Formal Enforcement of Mission Assurance Properties in Cyber-Physical Systems

Scott Harper, Jonathan Graf
Graf Research
Blacksburg, Virginia
{scott, jon}@grafresearch.com

Michael A. Capone, Justin Eng,
Michael Farrell, Lee W. Lerner
Georgia Tech Research Institute (GTRI)
Atlanta, Georgia
{mcapone6, jenglish7, lee.lerner}@gatech.edu
michael.farrell@gtri.gatech.edu

***Abstract*— Cyber-Physical Systems improve efficiency, accuracy, and access in systems ranging from household appliances to power stations to airplanes. They also bring new risks at the intersection of physical, information, and mission assurance. This paper presents CP-SMARTS, a framework providing a means for propagating CPS assurances from planning to deployment.**

Keywords—Cyber-Physical Systems; Formal Methods; Model Checking; Mission Assurance

I. Introduction

Cyber Physical Systems (CPSs) promise to bring the advancements of the cyber domain – instant communications, advanced computation, remotely-updatable software and firmware – to processes controlled in the physical world. Driven by the need for improved reliability, efficiency, accuracy, and speed, CPSs are all around us, from controls on household appliances, to power stations, industrial processes, and military equipment. While this rapid escalation of CPS is accelerating benefits of the cyber domain into our physical environment, it is also brings with it risks. The attack surface presented by a CPS allows an adversary to directly employ well-honed and mature cyber hacking tools in this new domain. Effects of these hacks can be extremely deleterious, especially if used on safety or military systems. Similarly, the impact of poorly engineered or improperly updated CPS firmware can be amplified into a serious physical-world emergency because of the criticality of the physical systems under cyber control. With the wide range of naturally occurring and adversarial threats targeting critical embedded systems used across a variety of platforms, the need for comprehensive categorization and mitigation of any possible failure modes facing a particular mission is vital.

The challenge described above is a traditional subject of CPS security research, a burgeoning field in academia and industry. A major focus of this traditional domain has been on creating trustworthy and reliable Cyber-Physical Controllers (CPCs). A single secure CPC is only one necessary component of a CPS, by itself insufficient to enforce protections. Consideration of CPS security in the context of a larger collection of systems or objectives requires mission awareness to permeate the full set of interacting cyber systems. When modeling and deploying cyber physical systems, each CPC must be viewed in the context of the role it plays in a mission. As the bridge between the cyber and physical domains, CPCs must be subject to information assurance principles on the cyber side and provide feedback throughout the interacting system of sensed action on the physical side. To achieve this goal, there must be a mechanism in place to flow verifiable higher-level mission objectives down to the processes used for development and deployment of full CPSs.

This paper describes a framework for cyber physical security called CP-SMARTS: *Cyber Physical Security for Mission-Aware ARmy Tactical Systems*. The framework aims to improve Mission Assurance (MA) capabilities and provide resiliency during the fulfillment of Mission Essential Functions (MEFs) by flowing verifiable rules from planning to hardware. Leveraging existing techniques within the realm of formal verification, it helps prove that a specific set of functionalities within a platform can be guaranteed to perform reliably and consistently under conditions presented by mission scenarios. A core element of the CP-SMARTS philosophy is to keep deployment in mind. This means creating methods that integrate well into common development and configuration environments. In so doing, it provides technologies that not only work on the whiteboard and in simulation but also can be readily adopted by commercial and military CPS designers.

II. Background

A variety of mission planning systems exist that can combine goals and constraints to produce rules that must be followed by mission actors. The rules resulting from these planning systems define high level requirements characterizing the expected behavior of equipment such as an unmanned aerial vehicle (UAV) during a mission. The CP-SMARTS framework tackles the problem of providing formal MA by interpreting these abstract specifications, decomposing them

This work was sponsored by the Office of the Secretary of Defense (OSD) under Contract No. W56KGU-16-C-0080. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Office of the Secretary of Defense.

into component guidance, and propagating them to linked CPC modeling and development environments.

Previous work in the space of CPS modeling and development includes both custom modeling frameworks and provable design languages. Many current systems use specifically developed modeling environments for CPS evaluation. It is clear, however, that the choice of modeling environments used in typical MA-oriented planning and development is dependent on the system(s) being modeled and should not be restricted by an assurance framework. Ideally, properties derived from mission planning should flow directly into the full range of models used for development and deployment of CPS resources.

A. *CPS MA Property Specification and Decomposition*

There are many levels of mission planning ranging from battlespace or theatre level (e.g. a collection of radios, tanks, UAVs, personnel, etc. with a common objective), through integrated component collections (e.g. a set of endpoints on a tactical network), to single independent systems (e.g. a satellite). Planning approaches exist at all levels, and the choice of a level for modeling is often a matter of perspective. CP-SMARTS is targeted at integrated component and system development. As such, system level guidance flows down to component models. An example of an approach that can produce top-level requirements related to this level of consideration is the NASA Process-based Mission Assurance Knowledge Management System (PBMA-KMS) [1], a wizard that produces assurance process requirements. Low level formal properties can be produced for assessment using formal verification techniques by leveraging established analysis techniques to decompose these requirements into lower level properties (and subsequent rules). Several methods for system decomposition exist, including Axiomatic Design Theory (ADT) [2] and Risk Analysis Graphs (RAGs) [3].

The ADT framework was proposed by Suh at the Massachusetts Institute of Technology in 1990, and provides a formal methodology under which a system can be decomposed into separate components. The approach targets a variety of hardware and software systems design and has applications in Cyber Physical Systems (CPS) design [4]. While originally developed for product lifecycle management, ADT has several underlying and fundamental principles that might benefit a framework like CP-SMARTS. With modification, it is possible that ADT can be used to develop rules and constraints that can be used to characterize the "correct" behavior of an underlying CPS.

Within the context of a system decomposition, ADT provides a useful tool to identify how a particular system might go about meeting a set of design goals. It can relate a series of given MEFs to component and sub-system objects upon which a given MEF might depend. As an example, if a MEF required that an unmanned aerial vehicle operated with a no-fly zone restriction, then ADT would allow for detailed relationships to be extrapolated. This extrapolation exemplifies the dependencies on the UAV Command and Control Algorithms (CCAs), system level sensors such as GPS, as well as the specific processes involved in control of the flight surfaces. Operational and security rules could be generated which specify the expected and "correct" set of behaviors for these components, and formal constraints would then be derived for enforcement in hardware and verification. Using probability theory within ADT, quantifiable metrics could be produced as to the likelihood of "enforceability" of a set of MEFs.

Another approach to decomposition of MA rules is proposed by Vai, et al., in the paper *Systems Design of Cybersecurity in Embedded Systems* [3]. The authors propose the Risk Analysis Graph approach towards secure CPS design to emphasize the analysis of system behavior when certain critical functionalities are lost as the result of either malicious cyber-attacks or system faults. The RAG process begins with an identification of all mission objectives (analogous to MEFs). These mission objectives are hierarchically mapped towards a series of potential attack categories, which indicate at a high level any possible failures that can threaten the successful fulfillment of a given mission objective during the execution of a mission. The identification of attack categories focuses on the overall effect that a failure might incur, rather than the details of how a failure was instigated. For example, enemy jamming of onboard radios would be represented as a "loss of communications". Given this effect-driven perspective, RAGs do not attempt to tackle the formidable problem of identifying all possible attack vectors or potential faults and instead focus on how the effects may be mitigated at a system level.

Following identification of all attack categories that threaten a set of mission objectives, the RAG approach then maps the categories to specific system functions that identify the system-level mechanisms that would lead to the realization of the identified failure modes. Each attack category is compared against the design of the CPS to single out what specific abstract functions would need to experience failure to realize the category in question. Subsequently, each mission objective is then able to trace any potential risks to its fulfillment down to the failure of a specific functionality. These system functions can then trace their own dependencies down to specific subsystems, representing hardware or software components that support the overall function. This yields a comprehensive breakdown akin to that shown in Fig 1.

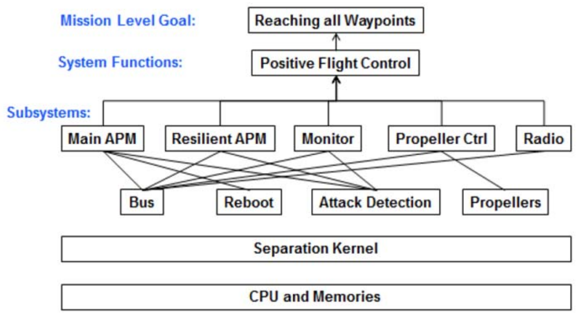

Fig 1. RAG approach to breaking a MEF into System Functions and Subsystem dependencies (from Vai, et al.)

Once all objectives have been hierarchically mapped to any identified attack categories, which are in turn related to system functions and subsystems, RAG proposes the computation of resiliency metrics that can mathematically represent the likelihood of fulfilling an objective. By taking a fault tree analysis approach towards identify failure probabilities at the subsystem and system function level, a group of Subject Matter Experts (SMEs) can then provide design and vulnerability analysis of the CPS in question to produce a quasi-quantitative scoring that indicates the likelihood of failure. Although the RAG approach is oriented at identifying aspects for added redundancy in a system, a similar approach might be taken to address the CP-SMARTS goal of trustworthy component generation.

B. *Integrated CPS Modeling*

The DoD High-Level Architecture (HLA) is an established framework that was formed to provide a standard architecture for design and integration of simulators [5]. HLA provides an API that contains a model for sharing and synchronizing data across disparate simulators. While it is not in itself a modeling tool, it is supported by many existing simulators and provides a standard means of integrating them. The use of HLA as an umbrella for cyber-physical system modeling has previously been shown to be a useful approach [6], and has been both standardized (IEEE-1516) and called out as a standard architecture for DoD use in the 1995 Modeling and Simulation Master Plan (5000.59-P) as a common technical framework to facilitate interoperability. Extending the HLA architecture to support stateless CPS MA requirement information would provide a solid standards-oriented solution for MA constraint propagation.

An example of HLA use in multi-simulator modeling was proposed and demonstrated for a wind tunnel modeling effort by Davis, et al. (Fig 2) [7]. Their testbed was designed to model a system of cyber-physical hardware elements interacting via a SCADA network with a master providing monitoring and control functions. The goal was to assess vulnerabilities of the implementation in a realistic setting and to test new solutions for next-generation systems. To achieve this goal, they were aiming to develop a modular and reconfigurable test infrastructure that could provide a realistic model of a real-world process. The solution they arrived at combined Simulink [8], OMNeT++ [9], Colored Petri Nets, and Delta3D [10] domain-specific models under an HLA umbrella. The result provided a good demonstration of behavioral modeling of a CPS and analysis of the impact of inserting known exploits into the system.

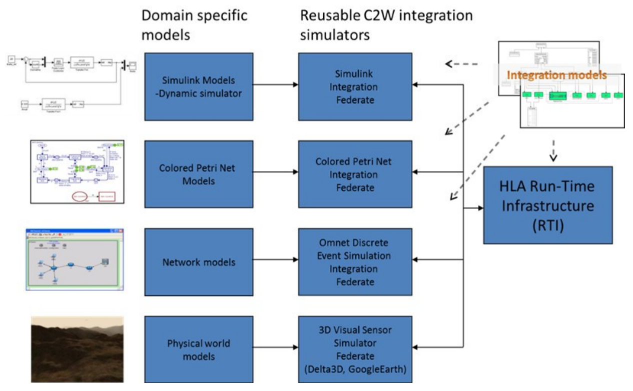
Domain specific models
Reusable C2W integration simulators
Simulink Models -Dynamic simulator
Simulink Integration Federate
Integration models
Colored Petri Net Models
Colored Petri Net Integration Federate
HLA Run-Time Infrastructure (RTI)
Network models
Omnet Discrete Event Simulation Integration Federate
Physical world models
3D Visual Sensor Simulator Federate (Delta3D, GoogleEarth)

Fig 2. Model Integration under an HLA umbrella (from Davis, et al.)

While that work demonstrated a promising approach for experimenting with known vulnerabilities in a typical system Verification and Validation environment, the integrated testbed did not provide a solution for integration of mission assurance (MA) goals, formalized vulnerability assessment, or forward-flow design with the intent to incorporate MA-derived rules into CPC design or mission-specific configuration. In fact, HLA's focus on behavioral simulation aspects such as coordination of timing and data transfer across disparate simulator does not leave it inherently suited for MA integration aspects such as rule propagation and cross-model rule analysis. It is conceivable, however, that the framework could be extended to add in these aspects in support of combined modeling and formal MA requirements analysis.

C. *CPS Mission Assurance Property Verification*

There exists various literature on the state of hardware and software verification. Two approaches that operate on full system models are the Functional Failure Identification and Propagation (FFIP) framework [11] and System Level Autonomy Trust Enablers (SLATE) [12]. The former is targeted at early (block diagram) design prior to the availability of high-fidelity models, and the latter is tightly coupled with its own modeling environment and rule types (constrained bounds checks). A generally-applicable approach would ideally move beyond constrained point solutions such as these and into a broader framework that more widely supports the aspects of CPS design, development, and deployment. With this goal in mind, we consider the relationship between rule definitions and proving solutions.

One approach to top-level rule specification is the use of a rule-based declarative language such as Prolog or the related Datalog. These languages inherently allow for combined rule analysis and proving. Using this method for top-level rule specification can provide a portable solution capable of integrating feedback from lower level models. Several existing vulnerability analysis approaches use this type of foundation to process attack tree information and system properties to discover vulnerabilities [13]. The use of a declarative language as the system-level representation of rules would allow these solutions to be easily integrated into a development and verification flow and provides a clear path to model proving result integration.

Another tactic that provides a general approach to rule propagation is the development of formal definitions to specify the behavior of a system using propositional calculus. Employing a model checking approach with this type of specification can make use of a variety of tools to formally prove a design satisfies its governing security rules, and by extension any desired Mission Essential Functions [14]. Some drawbacks are inherent with this technique however, and there exists a nontrivial difficulty in establishing correct formal definitions for a system. Errors in the modeling process could reduce the effectiveness of any proofs verified, and there also exists the possibility of state space explosion which might hinder the process of formally verifying a set of constraints [15]. Solutions to this do exist in practice and, as in the case of one solution called Bounded Model Checking [16], aim to reduce the size of the state space to a tractable form.

Constraining a system typically requires that certain properties such as liveness and safety are enforced over time and for the duration of a system's operation. There are various forms of model checking methodologies used to ensure that a set of properties can be upheld in a system. Typically, a mathematical formalism known as temporal logic is employed to create logical statements which can appropriately express expected behavior. A core concept involves the use of a set of temporal operators, each of which can be used to specify how a given statement should hold across the states of the system. A model checking process is then responsible for performing temporal logic verification across the system model.

One common form of propositional temporal logic is Linear Temporal Logic (LTL) [17]. Like other forms of these logics, it uses atomic propositions (statements or assertions that must be true or false) and boolean connectives (e.g. conjunction, disjunction, etc.) to create complicated yet expressive properties that form the basis of safety or liveness properties. Temporal operators within LTL can be used to constrain a system, and each of these operators can be evaluated over a linear path through a Kripke structure [18] to verify system properties. For example, a top-level UAV property example might make use of the "Global" operator to specify an altitude maximum that the platform should always be beneath in all states. Computational Tree Logic (CTL) [19] is another temporal logic that extends the proposition space by introducing additional operators to allow for evaluation of conditions across multiple branches. This allows it to supports the evaluation of temporal operators across branching paths through a state space. CTL is an extremely powerful methodology, however support for formal verification using this type of logic is extremely limited due to its complexity.

Assertion based verification (ABV) is a formal verification method that has comparatively wide tooling support within hardware component design. It differs from more general LTL or CTL based model checking in that it was created specifically with implementation as a component of hardware description languages in mind. One of the most common forms of ABV is the use of SystemVerilog Assertions (SVAs) [20]. These assertions provide a means by which the state space of hardware modules can be specified for both simulation and formal verification. SVA implements evaluation support for a subset of temporal operators and goals like safety and liveness properties can be expressed for verification. Despite a limited expressiveness due to SVA not implementing the whole gamut of temporal operators, the tool support for SVA based verification provides a promising path forward for verifying MA properties and functionality at the hardware component level within the context of a multi-component CPS framework.

To further improve on the assurances provided with formalisms at the CPS level, it is important to consider formalisms of the mission specifications and MEFs themselves. While the guidelines produced during the system decomposition are in turn used to create formal properties and rules, the same processes used to model check the underlying CPS can be used to formally verify properties that might describe the mission specifications directly. If mission specifications and MEFs are encoded as a series of formal properties and rules, it is then possible to leverage active research in the formal specification of mission planning such as that proposed by Humphrey, et al. [21] to model check the behavior of a CPS against the mission requirements directly.

While this may initially seem like a superfluous extension of the formal verification being conducted at the subsystem and system function level, it is important to note that subjecting the model of a system to low level system guards in addition to high level mission planning requirements can provide a much more comprehensive level of overall mission assurance than either a low or high level set of assurances would be able to provide individually. With a model checking process capable of assuring that high level mission specifications are formally verified to be met, a framework like CP-SMARTS can conclusively verify whether low-level subsystem assurances identified during the system decomposition stage are in fact sufficient to support all MEFs (Fig 3).

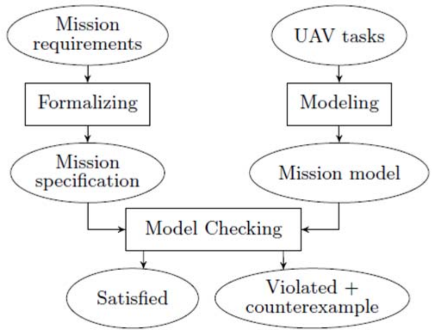

Fig 3. Canonical model checking for verification of mission requirements (from Humphrey, et al.)

Similar to the processes outlined for formal verification and model checking of CPS subsystems, LTL can be used to specify mission requirements in a formal temporal logic by chaining together operators. Linking operators allows for a high level of expressiveness. While one might argue the case for CTL levels of branching expressiveness, most existing literature and tool support would provide heavier weight towards an LTL approach. Furthermore, the complexity of most mission specifications might be considered linear to the point that CTL might be overkill for the goals of a CPS MA enforcement framework like CP-SMARTS.

D. Hardware enforcement of MA properties

The CP-SMARTS framework is intended to enable propagation of top-level MA guidance into both design and deployment of CPS resources. As such, it must be able to support integration with embedded hardware and software root-of-trust components to provide mission-level assurances of operational qualities.

One approach to propagation of high-level properties into hardware solutions is the concept of proof-carrying hardware [22]. Derived from proof-carrying code (PCC), the idea of proof-carrying hardware (PCH) is to allow for consumer verification of security properties at runtime without the need for a Trusted Platform Module (TPM) or secure channel. This approach places the burden for establishing security on the producer of the module. The producer is required to develop in a manner that is compatible with a proving language such as that used by the Coq proof assistant [23], then hardware development is done using these rules. While this is an interesting approach, it restricts development, modeling, and deployment methods. Furthermore, it provides assurance of properties in a CPS, but does not enforce those properties in the event of run-time aspects such as tampering or malicious attack.

A more general approach that includes run-time enforcement is the use of Hardware Enforcement Guards (HEGs). At an abstract level, HEGs share many similarities to the "shims" expressed in the VeriDrone project [24]. A HEG is a synthesized piece of hardware that is placed in-between a potentially insecure control system and any peripheral modules, where peripheral modules include the sensors, input signals, motors, etc. attached to the system. The placement of the HEG ensures that the guard is capable of deriving context from the system inputs as well as enforcing policies defined by the mission requirements. This approach supports standard development flows and clearly fits within a rule propagation and decomposition framework concept.

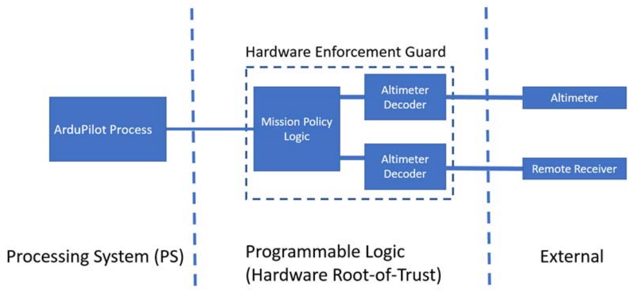

Fig 4. Hardware Enforcement Guard controlling UAV input data from a remote pilot based on altimeter data.

As an example of the HEG approach, assume a set of MEFs requires a UAV to maintain altitude once in flight, while also being responsive to commands from a UAV pilot. In the event that the pilot attempts to decrease the altitude (either in a benign or malicious manner) below the threshold, the HEG can prevent the system from reacting to the command by parsing the command input and the altimeter data to determine that the commanded action violates the guard specification set forth by the mission requirement. In this scenario, the HEG could simply prevent the command from being received by a software executing on an embedded processor. The block diagram of a theoretical HEG proposed in the hypothetical scenario is illustrated in Fig 4. The HEG can be altered and adapted as necessary depending on the mission requirements as well as the configuration of the system. The only requirement of an individual HEG is that it has direct access to contextual information from sensors, etc. and is capable of directly controlling the behavioral output.

III. CP-SMARTS

The objective of CP-SMARTS is to provide a mechanism to enforce formal assurances within embedded hardware and software root-of-trust components to provide mission-level assurances of operational qualities. The framework enables system protection by flowing verifiable system security properties from planning through development and into deployment. This is done by overlaying an inter-model rule propagation and checking system on top of a multi-model system environment. A combination of a top-level generic property framework with lower-level property translation mechanisms is used to enable interaction with a wide variety of planners, models, and development environments. The approach is designed such that it can be used to extend and enhance other model integration approaches such as the DoD HLA. This design choice allows CP-SMARTS to be readily integrated into a wide range of CPS planning, development, and deployment styles. The result of applying this framework is an improved understanding of vulnerabilities for a given CPS as well as the ability to enforce verifiable guarantees of operation and run-time protections as it is developed and deployed.

A. CP-SMARTS Foundation

CP-SMARTS consists of a defined top-level provable rule specification, an automatable approach for decomposing rules into mission component requirements, translators to convert the component-level rules into appropriate provable forms, and techniques to automatically move the rules into run-time enforcement mechanisms (Fig 5). It is worth noting that the flow of MA requirements into the run-time environment is necessarily bi-directional, where feedback mechanisms provide planning and development guidance by indicating the ability to fulfill rule requirements at the component level. Data propagation is done using a RESTful approach in which machine-parsable textual data is used to convey rules, prover results, and (potentially) configuration updates between models. Unlike standard RESTful approaches, however, the framework supports both client-server and distributed data flow approaches.

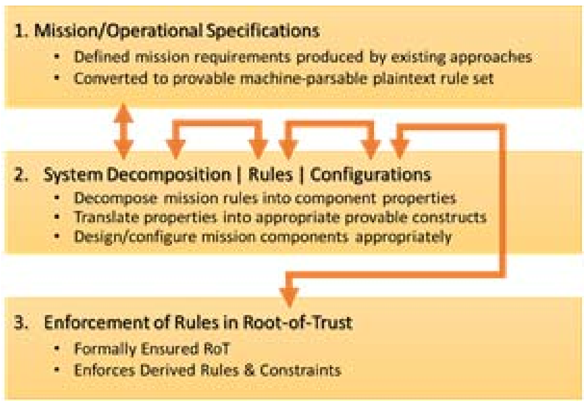
1. Mission/Operational Specifications
• Defined mission requirements produced by existing approaches
• Converted to provable machine-parsable plaintext rule set
2. System Decomposition | Rules | Configurations
• Decompose mission rules into component properties
• Translate properties into appropriate provable constructs
• Design/configure mission components appropriately
3. Enforcement of Rules in Root-of-Trust
• Formally Ensured RoT
• Enforces Derived Rules & Constraints

Fig 5. CP-SMARTS framework hierarchy.

The CP-SMARTS framework foundation borrows from (and potentially extends) the HLA-managed testbed concept to provide an integrated MA modeling framework that allows for consideration of rules regarding the safety and correctness constraints of the physical systems and their interaction with the digital components of a CPS. The approach is to extend verifiable analysis and enforcement of shared mission parameters down to implementation and up to planning by adding rule propagation and checking methods to an HLA-like API. These interface definitions provide potential for integration with existing and future mission planning (e.g. U.S. DoD *Joint Mission Planning System* - JMPS) and mission assurance (e.g. Los Alamos National Laboratory *Mission Assurance Framework* [25]) frameworks as well as low-level security-focused design and enforcement approaches such as the *Trustworthy Autonomic Interface Guardian Architecture* (TAIGA) [26].

The general approach we have settled upon for CP-SMARTS allows for the models targeted at mission planning, system development, and device tasking to be chosen for their intended low-level purpose and mixed together for higher-level analysis. The primary challenges in this approach are to decompose a system into verifiable modeled components and to define data interchanges between model types and levels such that operational assurance is maintained in a verifiable manner. With sufficient granularity in the decomposition process, the creation of formal rules can be achieved. These rules can be verified using formalisms such as propositional and temporal logic, and can provide a means by which the underlying hardware in a CPS can be constrained. By employing common formal verification techniques such as model checking, it is possible to mathematically assess whether a given platform can satisfy a certain set of properties, and by extension the corresponding MEFs. Ideally, automated information flows for property decomposition, propagation, and translation will prevent operator introduction of vulnerabilities.

B. *CP-SMARTS Future Work*

The foundation of the CP-SMARTS framework has been established, including general definitions of aspects such as rule specification, translation, propagation, proving, and enforcement. Future work will define the exact nature of these aspects. Key aspects of that work are outlined here.

The CP-SMARTS property exchange format will be a structured human-readable and machine-parsable language, allowing for a broad range of integration. A similar approach has been taken in other security-oriented solutions. For example, the Security Content Automation Protocol (SCAP) commonly used to specify software configurations for computing machines uses the XML-derived Extensible Configuration Checklist Description Format (XCCDF) as its foundation for automated machine checking [27]. While XCCDF is not necessarily well suited to formal proving, it does provide a good example of a human-readable machine-parsable format that has been readily adapted into a range of products.

The content of any given rule within the exchange format varies based upon types of rules being checked, and CP-SMARTS will define rule categories that have specific exchange formats. Handlers that allow for low-level plug-and-play modeling (e.g. swapping out a C model used in planning with FPGA hardware model for development) will produce and consume these rules for various modeling environments. Initial definition of a set of common rule categories that apply to known MA aspects in the CPS domain will allow for development of interface shims for the selected demonstration modeling environments. A key aspect of the work will be to define the process for creating new categories and developing additional interface shims that function in the broader framework. Models will consume rules, convert them to internal formats needed for proving within the specific modeling environment, and return data related to proof results to the higher-level system. Feedback provided by these results allows for iterative cross-model development to meet the security goals of the system. Our top-level reasoning engine will consume the XML-like rule and result exchanges to reason over the system as a whole. This general framework is depicted in Fig 6.

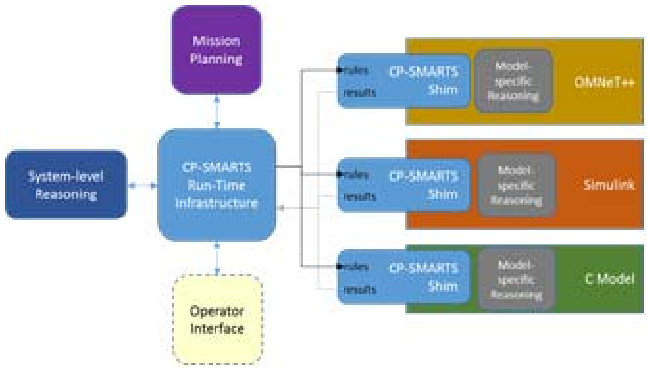

Fig 6. Framework component interaction

An important factor in enabling the CP-SMARTS reasoning framework is development of a formal means by which system decomposition can be conducted. Formal decomposition theory is a fundamental underpinning that allows for the creation of a framework by which MEFs or other mission assurance directives can be translated into properties or rules. These rules represent incontrovertible and expected behavior that must be enforced and, within the context of the CP-SMARTS system decomposition, form the basis of a series of constraints that can be used to formally verify a design. The ADT and RAG approaches will both be considered as candidates for this aspect in the second phase.

The CP-SMARTS framework leverages existing mission planning and adversary modeling platforms for initial rule construction and provides automatable decomposition and translation methods. In support of this approach, top-level property set will initially be converted into the standard framework transport format (e.g. XML) where they are expressed in a language that supports system-level reasoning (e.g. Prolog or LTL). A class-based approach to rule definitions provides for automation of class-specific rule decomposition, translation, and model checking. While a full class set was not defined as part of Phase 1, classes such as liveness, safety, consistency, confidentiality, trust are likely to emerge as part of the Phase 2 development.

IV. Conclusion

The CP-SMARTS framework is a promising solution for production of mission-aware tactical systems. Its support for assessment of risks at the juncture of Cyber-Physical, Information Assurance, and Mission Assurance domains provides a novel solution to CPS protection. The foundation of CP-SMARTS as a framework of interacting models with defined, automatable, standards for CPS Mission Assurance rule data entry and automatable model constraint propagation allows rules for cyber threat mitigation to flow between planning, development, and deployment and provides support for verification of safety and correctness rules and requirements across domains. Decomposition of high-level operational and security properties into component rules allows existing modeling and verification approaches to be both combined for system analysis and leveraged for forward design flow.

As a very simple example of the power that this can provide, consider a mission involving a UAV drone on a programmed flightpath between two waypoints. Under adversarial conditions, sensor spoofing attacks and other malicious actions targeting the drone are to be expected with the intention of damaging or otherwise disabling the system. A high-level Mission Essential Function might state that the drone must not be allowed to drop under a certain flight altitude, which constitutes one possible condition derived in partial fulfillment of the larger objective at hand – the successful completion of the UAV's transit. This MEF might be decomposed into a property stating that combined motor power should never drop below a certain level required to maintain stable flight. A formal rule can then be created to govern the appropriate system level characteristics (Duty Cycle, Electronic Speed Controller commands, etc). Formal verification techniques can then be applied at the component level to assess whether the UAV system can satisfy this property, and identify relevant changes to the UAV design in the event that it cannot.

This introduction outlined the CP-SMARTS approach, describing its foundational background, structure, and application. Future work will advance the technology by developing its components and demonstrating its efficacy in relevant environments.

References

[1] J. Newman, S. Wander, W. Vantine, and P. Benfield, "Introducing the NASA Process Based Mission Assurance Knowledge Management System (PBMA-KMS)," Joint ESA-NASA Space-Flight Safety Conference, Aug 2002.

[2] N. Suh, The Principles of Design, Oxford University Press, 1990.

[3] M. Vai, W. D, N. Evancich, L. Kwak, M. Britton, J. Foley, L. M, D. Schafer, and J. Dematteis, "Systems Design of Cybersecurity in Embedded Systems," IEEE High Performance Extreme Computing Conferece (HPEC), 2016.

[4] S. Suh, U. Tanik, U. Carbone, and J. Eroglu (Eds.), Applied Cyber-Physical Systems, Springer-Verlag New York, 2014.

[5] M. Reid, "An Evaluation of the High Level Architecture (HLA) as a Framework for NASA Modeling and Simulation," 25th NASA Software Engineering Workshop, Nov 2000.

[6] A. Giani, F. Karsai, T. Roosta, A. Shah, B. Sinopoli, and J. Wiley, "A Testbed for Secure and Robust SCADA Systems," ACM SIGBED Review - Special issue on the the 14th IEEE real-time and embedded technology and applications symposium (RTAS'08) WIP session, Vol 5 Issue 2, Article 4, July 2008.

[7] A. Davis, G. Karsai, H. Neema, A. Giani, B. Sinopoli, and R. Chabukswar, "TRUST for SCADA: A Simulation-based Experimental Platform," talk or presentation, Nov 2009.

[8] Mathworks Simulink multi-domain simulation environment, https://www.mathworks.com/products/simulink.html

[9] OMNeT++ Discrete Event Simulator, https://omnetpp.org/

[10] Delta3D game and simulation engine, http://delta3dengine.org/

[11] T. Kurtoglu and I. Tumer, "A Graph-Based Fault Identification and Propagation Framework for Functional Design of Complex Systems," Journal of Mechanical Design, May 2008.

[12] M. Boddy, T. Carpenter, H. Shackleton, and K. Nelson, "System-Level Autonomy Trust Enablers (SLATE)", U.S. Air Force T&E Days Conference, 2008.

[13] MulVAL: A logic-based enterprise network security analyzer, http://www.arguslab.org/mulval.html

[14] R. Armstrong, R. Punnoose, M. Wong, and J. Mayo, "Survey of Existing Tools for Formal Verification," Sandia Report SAND2014-20533, Sandia National Laboratories, Albuquerque, 2014.

[15] E. Clarke, W. Klieber, M. Novacek, and P. Zuliani, "Model Checking and the State Explosion Problem," Carnegie Mellon University, ETH Zurich, 2011.

[16] A. Biere, A. Cimatti, E. Clarke, O. Strichman and Y. Zhu, "Bounded Model Checking," Advances in Computer Science, Vol 58, Academic Press, 2003.

[17] K. Rozier, "Linear Temporal Logic Symbolic Model Checking," Comput. Sci. Rev. Vol 5, pp. 163-203, 2011.

[18] https://en.wikipedia.org/wiki/Kripke_structure_(model_checking)

[19] https://en.wikipedia.org/wiki/Computation_tree_logic

[20] IEEE-1800-2012, "IEEE Standard for SystemVerilog – Unified Hardware Design, Specification, and Verification Language," IEEE Standards Association, 2013.

[21] L. Humphrey, E. Wolff and U. Topcu, "Formal Specification and Synthesis of Mission Plans for Unmanned Aerial Vehicles," Association for the Advancement of Artificial Intelligence, pp. 116-121, 2014.

[22] S. Drzevitzky, U. Kastens, and M. Platzner, "Proof-carrying hardware: Towards runtime verification of reconfigurable modules," International Conference on Reconfigurable Computing and FPGAs (ReConFig), 2009.

[23] The Coq Proof Assistant - https://coq.inria.fr/

[24] D. Ricketts, M. G and L. S, "Towards Verification of Hybrid Systems in a Foundational Proof Assistant," International Conference on Formal Methods and Models for Codesign (MEMOCODE), 2015.

[25] H. Hahn, "A Mission Assurance Framework for R&D Organizations," 26th Annual INCOSE International Symposium (IS), July 2016.

[26] Z. Franklin, C. Patterson, L. Lerner, and R. Prado, "Isolating trust in an industrial control system-on-chip architecture," 7th International Symposium on Resilient Control Systems (ISRCS), 2014.

[27] XCCDF - The Extensible Configuration Checklist Description Format, https://scap.nist.gov/specifications/xccdf/