Towards a Correct-by-Construction Design of Integrated Modular Avionics

Baoluo Meng, Joyanta Debnath, Sarat Chandra Varanasi,
Emmanuel Manoloios, Michael Durling, Saswata Paul
General Electric Research
Niskayuna, NY 12309. USA
Email: {baoluo.meng, joyanta.debnath, saratchandra.varanasi, emmanuel.manolios, durling, saswata.paul}@ge.com

Daniel Prince, Saif Alsabbagh, Richard Haadsma, Craig McMillan
GE Aviation Systems
Grand Rapids, MI 49512, USA
Email: {daniel.prince, saif.alsabbagh, richard.haadsma, craig.mcmillan}@ge.com

Chi Zhang, Tim Oates
University of Maryland, Baltimore County
Baltimore, MD 21250, USA
Email: {chzhang1, oates}@umbc.edu

***Abstract*—This paper presents a formal language and framework, OYSTER, to develop correct-by-construction design of Integrated Modular Avionics (IMA). The OYSTER language is created as an annex to the Architecture Analysis and Design Language (AADL) for encoding constraints for aspects of IMA. The OYSTER constraints involve determining the correct locations of hosted applications within an IMA system, validity of the port connections involved in the design, and the conformance of virtual links allocated with bandwidth and jitter requirements. OYSTER also allows synthesis of communication paths for the allocated virtual links. The OYSTER prototype tool is developed as a plugin to the Open Source AADL Tool Environment (OSATE), and invokes Satisfiability Modulo Theories (SMT) solvers to synthesize correct-by-construction architecture designs for IMA. In addition, behaviors of applications running on IMA components and their safety properties can be modeled in the Assume Guarantee REasoning Environment (AGREE) annex and checked by the Kind2 model checker. The verification results are guaranteed to be correct by the independently verifiable proof certificates produced by Kind2. Finally, the paper evaluates OYSTER on a GE Aviation use case – a fuel control system IMA, and discusses the lessons learned.**

I. Introduction

Integrated Modular Avionics (IMA) [36] are hybrid platforms that provide computing, communication, and I/O services for modern military and commercial aircraft. The implemented real-time embedded systems are architected and overlaid on the partitioned platform resources to form a highly-integrated system with full isolation and independence of each individual system. The platform elements are architected to maintain a high-integrity, fault-tolerant environment necessary for hosting critical system functionality. IMAs are able to simultaneously support critical and non-critical applications (at both high and low integrity levels) due to partitioned boundary layers between the applications.

Since the failure of IMA systems can have catastrophic consequences, the development of IMA platforms for modern commercial and military aircraft involves rigorous processes and tools along with tedious manual work to ensure that no errors are introduced. Therefore, IMA solutions are oftentimes produced by commercially available and/or internally developed proprietary tools, many of which have been certified for use by regulatory authorities such as the Federal Aviation Administration (FAA) and European Aviation Safety Agency (EASA). However, this makes IMA architecture solutions expensive to implement and changes to an IMA design (e.g., requirements changes) can be labor-intensive.

During a typical IMA development cycle today, a systems integrator will spend thousands of person-hours collecting, integrating, and fine-tuning an overall IMA system for qualification and fielding on real aircraft. One of the main contributors to the cost of integration of these network-based systems is the management of their changes and impacts of the changes on the rest of the IMA. Doing so requires systems expertise, knowledge about the implementation details of the IMA construction, and operational details of the qualified verification and configuration tools that must be used. The frequency of changes during IMA development can be very high and can even get to a point where it may become impractical/unsustainable for a human integrator to be able to absorb and understand all of the changes and their potential impacts. Additionally, the financial and scheduling constraints usually do not allow for a full "stop work" to assess the changes every time. Therefore, the access to tools that can aid in decision-making and can recommend appropriate changes for converging on a qualifiable and fieldable solution is crucial when dealing with systems of this scale and complexity.

Formal methods are mathematically-rigorous means for the specification, development, and verification of software and hardware systems, that can be harnessed for ensuring error-free system integration. Techniques such as model checking [17] and Satisfiability Modulo Theories (SMT) [9] can be used for automatically detecting if a given architecture violates a given property and for synthesizing potential changes in an architecture that might be needed in order to satisfy a given property. Therefore, such formal methods techniques

https://doi.org/10.34727/2023/isbn.978-3-85448-060-0_30

are well-suited for the development of IMA solutions. In this paper, we introduce the OYSTER framework, which uses model checking and SMT-based techniques for the synthesis of *correct-by-construction* IMA architectures. We have evaluated the feasibility of using OYSTER on real-life industrial applications by applying it on an IMA use case provided by our industry partners at GE Aviation.

This paper makes the following contributions to aid in the development of IMA solutions:

- Development of a formal language namely, OYSTER, to encode IMA architecture constraints and a translation scheme to SMT.
- An end-to-end tool prototype to synthesize a correct-by-construction IMA architecture solution using SMT solver and model checkers.
- A framework prototype for generating *independently verifiable* solutions for behavior models towards certification.
- Evaluation of the OYSTER framework on a fuel control system IMA use case provided by GE Aviation to show its practical feasibility.

The rest of the paper is structured as follows: Section II provides an overview of the overall architecture of the OYSTER framework; Section III details the OYSTER language for encoding IMA requirements; Section IV describes the architecture synthesis and proof generation capabilities of OYSTER; Section V presents an application of OYSTER on our use case followed by a discussion about lessons learned during the process; Section VI summarizes related work on system architectures synthesis; and finally, Section VII concludes the paper with a discussion on possible directions of future work.

II. OYSTER Overall Architecture

The OYSTER framework is depicted in Figure 1. The goal of this framework is to leverage formal methods tools to auto-synthesize a correct-by-construction IMA platform architecture, and prove formal properties about IMA behaviors using model checkers. The framework starts with modeling IMA software and hardware components in Architecture Analysis and Design Language (AADL) [23] that will be used to construct an IMA platform (corresponding to ①). The IMA architectural requirements are collected and encoded in OYSTER annex (②). The two pieces of information are translated to input to Satisfiability Modulo Theories (SMT) solvers for architecture synthesis (③). A satisfiable solution from the solver is converted back to an instantiated AADL model (④) that meets all the constraints specified in OYSTER. In addition, the model behavior and safety properties (⑤) can be manually added to the synthesized model to be further checked by a model checker – Kind2 [16] (⑥). In case the formal properties are proved valid, Kind2 (⑦) will produce three kinds of proof certificates in SMT-LIB [8] and Logic Framework with Side Condition (LFSC) [39] formats (⑧). In case the formal properties are disproved, we will leverage the counter-example and the blame assignment feature [28] of Kind2 to help system engineers to localize the issues. The first one certifies the front-end translation faithfulness, which is in SMT-LIB format. It requires an independently developed model checker for Lustre, e.g., JKind by Collins Aerospace [25], to be part of the certificate generation process. The second one encodes the k-inductive proof steps in SMT-LIB format for the validity of formal properties and can be independently verified by third-party SMT solvers, e.g., Z3 [33], cvc5 [7], Yices2 [22] etc. Finally, the LFSC proof certificate is a formal proof that the safety properties are invariants in the system, and can be independently verified by LFSC proof checkers (⑨).

The framework enables a system architect/developer to use SMT techniques to auto-generate system architecture models and utilize a back-end model checker (Kind2) to produce proof certificates from verification of safety properties of system behavior models.

III. Overview of IMA Requirements & OYSTER Annex

IMA architectural requirements state several constraints about the location of components on the IMA cabinets along with network connectivity and bandwidth constraints between several several components situated within the cabinets. Moreover, Virtual Link flows specific information flows from various sensors to actuators. Initially, all the components present in the cabinet are stated along with their port connections and virtual link flows as the OYSTER AADL annex [3].

Essentially, annexes enable descriptions of Domain Specific Languages extending the basic AADL definitions. In our case, the OYSTER annex captures the IMA requirements. OYSTER is a front-end AADL annex language to capture these architectural constraints. Then the actual synthesis is performed by translating the OYSTER annex into SMT and invoking an SMT-solver. The solution returned by the SMT-solver is translated back to AADL which represents the synthesized IMA Architecture. If the set of OYSTER annex constraints are unsatisfiable, we report the UNSAT core from Z3 back to the user. Although UNSAT cores from SMT solvers are not necessarily unique or minimal, reporting it back to the user serves as an initial step towards providing actionable feedback for the user while specifying IMA requirements. The synthesized AADL architecture is subject to further behavior analyses as mentioned in the OYSTER toolchain workflow.

The OYSTER language supports modeling the following constraints: fixed-location constraints (FLCs), co-location constraints (CLCs), resource utilization constraints (UCs), separation constraints (SCs), virtual link constraints (VLCs), and port connection constraints (PCCs). The OYSTER language is also equipped with syntax highlighting and type checking for usability. The OYSTER language and its informal semantics is presented next.

Fixed-location Constraints. FLCs constrain a component X to map to another component Y within the IMA architecture. For example, a General Processing Module (GPM) `GPM_L1` is mapped to a Common Computing Resource (CCR) `CCR_L1` as:

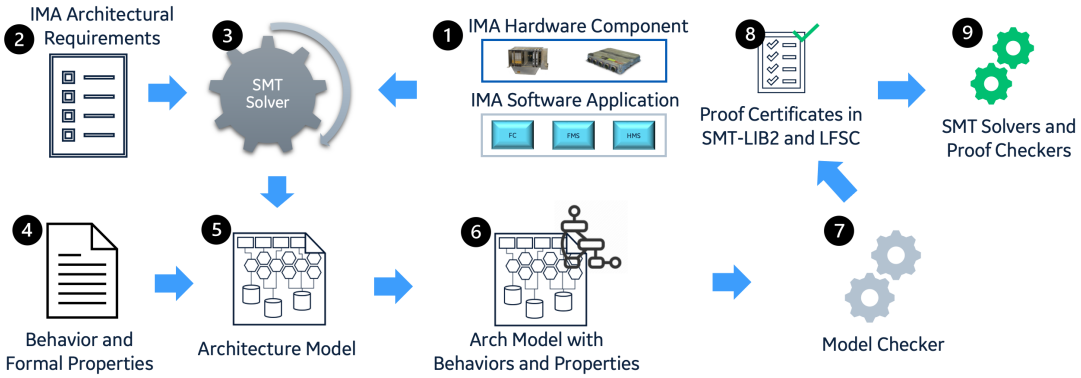

Fig. 1. High-level overview of OYSTER framework

```
{Fixed-Location-Constraint FLC1: (GPM_L1 -> CCR_L1);
```

Co-location Constraints. CLCs co-locate components $\{C_1, ..., C_k\}$ within a same target component C_τ. For example, a GPMApp and ACSApp can be co-located to CCR_L1 as:

```
Co-Location-Constraint CLC1 :
(GPMApp APP_FIDO) and (ACSApp SwitchApp_L1) -> CCR_L1;
```

Utilization Constraints. UCs state that the resources allocated to the various hosted applications (`CPU, RAM, ROM`) should not exceed the resources available on a computing resource. For example, in OYSTER, one could state that the sum of CPU allocated to applications named `APP_FIDO` and `APP_FILE_SYSTEM` should not exceed the CPU provided for the computing resource `CCR_L1`, as shown below.

```
Utilization-Constraint UC1 [CPU]:
(CCR CCR_L1: cpuProvided) > (GPMApp APP_FIDO: cpuUsed) +
↪  (GPMApp APP_FILE_SYSTEM: cpuUsed);
```

Separation Constraints. Separation constraints specify that a given set of components shall not be mapped to same components. The following separation constraint states that the applications `APP_FUEL_SYSTEM_CONTROL`, `APP_FIDO` and `APP_FILE_SYSTEM` should be hosted on different `GPM` component.

```
Separation-Constraint SC1:
  (GPMApp APP_FUEL_SYSTEM_CONTROL, APP_FIDO,
  ↪  APP_FILE_SYSTEM) -> distinct GPM;
```

Virtual Link Constraints. A virtual link constraint (VLC) defines both unicast and multicast flows of a virtual link. All the flows in a virtual link can have only one source publisher, but may have one or more destination subscribers. In addition, a VLC constraint allows users to specify a set of messages for each flow in the virtual link. Each of these sets are separated by a "#". A message in a VLC is represented as MessageSize@RefreshPeriod.

```
-- Message size unit = byte, Refresh period unit = msec
Virtual-Link-Constraint VL1: (App1 ~> App2, App3)
↪  {12@1000} # {12@1000, 12@1000};
Virtual-Link-Constraint VL2: (App4 ~> App5) {20@80};
```

Port Connection Constraints. PCCs specify physical bidirectional connections between two components.

```
--- GPM <=> ACS connections bandwidth unit = byte
Port-Connection-Constraint PCC1: (GPM_L1.portA <->
↪  ACS_L1.port1) 1000000000; -- 1 Gigabyte
```

IV. IMA Synthesis & Proof Certificates Generation

The goal of IMA synthesis is to automatically synthesize an IMA architecture that satisfies all the constraints encoded in the OYSTER annex. The inputs to the synthesis task comprise of an AADL model annotated with AADL properties along with OYSTER constraints. The inputs are then translated to SMT-LIB for constraint solving. In our case, OYSTER uses the Z3 SMT solver. A satisfiable solution from Z3 is then automatically translated to an AADL model containing detailed AADL implementations respecting component locations, their port connections, and virtual link flows satisfying required OYSTER constraints. The OYSTER toolchain provides a plethora of options and toggles for the user to either check or uncheck for Virtual Link Synthesis (feasible or optimal solution), check the Virtual Links' Network Bandwidth Utilization and also the capability to schedule GPM Applications hosted on a designated GPM Processor.

A. From OYSTER Annex to SMT

The components are categorized by their avionics types and declared as SMT enumerated types. For instance, we declare enumerate types ACS and GPM in SMT for the Avionics Cabinet Switch (ACS) and General Processing Module (GPM) respectively.

Fixed-Location Constraints. FLCs are translated to uninterpreted functions. For an FLC, `GPM_L1 -> CCR_L1`, An uninterpreted function $gpm\_to\_ccr : GPM \rightarrow CCR$ is declared and an assertion will be declared: $gpm\_to\_ccr(gpm\_l1) = ccr\_l1$.

Co-Location Constraints. For each component, C_k that is mapped to a target component C_τ, a function $f_{(C_k,C_\tau)}$ is declared and its type is $(Type_{C_k} \rightarrow Type_{C_\tau})$. The co-location of two components C_i, C_j itself to C_τ is declared as a constraint asserting the equality of $f_{(C_i,C_\tau)}$ and $f_{(C_i,C_\tau)}$.

Separation Constraints. Separation Constraint is the dual of Co-location constraint. For components, C_i, C_j to be separated with respect to a target component C_τ. The entire process is the same as that of Co-location constraint, but for the last step where we state $f_{(C_i,C_\tau)} \neq f_{(C_i,C_\tau)}$.

Utilization Constraints. The utilization constraints state that the computing resources provided to the CCR are sufficient to the usage needs of hosted applications. We encode an uninterpreted function of type $(Type_C \times Type_R \rightarrow Int)$ for each component – resource pair, and assert that the sum of the resources used meets the resources provided.

Port Connection Constraints. A port connection constraint c, from $portA$ to $portB$ is represented by declaring a function of type $(Port \times Connection) \rightarrow Port$, where the sort $Connection$ is used to capture the name of the connection itself, from $portA$ to $portB$. Then, the definition is instantiated as an assertion: $port\_connection\_port(portA) = portB$.

Virtual Link Constraints. There is a Virtual Link Constraint between a source s and a destination t, with a refresh rate of r and message size m. Then, Virtual Link Constraints induce multiple sets of SMT constraints, with each set constraining the desired Virtual Link specification in the IMA System. Each such criteria and their associated constraints are described:

- **Path Constraints.** The port connections involved in the Virtual Link from s^i and t^i need to be synthesized for a virtual link i. This synthesis can be formulated as a connection selection problem. The set of connections selected constitute the path between s^i and t^i. Each connection c_{xy} represents a port connection between component x and component y. If a connection c_{xy} is selected, then it should be assigned a weight of 1, otherwise a weight of 0. Furthermore, the sum of all outgoing connections c_{sk} from the source s^i to k must be 1, to enforce only one outgoing flow. The sum of all connections involving an intermediate component not in s, t must equal 2, to enforce one incoming flow and one outgoing flow. Finally, the sum of all incoming connections to c_{kt} must equal 1, to enforce only one incoming flow. Formally,

$$\sum_{l=1}^{outdeg(s)} {c^i}_{sl} = \sum_{l=1}^{indeg(t)} {c^i}_{lt} = 1$$

$$\sum_{l=1}^{indeg(x)} {c^i}_{lx} + \sum_{l=1}^{outdeg(x)} {c^i}_{xl} = 2 \ for \ x \neq s, x \neq t$$

  The shortest path can be selected by optimizing over the sum of weights of all connections in a path from s^i to t^i while satisfying the above constraints. The optimization is performed using MaxSMT solving capabilities of Z3 [13].
- **Bandwidth Constraints.** The virtual links specified in the OYSTER annex must also adhere to the bandwidth constraints on the Aircraft Data Network. The constraints and concepts are defined the ARINC 664 specification [4]. And two important parameters need to be synthesized for each virtual link: Bandwidth Allocation Gap (BAG), Maximum Transmission Unit (MTU). The BAG represents the minimum interval between frames on the virtual link. The MTU represents the largest size of data packet in a single frame that can be transmitted over a network connection. They should also satisfy BAG, MTU, and the jitter constraints. The complete set of ARINC 664 constraints can be found elsewhere [4]. To ensure the paper remains self-contained, we incorporate the summarized formulas from the literature [6]. All the constraints involved are linear constraints over integers and can be straightforwardly encoded in SMT-LIB. For virtual link i, BAG^i denotes its BAG, n^i represents total number of messages in i (indexed from 1 to n^i), s^i_j the message size of j^{th} message, p^i_j the refresh period for the j^{th} message and MTU^i its MTU. Let B denote the bandwidth of the entire network. Then, the following constraints need to satisfied:

  Real-time Constraints on Messages:

$$\sum_{j=1}^{n^i} \frac{\lceil s^j_i / MTU^i \rceil}{p^i_j} \leq 1/BAG^i$$

  Bandwidth Constraints:

$$8 * \sum_{i=1}^{n} \frac{(MTU^i + 67)}{BAG^i} * 10^3 \leq B$$

  Jitter Constraints:

$$40 + 8 * \sum_{i=1}^{n} \frac{MTU^i + 67}{B} \leq 500$$

  General BAG and MTU Constraints:

$$BAG^i \in \{2^k | k \in \mathbb{N} \wedge 1 \leq k \leq 7\}$$

$$MTU^i \in \mathbb{N} \wedge 1 \leq MTU^i \leq 1471$$

GPM Applications Scheduling. The OYSTER toolchain can also generate static schedules for applications hosted on GPM using SMT solvers. Four important characteristics are associated with applications: start time, duration, period, and priority. The start time denotes when to execute an application. Duration defines the time taken to execute an application. Period refers to the frequency of execution of the application. Priority indicates the order to execute an application relative to the other applications. The input to the scheduling problem is the priority, duration and period of applications. The output is the start time for each application so that the priorities are respected and no duration overlaps. The inputs for GPM application scheduling are captured in AADL as an OYSTER property and annotated against GPM applications specified within the IMA system.

We first define the schedulability condition for a pair of applications i, j, followed by constraints involved in the

scheduling problem, where $GCD(x, y)$ denotes the greatest common divisor of integers x, y.

$sched(i, j) \triangleq start_j > start_i \wedge duration_i \leq start_j - start_i \leq GCD(period_i, period_j) - duration_j$

- No scheduling conflicts:

$$\forall i, j \; start_i \geq 0 \wedge start_j \geq 0 \implies sched(i, j)$$

- High priority apps start early:

$$\forall i, j \; priority_i < priority_j \implies start_i < start_j$$

- Applications start times are all distinct:

$$\forall i, j \; i \neq j \implies start_i \neq start_j$$

- Every application must be scheduled: $\forall i \; start_i \geq 0$

UNSAT Cores and Feedback When the constraints specified in the OYSTER annex are unsatisfiable, OYSTER toolchain computes the UNSAT core using Z3 and reports the unsatisfiabile constraints at a high level. If the location constraints are unsatisfiable, then OYSTER recommends the developer to check all fixed-location, separation and co-location constraints that may be inconsistent with each other. UNSAT cores concerning utilization constraints are straightforwardly reported recommending the developer to check the resources allocated (CPU, Memory) against the resources being used. UNSAT cores for virtual links often involving exceeding the maximum bandwidth allocation. In such a case, OYSTER recommends to the developer to either increase the maximum allocated bandwidth or to reduce the number of virtual links allocated. For GPM Application the conflicts in schedulability between pairs of GPM apps are reported back to the developer.

B. Proof Certificates Generation by Model Checking

Another feature of OYSTER is to enhance the synthesized architecture with behaviors and safety properties in the Assume Guarantee REasoning Environment (AGREE) [18] annex of AADL. It utilizes the Kind2 model checker to prove whether the model satisfies the safety properties. In case the properties are proved valid, accompanying proof certificates will be generated by Kind2 and can be independently verified by third party tools to ensure the correctness of the results. Different behavior aspects of IMA such as application execution schedule and latency analysis can be encoded in this framework. In this work, we consider the execution schedule of applications on a GPM. The schedule defines the start time, priority and duration of each task. It is essential for ensuring that the system operates efficiently and effectively, as it helps to resolve conflicts between different applications and functions, prevent overloading of resources, and optimize the use of processing power and memory. We have modeled the application execution schedule as a behavior model in AGREE annex [2]. The formal properties of interests are that pair-wise applications shall not have any conflicts. To further increase the stakeholders' confidence in the correctness of the IMA solutions, we introduce additional model checking layer to ensure the correctness of schedules by the proof certificates generated by the model checker.

V. Evaluation

To demonstrate the capabilities of the OYSTER tool[1], GE Aviation developed a smaller-scale (yet fully-defined) IMA Architecture for a rotorcraft air vehicle. One of the major avionics systems of this architecture is the Fuel Control System, which we have chosen as the basis of the OYSTER use case. More specifically, the focus is on the IMA aspects of the architecture that host this Fuel Control System ARINC 653 Application and gateway its data throughout the Aircraft Data Network (ADN). The Fuel Control Application is responsible for managing various aspects of the Fuel Control System of an aircraft. These functions include pumping of fuel, delivering fuel to the engines, monitoring fuel flow, etc. This application allows the flight crew to control fuel tank selection, shutoff valve functions, and the main and standby pumps. It also provides monitoring and reporting of fuel system characteristics such as fuel quantity, temperature, and pressure. A notional diagram of the IMA fuel system control application use case can be found elsewhere [1].

The OYSTER toolchain was developed as a plugin for the Open Source AADL Tool Environment (OSATE) [15]. The IMA use case involves 43 IMA components that also involves 5 virtual links, and 6 applications to be scheduled on a particular GPM (on GPM_R2). The total number of OYSTER constraints are in the order of hundred constraints. Configuring them manually can be a challenging task. OYSTER makes it easier to specify these constraints and uses formal methods tools to synthesize correct-by-construction solutions. The IMA architecture synthesis takes $7.751s$. The schedule synthesized for our use case was encoded in AGREE annex to simulate the execution of schedules. We consider 15 formal properties for 6 applications, and the entire process for proving properties takes $6.441s$. However, with proof certificates generation, the process takes $33.196s$, which is expected because proofs generation is expensive. We have also successfully validated the correctness of the proof certificates by running Z3, cvc5, and LFSC checker. The performance evaluation indicates that OYSTER is usable practically. We leveraged a model checker to generate proof certificates to ensure formal properties of the behavior model are indeed valid. For our example use case (application schedules), the model checking scales well (proved all properties in under 3 mins) to a typical number of applications (10 apps) that one would expect in practical IMA systems. All experiments were conducted by running OYSTER on an `Intel(R) Core(TM) i7 CPU @ 2.9GHz` Processor with 4 cores and 16 GB RAM running macOS Ventura (Version 13.1) .

A. Lessons Learned

For any aircraft development program, the airworthiness of the entire aircraft must be established through a rigorous certification process. This extends not just to the airframe itself, but also to the computing systems installed on the aircraft (i.e. the IMA). One key aspect of an IMA is that a

[1]OYSTER GitHub: https://github.com/ge-high-assurance/OYSTER

majority of system functions (both safety critical and non-safety critical) are now implemented as software applications running on the IMA platform. As such, this airborne software is also required to adhere to the same rigorous certification process. Guidance for certifying airborne software is provided in DO-178C. In order to make OYSTER part of the GE Aviation IMA development pipeline, there are several key challenges to overcome.

- Since the goal of OYSTER is to automate and replace some of the steps currently performed by the existing qualified toolchain, it will have to meet all of the objectives of DO-178C Software Considerations in Airborne Systems and Equipment Certification or DO-330 tool qualification standard.
- There are several steps in our process where translation of data has to occur to go from one tool to the next. Each of these points will have to have a qualified verifier developed to prove that nothing was corrupted/changed/lost during transformation.
- One major concern is the format of the formal proof certificates and other verification evidence that is being produced. It is not easily human-readable, and even if the format was changed to something easier to read, an FAA certification authority with no knowledge of formal methods would be unable to understand the proofs. We cannot assume that aviation/avionics experts will have this expertise.
- We expect the learning curve for a systems integrator on a real aircraft development program to be able to use these tools without help will be steep. Training for system developers will be needed. We also need to understand the level of expertise with formal methods tools such as SMT, Kind2 model checker, and AADL modeling that they are expected to have.

VI. Related Work

Several works exist in the literature on the generation of schedules and architectural models. SMT-based system scheduling synthesis for applications have been proposed for time-triggered platforms [12], [11], [10] and TTEthernet networks [19]. SMT-based techniques have also been proposed for the synthesis [24], [35] and refinement [21], [20], [26] of system architectures. SAT-solving techniques have been proposed for generating architectural models [31], [37]. Correct-by-construction techniques for developing architectural models include approaches that use the "B" method [29], linear temporal logic [34], [41], mixed integer programming [42], AADL-based tools and techniques have been developed for synthesis [27], reconfiguration [43], and verification [32], [38], [40], integrated design modeling [14], and domain-specific languages [5]. The CoBaSa framework has been applied to industrial-scale IMA architecture synthesis problems [31] and is closely related to our work. The difference lies in the use of solvers and solver theories. CoBaSa uses Pseudo-boolean (PBSAT) and Integer Linear Programming (ILP) solvers to perform IMA architecture synthesis [31], [30], whereas OYSTER uses modern SMT solvers with combination of Quantifier-free Equality under Uninterpreted Functions (QF_EUF), Quantifier-Free Linear Integer Arithmetic (QF_LIA), Boolean and Algebraic Datatype theories. OYSTER encodes IMA constraints in SMT allowing for reporting of UNSAT cores, which could easily localize issues and provide useful feedback to journeyman developers about the constraints being violated; whereas CoBaSa does not.

VII. Conclusion and Future Work

We have presented a formal language and end-to-end framework called OYSTER to automatically synthesize aspects of industrial IMA platforms. The language enables users to encode IMA architecture constraints. The toolchain takes in the AADL models annotated with OYSTER constraints as input and auto-synthesizes a correct-by-construction IMA architecture instantiated with implementation details. The synthesized architecture can be annotated with behavior models and safety properties. The safety properties can then be discharged by the Kind2 model checker, which is guaranteed to be correct by the formal proofs generated by the model checker. Users may independently verify the correctness of the proofs by using third-party SMT solvers and proof checkers. The framework was applied on a use case provided by GE Aviation, and the evaluation showed promising results along with lessons learned. One potential direction of future work would be to support multi-core scheduling (e.g., schedule GPM applications in multiple GPM processors) and integrate OYSTER with GE Aviation's existing development pipeline. Another research direction is to extend OYSTER to support more aspects of IMA and seek certification for OYSTER solutions.

Acknowledgement & Disclaimer: Distribution Statement "A" (Approved for Public Release, Distribution Unlimited). This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

References

[1] OYSTER Integrated Modular Avionics Use Case Arthictecture. https://github.com/ge-high-assurance/OYSTER/blob/main/models/notional_architecture/IMA.PNG (2022), [Online; Accessed 2023-08-14]

[2] OYSTER Integrated Modular Avionics Use Case Behavior Model. https://github.com/ge-high-assurance/OYSTER/blob/main/models/FuelControlSystem_Behavior/FuelControlSystem_Behavior.aadl (2022), [Online; Accessed 2023-08-14]

[3] OYSTER Language Grammar. https://github.com/ge-high-assurance/OYSTER/blob/main/tools/plugin/verdict/com.ge.research.osate.oyster.dsl/src/com/ge/research/osate/oyster/dsl/Oyster.xtext (2022), [Online; Accessed 2023-08-14]

[4] Airlines Electronic Engineering Committee: Aircraft Data Network Part 7, AFDX NETWORK, ARINC Specification 664 (2002)

[5] Alves, R., Amaral, V., Cintra, J., Tavares, B.: A Family of Domain-Specific Languages for Integrated Modular Avionics. In: International Conference on the Quality of Information and Communications Technology. pp. 239–254. Springer (2019)

[6] An, D., Kim, K.H., Kim, K.I., et al.: Optimal Configuration of Virtual Links for Avionics Network Systems. International journal of aerospace engineering **2015** (2015)

[7] Barbosa, H., Barrett, C., Brain, M., Kremer, G., Lachnitt, H., Mann, M., Mohamed, A., Mohamed, M., Niemetz, A., Nötzli, A., et al.: cvc5: A Versatile and Industrial-Strength SMT Solver. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 415–442. Springer (2022)

[8] Barrett, C., Stump, A., Tinelli, C.: The SMT-LIB Standard: Version 2.0. In: Proceedings of the 8th international workshop on satisfiability modulo theories (Edinburgh, UK). vol. 13, p. 14 (2010)

[9] Barrett, C., Tinelli, C.: Satisfiability Modulo Theories. Springer (2018)

[10] Beji, S., Hamadou, S., Gherbi, A., Mullins, J.: SMT-based cost optimization approach for the integration of avionic functions in IMA and TTEthernet architectures. In: 2014 IEEE/ACM 18th International Symposium on Distributed Simulation and Real Time Applications. pp. 165–174. IEEE (2014)

[11] Biewer, A., Andres, B., Gladigau, J., Schaub, T., Haubelt, C.: A symbolic system synthesis approach for hard real-time systems based on coordinated smt-solving. In: 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE). pp. 357–362. IEEE (2015)

[12] Biewer, A., Gladigau, J., Haubelt, C.: Towards tight interaction of asp and SMT solving for system-level decision making. In: ARCS 2014; 2014 Workshop Proceedings on Architecture of Computing Systems. pp. 1–7. VDE (2014)

[13] Bjørner, N.S., Phan, A.: νZ - Maximal Satisfaction with Z3. In: Kutsia, T., Voronkov, A. (eds.) 6th International Symposium on Symbolic Computation in Software Science, SCSS 2014, Gammarth, La Marsa, Tunisia, December 7-8, 2014. EPiC Series in Computing, vol. 30, pp. 1–9. EasyChair (2014). https://doi.org/10.29007/jmxj, https://doi.org/10.29007/jmxj

[14] Bonev, M., Hvam, L., Clarkson, J., Maier, A.: Formal computer-aided product family architecture design for mass customization. Computers in industry **74**, 58–70 (2015)

[15] Carnegie Mellon University: Open Source AADL Tool Environment (OSATE). https://osate.org/index.html (2016), [Online; Accessed 2023-08-14]

[16] Champion, A., Mebsout, A., Sticksel, C., Tinelli, C.: The Kind 2 model checker. In: International Conference on Computer Aided Verification. pp. 510–517. Springer (2016)

[17] Clarke, E.M.: Model checking. In: Foundations of Software Technology and Theoretical Computer Science: 17th Conference Kharagpur, India, December 18–20, 1997 Proceedings 17. pp. 54–56. Springer (1997)

[18] Cofer, D., Gacek, A., Miller, S., Whalen, M.W., LaValley, B., Sha, L.: Compositional Verification of Architectural Models. In: NASA Formal Methods Symposium. pp. 126–140. Springer (2012)

[19] Craciunas, S.S., Oliver, R.S.: SMT-based Task-and Network-Level Static Schedule Generation for Time-Triggered Networked Systems. In: Proceedings of the 22nd international conference on real-time networks and systems. pp. 45–54 (2014)

[20] Delmas, K., Delmas, R., Pagetti, C.: SMT-based Architecture Modelling for Safety Assessment. In: 2017 12th IEEE International Symposium on Industrial Embedded Systems (SIES). pp. 1–8. IEEE (2017)

[21] Delmas, K., Delmas, R., Pagetti, C.: SMT-based Synthesis of Fault-Tolerant Architectures. In: International Conference on Computer Safety, Reliability, and Security. pp. 287–302. Springer (2017)

[22] Dutertre, B.: Yices 2.2. In: International Conference on Computer Aided Verification. pp. 737–744. Springer (2014)

[23] Feiler, P.H., Gluch, D.P.: Model-based engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language. Addison-Wesley (2012)

[24] Finkbeiner, B., Schewe, S.: SMT-based Synthesis of Distributed Systems. In: Proceedings of the second workshop on Automated formal methods. pp. 69–76 (2007)

[25] Gacek, A., Backes, J., Whalen, M., Wagner, L., Ghassabani, E.: The JKind Model Checker. In: International Conference on Computer Aided Verification. pp. 20–27. Springer (2018)

[26] Goldman, R.P., Bryce, D., Pelican, M.J., Musliner, D.J., Bae, K.: A Hybrid Architecture for Correct-by-Construction Hybrid Planning and Control. In: NASA Formal Methods Symposium. pp. 388–394. Springer (2016)

[27] Hardin, D.S., Slind, K.L.: Formal Synthesis of Filter Components for Use in Security-Enhancing Architectural Transformations. In: 2021 IEEE Security and Privacy Workshops (SPW). pp. 111–120. IEEE (2021)

[28] Larraz, D., Laurent, M., Tinelli, C.: Merit and Blame Assignment with Kind 2. In: Formal Methods for Industrial Critical Systems: 26th International Conference, FMICS 2021, Paris, France, August 24–26, 2021, Proceedings 26. pp. 212–220. Springer (2021)

[29] Lecomte, T., Servat, T., Pouzancre, G., et al.: Formal Methods in Safety-Critical Railway Systems. In: 10th Brasilian symposium on formal methods. pp. 29–31 (2007)

[30] Manolios, P., Papavasileiou, V.: ILP Modulo Theories. In: Computer Aided Verification: 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings 25. pp. 662–677. Springer (2013)

[31] Manolios, P., Vroon, D., Subramanian, G.: Automating Component-Based System Assembly. In: Proceedings of the 2007 international symposium on Software testing and analysis. pp. 61–72 (2007)

[32] Meng, B., Larraz, D., Siu, K., Moitra, A., Interrante, J., Smith, W., Paul, S., Prince, D., Herencia-Zapana, H., Arif, M.F., et al.: VERDICT: A Language and Framework for Engineering Cyber Resilient and Safe System. Systems **9**(1), 18 (2021)

[33] Moura, L.d., Bjørner, N.: Z3: An Efficient SMT Solver. In: International conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 337–340. Springer (2008)

[34] Nilsson, P., Hussien, O., Balkan, A., Chen, Y., Ames, A.D., Grizzle, J.W., Ozay, N., Peng, H., Tabuada, P.: Correct-by-construction adaptive cruise control: Two approaches. IEEE Transactions on Control Systems Technology **24**(4), 1294–1307 (2015)

[35] Peter, S., Givargis, T.: Component-based synthesis of embedded systems using satisfiability modulo theories. ACM Transactions on Design Automation of Electronic Systems (TODAES) **20**(4), 1–27 (2015)

[36] Prisaznuk, P.J.: Integrated Modular Avionics. In: Proceedings of the IEEE 1992 National Aerospace and Electronics Conference@ m_NAECON 1992. pp. 39–45. IEEE (1992)

[37] Reimann, F., Lukasiewycz, M., Glass, M., Haubelt, C., Teich, J.: Symbolic system synthesis in the presence of stringent real-time constraints. In: Proceedings of the 48th Design Automation Conference. pp. 393–398 (2011)

[38] Siu, K., Moitra, A., Li, M., Durling, M., Herencia-Zapana, H., Interrante, J., Meng, B., Tinelli, C., Chowdhury, O., Larraz, D., et al.: Architectural and Behavioral Analysis for Cyber Security. In: 2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC). pp. 1–10. IEEE (2019)

[39] Stump, A., Oe, D., Reynolds, A., Hadarean, L., Tinelli, C.: SMT proof checking using a logical framework. Formal Methods in System Design **42**(1), 91–118 (2013)

[40] Suo, D., An, J., Zhu, J.: AADL-based modeling and TPN-based Verification of Reconfiguration in Integrated Modular Avionics. In: 2011 18th Asia-Pacific Software Engineering Conference. pp. 266–273. IEEE (2011)

[41] Wongpiromsarn, T., Topcu, U., Murray, R.: Formal Synthesis of Embedded Control Software: Application to Vehicle Management Systems. In: Infotech@Aerospace 2011. p. 1506 (2011)

[42] Xuan, Z., Xiong, H., Feng, H.: Hybrid partition-and network-level scheduling design for distributed integrated modular avionics systems. Chinese Journal of Aeronautics **33**(1), 308–323 (2020)

[43] Zhang, Q., Wang, S., Liu, B.: Approach for Integrated Modular Avionics Reconfiguration Modelling and Reliability Analysis based on AADL. Iet Software **10**(1), 18–25 (2016)