectively the high-level and low-level networks. The low-level network can exchange information with the external world. The component D represents the Starlight device, which receives commands from the user and dispatches the commands to H or to L based on an internal state. The state is changed with two *switch* commands, namely *switch_to_low* and *switch_to_high*. The original architecture has only the blue components, with D in place of E. We extended this architecture with a safety mechanism to make the system "fail-secure" with respect to failures of the dispatcher: the dispatcher is extended with a monitor M; the communication of the dispatcher to L is filtered by M that in case of failure of D blocks the communication. To avoid confusion we refer to the actual device that is filtered by M as the dispatcher (D), while to the component consisting of D and M as the extended dispatcher E.

4.2 System contract

The architecture has been enriched with contracts that formalize the functional requirements to ensure that the system responds correctly to the user commands, and the security requirement that there is no leakage of high-level data. Here, we focus on the latter, which says:

Req-Sys-secure: No high-level data shall be sent by L to the external world.

The architecture ensures Req-Sys-secure assuming the following requirement on the user:

Req-User-secure: The user shall switch the dispatcher to high before entering high-level data.

Moreover, we consider the following safety requirement:

Req-Sys-safe: No single failure shall cause a loss of Req-Sys-secure.

We formalized the requirements of the system and of the components using OCRA contracts. In the following, we use the concrete syntax accepted by the tool. We briefly clarify the used notation: "and", "or", "not", "implies" are standard Boolean operators; "always", "never", "since", "in the past" are standard temporal operators of LTL with past also referred to with the mathematical notation G, $G\neg$, S, O; "last_data" is a built function to refer to the last data passed by the event of a event data port; italics names refer to ports or uninterpreted functions declared in the model.

The requirements Req-Sys-secure and Req-User-secure have been formalized into the FO-LTL formulas:

Formal-Sys-secure: never *is_high*(last_data(*outL*))

Formal-User-secure: always ((*is_high*(last_data(*cmd*))) implies ((not *switch_to_low*) since *switch_to_high*))

Note that the formalization of Req-User-secure improves the informal requirement, which is not precise. A literal formalization would be:

Formal-User-secure-wrong: always ((*is_secure*(last_data(*cmd*))) implies (in the past *switch_to_high*))

but this is wrong, because we have to ensure that the last switch was a *switch_to_high*, without a more recent *switch_to_low*[7]. We can actually improve the informal requirement as:

Req-User-secure-new: Whenever the user sends commands with high data, she shall previously issue a *switch_to_high* and no *switch_to_low* since the last *switch_to_high*.

which is formalized by Formal-User-secure.

Note that while Req-Sys-secure is a requirement on the implementation of the Starlight system, Req-User-secure is actually a requirement on its environment (the user). This is reflected by the system contract specification, which sets Formal-Sys-secure as the guarantee and Formal-User-secure as the assumption of the system contract.

[7] As suggested by one of the reviewers, in an alternative model, we could use only one event data instead of two switch events and ensure that the last switch was low.

4.3 Component contracts

The dispatcher ensures the system security requirement with the following local requirement:

Req-D-low-mode: The dispatcher shall send commands to L only if the last switch was a *switch_to_low* and the input command has been received after.

formalized into:

Formal-D-low-mode: always (*cmdL* implies (((not *switch_to_high*) since *switch_to_low*) and ((not *switch_to_low*) since *cmd*)))

In order to fulfill requirement Req-Sys-safe, we also filter the commands to L by a monitor M, which has a requirement **Req-M-low-mode** identical to Req-D-low-mode, and formalized in the same way. Thus, D passes also the switches to the monitor and must ensure the following requirement:

Req-D-fw-switch: Whenever the dispatcher receives a *switch_to_high*, it shall pass it to M before doing any other actions and it sends a *switch_to_low* to M only if the last received switch was a *switch_to_low*.

formalized into:

Formal-D-fw-switch: always ((*switch_to_high* implies ((not (*cmdH* or *cmdL* or *return* or *monitor_switch_to_low*)) until *monitor_switch_to_high*)) and (*monitor_switch_to_low* implies ((not *switch_to_high*) since *switch_to_low*)));

Finally, in order to make the refinement correct, we must require all components to not invent high data. We express this by requiring that D, M, and L only pass the data that they have received. Thus, for D, we require that:

Req-D-data: D shall pass to *cmdL* only the data that has been received with last *cmd*.

formalized into:

Formal-D-data: always ((*cmdL* implies ((in the past *cmd*) and (last_data(*cmdL*) = last_data(*cmd*)))))

The requirements **Req-M-data** and **Req-L-data**, of M and L respectively, are analogous. Note that these formulas are actually guarantees of corresponding contracts, without assumptions (i.e. assumptions equal to *true*).

4.4 Analysis results

Given the above contract specifications, OCRA can prove the system Req-Sys-secure assuming Req-User-secure is correctly refined by the contracts of D, M, and L (see [16] for more details on the technique). One can also show that by using Formal-User-secure-wrong instead of Formal-User-secure the refinement is not correct and yields a counterexample trace execution.

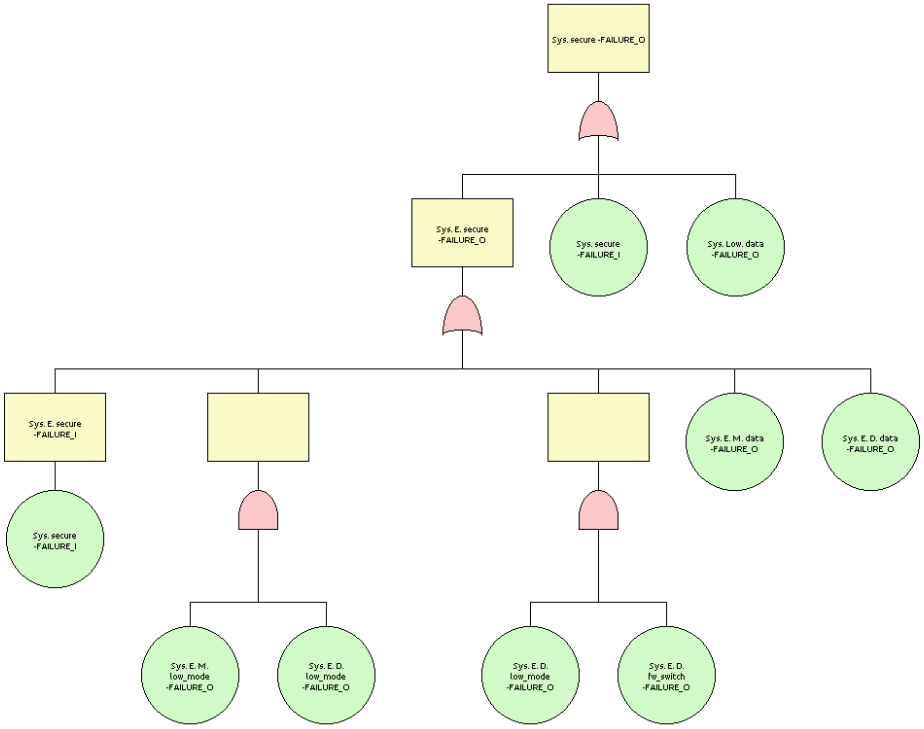
Sys. secure -FAILURE_O
Sys. E. secure -FAILURE_O
Sys. secure -FAILURE_I
Sys. Low. data -FAILURE_O
Sys. E. secure -FAILURE_I
Sys. E. M. data -FAILURE_O
Sys. E. D. data -FAILURE_O
Sys. secure -FAILURE_I
Sys. E. M. low_mode -FAILURE_O
Sys. E. D. low_mode -FAILURE_O
Sys. E. D. low_mode -FAILURE_O
Sys. E. D. fw_switch -FAILURE_O

Fig. 3. Fault-tree generated from the contract refinement. Events are labeled with the name of the component instance followed by the name of the contract, followed either by FAILURE_O, which represents the failure of the component to satisfy the contract, or by FAILURE_I, which represents the failure of the component environment to satisfy the contract assumption.

In order to prove Req-Sys-safe, we use OCRA to produce a fault tree showing the dependency of the system failure on the failure of the components (see [8] for more details on the technique). The generated fault tree is exhibited in Fig. 3. It shows that neither Req-D-low-mode nor Req-M-low-mode are single points of failure. Instead, Req-D-data, Req-M-data, Req-L-data are single points of failure. While the failure of Req-L-data does not represent real threats since L never receives high data, the failure of Req-D-data and Req-M-data could result in D or M sending information that had been temporary stored in a buffer used for handling multiple requests or in a cache for improving performance. This can be solved for example by ensuring that such memories are deleted before every *switch_to_low* is completed.

5 Related work

Security-by-contract is an approach proposed in [30] to increase the trust in code downloaded on mobile applications. The work proposes a framework where downloaded code can be checked according to a security contract. With respect to this work, there is no focus on the system architecture, the refinement of contracts, or safety analysis taking into account component failures.

Information flow contracts are supported in SPARK, a safety-critical subset of Ada [4,3]. The SPARK contracts are specified at software level on procedures. So, in principle, they are complementary to our approach, which focuses more on the system-level architecture. As for future work, we will consider to extend the approach with information flow contracts. Currently, the information flow can be specified only at coarse level with the connections in the architecture. To our knowledge, there are no works combining SPARK information flow contracts with safety analysis.

In [10], an avionic case-study architecture is formalized in Alloy and analyzed with respect to safety and security requirements. Similarly to our approach, first-order logic is used to formalize the requirements, although Alloy does not support temporal operators. The case study formalizes also security attacks that are not present in our example. Different to our approach, the failures and security attacks are explicitly modeled, while in our case we exploit a feature of OCRA to automatically inject the failures starting from the nominal contract specification. Our conjecture is that the same case study of [10] can be formalized in MILS-AADL or directly in OCRA with the possibility of checking contract refinement and performing contract-based fault-tree analysis.

Another case study on validation of safety and security requirements has been presented in [35], but it focuses on testing.

Fault trees and FMEA have been extended in [36,34] to consider also security aspects. Different to our approach and other model-based safety analysis techniques, these works are not based on the automatic generation of fault trees and FMEA tables from the system design.

6 Conclusions

In this paper, we briefly overview the approach to safety and security undertaken in D-MILS and we describe a small example of the D-MILS approach to the verification of the system architecture with respect to safety and security requirements. The example is based on the Starlight device that switches commands between high-level and low-level servers. The requirements of the system and its components have been formalized using OCRA contracts, their refinement has been verified and analyzed taking into account the failure of components. This is a preliminary application of the methodology, which will be further evaluated in the D-MILS project demonstrators. In the future, we would like to integrate contracts and their analysis with finer-grained information flow properties as do the SPARK contracts discussed in [4,3].

Acknowledgments This work was performed on the D-MILS project ("Distributed MILS for Dependable Information and Communication Infrastructures", European Commission FP7 ICT grant no. 318772), with our partners fortiss, Verimag, RWTH Aachen, U of York, Frequentis, Lynx, TTTech, and INRIA, funded partially under the EC's Seventh Framework Programme.

References

1. *D-MILS Project*. `http://http://www.d-mils.org/`.
2. *GSN community standard*, tech. rep., Origin Consulting (York) Limited, 2011.
3. T. Amtoft, J. Hatcliff, and E. Rodríguez, *Precise and Automated Contract-Based Reasoning for Verification and Certification of Information Flow Properties of Programs with Arrays*, in ESOP, 2010, pp. 43–63.
4. T. Amtoft, J. Hatcliff, E. Rodríguez, Robby, J. Hoag, and D. Greve, *Specification and Checking of Software Contracts for Conditional Information Flow*, in FM, 2008, pp. 229–245.
5. M. Anderson, C. North, J. Griffin, R. Milner, J. Yesberg, and K. Yiu, *Starlight: Interactive Link*, in 12th Annual Computer Security Applications Conference, 1996, pp. 55–63.
6. C. Boettcher, R. DeLong, J. Rushby, and W. Sifre, *The MILS Component Integration Approach to Secure Information Sharing*, in 27thAIAA/IEEE Digital Avionics Systems Conference, St. Paul, MN, Oct. 2008.
7. M. Bozzano, A. Cimatti, J. Katoen, V. Y. Nguyen, T. Noll, and M. Roveri, *Safety, Dependability and Performance Analysis of Extended AADL Models*, Comput. J., 54 (2011), pp. 754–775.
8. M. Bozzano, A. Cimatti, C. Mattarei, and S. Tonetta, *Formal Safety Assessment via Contract-Based Design*, in ATVA, 2014, pp. 81–97.
9. A. R. Bradley, *SAT-Based Model Checking without Unrolling*, in VMCAI, 2011, pp. 70–87.
10. J. Brunel, L. Rioux, S. Paul, A. Faucogney, and F. Vallée, *Formal Safety and Security Assessment of an Avionic Architecture with Alloy*, in ESSS, 2014, pp. 8–19.
11. R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, and S. Tonetta, *The nuXmv Symbolic Model Checker*, in CAV, 2014, pp. 334–342.
12. S. Chong and R. Van Der Meyden, *Using architecture to reason about information security*, arXiv preprint arXiv:1409.0309, (2014).
13. A. Cimatti, M. Dorigatti, and S. Tonetta, *OCRA: A tool for checking the refinement of temporal contracts*, in ASE, 2013, pp. 702–705.
14. A. Cimatti, A. Griggio, S. Mover, and S. Tonetta, *IC3 Modulo Theories via Implicit Predicate Abstraction*, in TACAS, 2014, pp. 46–61.
15. ———, *Verifying LTL Properties of Hybrid Systems with K-Liveness*, in CAV, 2014, pp. 424–440.
16. A. Cimatti and S. Tonetta, *Contracts-refinement proof system for component-based embedded systems*, Sci. Comput. Program., 97 (2015), pp. 333–348.
17. K. Claessen and N. Sörensson, *A liveness checking algorithm that counts*, in FMCAD, 2012, pp. 52–59.
18. *Specification of MILS-AADL*, Tech. Rep. D2.1, Version 2.0, D-MILS Project, July 2014. `http://www.d-mils.org/page/results`.
19. *D2.2 translation of mils-aadl into formal architectural modeling framework*, Tech. Rep. D2.2, Version 1.2, D-MILS Project, Feb. 2014. `http://www.d-mils.org/page/results`.

20. *Intermediate languages and semantics transformations for distributed mils – part 1*, Tech. Rep. D3.2, Version 1.2, D-MILS Project, Feb. 2014. `http://www.d-mils.org/page/results`.
21. *Intermediate languages and semantics transformations for distributed mils – part 2*, Tech. Rep. D3.3, Version 1.0, D-MILS Project, July 2014. `http://www.d-mils.org/page/results`.
22. *Compositional assurance cases and arguments for distributed mils*, Tech. Rep. D4.2, Version 1.0, D-MILS Project, Apr. 2014. `http://www.d-mils.org/page/results`.
23. *Integration of formal evidence and expression in mils assurance case*, Tech. Rep. D4.3, Version 0.7, D-MILS Project, Mar. 2015. `http://www.d-mils.org/page/results`.
24. *Compositional verification techniques and tools for distributed mils—part 1*, Tech. Rep. D4.4, Version 1.0, D-MILS Project, July 2014. `http://www.d-mils.org/page/results`.
25. *Distributed mils platform configuration compiler*, Tech. Rep. D5.2, Version 0.2, D-MILS Project, Mar. 2014. `http://www.d-mils.org/page/results`.
26. *Extended separation kernel capable of global exported resource addressing*, Tech. Rep. D6.1, Version 2.0, D-MILS Project, Mar. 2014. `http://www.d-mils.org/page/results`.
27. *Mils network system supporting TTEthernet*, Tech. Rep. D6.3, Version 1.0, D-MILS Project, Mar. 2014. `http://www.d-mils.org/page/results`.
28. R. DeLong, *Commentary on the MILS Network Subsystem Protection Profile*, tech. rep., Sept. 2011. Version 0.31.
29. R. DeLong and J. Rushby, *Protection Profile for MILS Network Subsystems in Environments Requiring High Robustness*, Sept. 2011. Version 0.31.
30. N. Dragoni, F. Massacci, T. Walter, and C. Schaefer, *What the heck is this application doing? - A security-by-contract architecture for pervasive services*, Computers & Security, 28 (2009), pp. 566–577.
31. Information Assurance Directorate, National Security Agency, *U.S. Government Protection Profile for Separation Kernels in Environments Requiring High Robustness*, Fort George G. Meade, MD 20755-6000, June 2007. Version 1.03.
32. H. Kopetz, A. Ademaj, P. Grillinger, and K. Steinhammer, *The Time-Triggered Ethernet (TTE) Design*, 8th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC), Seattle, Washington, (2005).
33. J. Rushby, *The Design and Verification of Secure Systems*, in EighthACM Symposium on Operating System Principles, Asilomar, CA, Dec. 1981, pp. 12–21. (ACM *Operating Systems Review*, Vol. 15, No. 5).
34. C. Schmittner, T. Gruber, P. P. Puschner, and E. Schoitsch, *Security Application of Failure Mode and Effect Analysis (FMEA)*, in SAFECOMP, 2014, pp. 310–325.
35. M. Sojka, M. Krec, and Z. Hanzálek, *Case study on combined validation of safety & security requirements*, in SIES, 2014, pp. 244–251.
36. M. Steiner and P. Liggesmeyer, *Combination of Safety and Security Analysis - Finding Security Problems That Threaten The Safety of a System*, in SAFECOMP Workshop DECS, 2013.
37. M. Y. Vardi, *An Automata-Theoretic Approach to Linear Temporal Logic*, in Banff Higher Order Workshop, 1995, pp. 238–266.