Compositional Reasoning over System Architectures with Integrated Cognitive Models

Parth Ganeriwala
Department of EECS
Florida Institute of Technology
Melbourne, FL
pganeriwala2022@my.fit.edu

Candice Chambers
Department of EECS
Florida Institute of Technology
Melbourne, FL
chambersc2017@my.fit.edu

Siddhartha Bhattacharyya
Department of EECS
Florida Institute of Technology
Melbourne, FL
sbhattacharyya@fit.edu

Isaac Amundson
Applied Research & Technology
Collins Aerospace
Cedar Rapids, IA
isaac.amundson@collins.com

Junaid Babar
Applied Research & Technology
Collins Aerospace
Cedar Rapids, IA
junaid.babar@collins.com

***Abstract*—Model-based systems engineering (MBSE) for cyber–physical systems increasingly must account for human operators who work with the growing autonomy. For safety- and security-critical applications, it is not sufficient to verify the cyber–physical platform in isolation; the system architecture must also capture assumptions about human cognition and provide evidence that operators are not put at risk. This paper presents a methodology for architecture-centric modeling and compositional verification that couples a system model with an executable cognitive model of the operator. System structure and contracts are described in the Architecture Analysis and Design Language (AADL) with AGREE assume–guarantee specifications, while operator behavior is encoded in a Soar-based cognitive model. AADL serves as the MBSE backbone, enabling engineers to represent hardware, software, communication, and human-in-the-loop components in a single, analyzable architectural model. An automated translation pipeline builds a unified model for model checking in nuXmv, enabling verification of temporal-logic properties that relate system architecture to safety of the human operator (e.g., avoidance of cybersickness, safe sensor handover). We illustrate the approach in realistic mixed-reality and increasingly autonomous system case studies, showing how architectural decisions, cognitive assumptions, and formal-methods tooling can be combined to support end-to-end assurance arguments in safety- and security-critical deployment contexts.**

***Index Terms*—MBSE, Formal Methods, Mixed-Reality Systems, High Assurance.**

I. Introduction

The design of safety-critical cyber–physical systems has shifted from isolated embedded controllers to tightly coupled human–machine teams. Mixed-reality (MR) interfaces and increasingly autonomous systems (IAS) [16], [26] are now deployed in domains such as aviation, border security, and defense, where human operators supervise, override, or collaborate with autonomous functions. In these settings, system failures are rarely purely technical: they often emerge from the interaction between the automation and the human's perception, workload, and decision strategy.

Model-based systems engineering (MBSE) provides engineers with a means to capture structure, behavior, and requirements within a single architectural model. For MR and IAS, however, architecture models that stop at the cyber–physical boundary are incomplete. The system-level safety argument depends not only on plant dynamics, software logic, and communication, but also on what the human operator can detect, how they respond, and under what conditions their performance degrades or breaks down. To reason about these questions rigorously, the architecture must be linked to an explicit model of cognition.

In this work, we utilize cognitive models to capture how humans perceive, interpret, and act in response to task and environmental stimuli. These models, grounded in empirical data from human-subjects studies, represent processes such as perception, memory, action selection, and reasoning [17]. Implementing cognitive behavior within a cognitive architecture is a sound systems and software engineering practice: the architecture encodes a general theory of cognition, much like a programming language runtime embodies a semantic model. As a result, cognitive models can reflect realistic limitations and idiosyncrasies in human behavior rather than idealized, perfectly rational agents. For safety-critical human-in-the-loop systems, this realism matters because system-level safety arguments depend on how the implemented system and the actual operator behave together, not on idealized assumptions about either.

High-assurance systems are usually tested extensively at multiple levels of integration, but even large test campaigns cannot cover all relevant combinations of inputs and events.

This effort was sponsored by the Defense Advanced Research Projects Agency (DARPA) under agreement number HR0011-24-9-0439. The views, opinions and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

This leaves residual uncertainty about whether critical requirements are truly satisfied. Model checking addresses this by exploring the state space symbolically: instead of running many test cases, the tool searches for executions that violate a formalized property or requirement [6].

In order to achieve such a high level of confidence in MR and IAS, well-formed models with a sufficiently rigorous semantics for both the system and human are required, as well as the formal guarantees that we wish to prove. Furthermore, the system and cognitive models must be analyzed together since the behavior of the system will likely influence the behavior of the human and vice versa.

Concretely, we instantiate this idea using the Architecture Analysis and Design Language (AADL) with Assume Guarantee REasoning Environment (AGREE) contracts for the system architecture, and a Soar-based cognitive model for the operator. We implement a Soar annex in the OSATE tooling, allowing cognitive productions to be embedded directly into AADL components. Additionally, we provide an automated translation from the combined AADL+Soar model to a nuXmv transition system. This enables us to check temporal logic properties that relate architectural decisions to operator outcomes.

Compared with prior work that analyzed cognitive models and architectures separately, this paper makes the following contributions:

1) It introduces a compositional workflow for co-modeling system architectures and human cognitive behavior, and for using cognitive verification results as contracts in architecture-level proofs.
2) It describes an open-source Soar annex for AADL and associated tooling that lets engineers treat cognitive productions as part of the architecture model.
3) It presents an automated translator and nuXmv-based analysis that verifies properties linking system-level guarantees to operator protection.
4) It evaluates the workflow on two representative case studies—a mixed-reality border-security scenario and an increasingly autonomous aviation scenario—highlighting how architectural design choices and cognitive assumptions jointly affect assurance.

The remainder of the paper is organized as follows. Section II presents related work on the application of rigorous approaches with formal methods. Section III describes our methodology, including the development of an advanced tool chain. Section IV illustrates the application of our methodology by: 1) analyzing cognitive attacks in MR environments by modeling user cognition under adversarial conditions, and 2) modeling a pilot's cognitive processes within an Urban Air Mobility scenario, where the pilot is treated as an autonomous agent. Section V concludes with a discussion of the accomplishments and limitations of our approach.

II. Related Work

Although model-based engineering tools have matured to enable the behavioral specification of software functions, it is difficult to apply formal methods effectively at the system level. Compounding matters, many formal methods tools have scalability limitations that prevent the analysis of complete systems. Compositional reasoning has been successfully demonstrated on architecture models in a variety of problem domains, including safety [27], security [5], machine learning [7], human-machine teaming [3], and assurance [31]. AADL [10] and the Assume Guarantee Reasoning Environment (AGREE) [32] were key enablers in all of these studies thanks to their formally defined semantics and open source implementations. AADL and AGREE are described in more detail in Section III. Verification of software architectures play a crucial role in ensuring the reliability and robustness of safety-critical autonomous systems [21], [24].

General theories of cognition, a structured framework for cognition, and logical reasoning of cognition are fundamental aspects of a cognitive architecture that supports the expression of intelligent behavior. Before selecting Soar, as the choice for modeling cognition, we surveyed several cognitive architectures, ACT-R [1], Soar [19], Mycin [29], and Clarion [28], as candidates for modeling human-automation interactions. Soar was selected based on its ability to encompass multiple memory constructs (e.g., semantic, episodic, etc.) and learning mechanisms (e.g., reinforcement, chunking, etc.). Furthermore, Soar production rules are expressed in first-order logic, which makes them amenable to verification. Finally, Soar takes the form of a programmable architecture with an embedded theory; this leads to the ability to execute Soar models on embedded system platforms, which enables the study of the design problem through the use of rapid prototyping and simulation.

Cognitive architectures have also been explored for ensuring reliable interaction between humans and autonomous systems [2], [11], [20]. Formal verification of cognitive models for assuring the human machine interface of increasingly autonomous systems is described in [2], [3], [13], [14], [22], [23], but it lacks the inclusion of the cognitive model in the architectural formalisms, which can result in inconsistency between the cognitive model and the specifications that were modeled in the architectural design.

Soar production rules commonly execute in pairs of *propose* and *apply* rules. The *propose* rule checks which Soar production rules are eligible to be executed, and the corresponding *apply* rule executes one of the eligible rules. In this research, we utilized the Soar production system framework to encode rules that describe procedures for an automated copilot in a commercial aircraft and a cognitive model of cybersickness for users of a mixed reality system. Furthermore, the first-order logic representation of the production ruleset facilitates its translation into an appropriate modeling formalism for formal verification, as detailed in the examples.

III. Methodology

In this section, we present our framework for reasoning over system architecture models that contain aspects of the environment, the cyber-physical system, and operator behavior.

Ultimately, we aim for the analysis to demonstrate that the system is correctly composed and contains the necessary components to prevent specific classes of attacks and unintended behavior related to operator cognition. This new framework, MATRICS, is shown in Fig. 1.

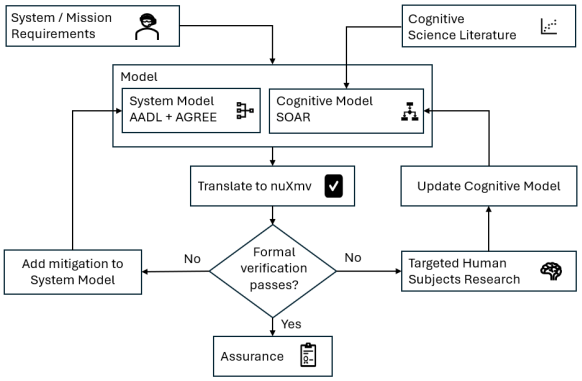

Fig. 1: Workflow for cognitive modeling and analysis.

A. Architectural Modeling

The design and analysis of cognitive guarantees become more comprehensible when relevant aspects of cognitive behavior, the environment, and the cyber-physical system can be represented within the same modeling framework. We use AADL [10], which defines language elements that have a sufficiently rigorous semantics for the specification and formal analysis of cyber-physical systems. Although human operators are not typically represented in AADL models, we do so in this work by including them as individual system subcomponents. One of AADL's more salient features is its *annex* mechanism, which enables the base grammar to be supplemented with domain-specific language (DSL) extensions. We leverage two such annexes in this work, AGREE and Soar (a newly developed annex, implementing a subset grammar of the Soar Cognitive architecture representing productions).

B. Behavioral Contracts for Architectural Components

AGREE [32] is a compositional assume-guarantee-style reasoning framework for AADL models. AGREE attempts to prove properties about an architectural component using properties associated with its subcomponents. The composition is performed in terms of assumptions and guarantees that are provided as formal contract annotations on each component in the model. Assumptions describe the expectations a component has on the environment, while guarantees describe bounds on the behavior of the component. AGREE uses k-induction model checking to prove the correctness of 1) component interfaces (i.e., the output guarantees of each component must be strong enough to satisfy the input assumptions of downstream components), and 2) component implementations (i.e., the input assumptions of a system, along with the output guarantees of its subcomponents, must be strong enough to satisfy its own output guarantees).

C. Cognitive Model within Architectural Model

Building cognitive models for physiological behaviors, we are using Soar [19] to represent the physiological preconditions that lead to cybersickness, as well as the postconditions that represent the effects of the attack. For example, a precondition modeled in Soar specifies the latency problem threshold as a trigger for cybersickness and selects the latency-check operator if the current system latency exceeds it. When this operator is applied, the postcondition of the human cognitive behavior is executed, which may result in incapacitation.

1) Integrating Soar into OSATE: To formally incorporate Soar cognitive models within AADL, we developed a Soar annex plugin for the Open Source AADL Tool Environment (OSATE), the Eclipse-based AADL reference implementation maintained by Carnegie Mellon University [9]. To achieve this, we leveraged Xtext [8], a framework for developing DSLs. Xtext enables the specification of DSL grammars that define the syntax and semantics of a language while automatically generating the corresponding parser, editor, and validation tools. By extending AADL's base grammar, we defined a Soar annex in Xtext, allowing Soar rules and productions to be embedded directly within AADL models. This integration enables seamless referencing of cognitive behaviors within system models, supporting automated analysis and verification.

2) Translating Soar to nuXmv: To enable formal analysis of embedded Soar models, we extended the translator developed in [2], [3], [13], [14], [22] to automatically parse the AADL Soar annexes into a formal nuXmv model [4]. The translation process is formalized in Algorithm 1 and aims to construct an Infinite State Machine (ISM) model compatible with nuXmv's input syntax. The input to the translator is the AADL Object ($AADLObject$) that contains the Soar annex that models cognition. The $AADLObject$ is a component type ($Ctype$) and its implementation ($CImpl$). The implementation contains a Soar Annex that specifies a Soar production rule, which is a function of a finite set of variables $v_i \in$ V, where i = 1, 2, 3, ... n, whose valuation val(V) = v_i represents the state of the system along with a finite set of well-formed formulae (WFF) $\phi = \{\phi_1, \phi_2, ...\phi_m\}$, representing the left-hand side of the Soar production rule (e.g., the preconditions), and a finite set of WFF $\psi = \{\psi_1, \psi_2, ...\psi_r\}$, representing the actions embodied by the right-hand side of the Soar production rule. The Soar production rules are represented as a tuple, $rname(V, (pre\{\phi_1, \phi_2, ...\phi_m\}, post\{\psi_1, \psi_2, ...\psi_r\}))$. The translator maps these rules into an Infinite State Machine defined as $ISM = (S, S_0, Vars, G, Act, Tr)$, where S denotes the set of system states, S_0 is the initial state, Vars represents the internal variables of the model, G defines the guard conditions derived from rule preconditions, Act contains the corresponding actions from the postconditions, and Tr encodes the transitions between states.

Algorithm 1 describes the translation in detail. In line 1, the algorithm iterates over each AADL object present in the system model. Line 2 selects the component implementation CImpl from the object. Line 3 extracts the Soar production

Algorithm 1 Generate Infinite State Machine $ISM = (S, S_0, Vars, G, Act, Tr)$ from $AADLObjects = (Ctype, CImpl(SAnnex(rname(V, (pre\{\phi_1, \phi_2, ...\phi_m\}, post\{\psi_1, \psi_2, ...\psi_r\}))))$

1: **for all** $AADLObjects$ **do**
2: $SELECT\ CImpl$
3: $SoarProd \leftarrow (EXTRACT\ CImpl(SoarAnnex))$;
4: **for all** $SubComponents \in CImpl$ **do**
5: $SubComponent.SoarProd \leftarrow (EXTRACT\ SubComponent.CImpl\ (SoarAnnex))$;
6: $APPEND$ SoarProd with $SubComponent.SoarProd$
7: **end for**
8: $TRANSLATE$ Soar2nuXmv(SoarProd);
9: **end for**

rules from the top-level component's Soar annex using a dedicated parser. If the component includes any subcomponents, as indicated in line 4, the algorithm recursively extracts the Soar rules embedded in those subcomponents' annexes (line 5), and appends them to the main rule set (line 6), preserving the hierarchical structure of the cognitive model. Finally, in line 7, the aggregated set of Soar rules is passed to the Soar2nuXmv function [3], [22], which performs the structural and syntactic transformation into the nuXmv model. The translation preserves the semantics of Soar's cognitive cycle, particularly the *propose* and *apply* phases, by encoding them as conditionally guarded state transitions within the nuXmv model. Preconditions from Soar rules are translated into logical guards that determine when a transition is enabled. Postconditions define the updates to system variables upon firing of transitions. The resulting transition system allows for temporal logic verification of cognitive properties specified in Linear Temporal Logic (LTL), enabling rigorous validation of Soar-based behavior within the AADL system context.

3) Model checking the embedded Soar model: We can systematically analyze the impact of cybersickness triggers on human operators within a formal verification framework. In essence, a Soar-based cognitive model can act as a library or repository of cybersickness behaviors. These behaviors can be translated into nuXmv [4] for formal verification of properties specified using linear temporal logic (LTL). nuXmv is a symbolic model checker for verification of finite and infinite state synchronous transition systems, extending NuSMV [4] language with support for integers and reals, and advanced SMT-based model checking techniques.

4) Integrating human behavior into system analysis: The integration of Soar-based cognitive reasoning within AADL/AGREE enables a new, innovative, automated analytical reasoning environment that combines formal reasoning on cognitive architectures with compositional, assume-guarantee-based reasoning on mixed-reality and autonomous system architectures. Guarantees in AGREE are obligations on a component's output that must hold as long as stated assumptions are valid. AGREE proves that a component's guarantees are satisfied by recursively proving the guarantees of its subcomponents.

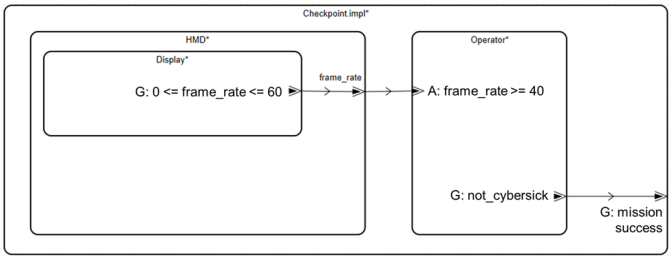

Fig. 2: AADL model containing HMD and Operator components annotated with AGREE assumptions (A) and guarantees (G).

At the lowest level of the system hierarchy (where there are no additional subcomponents), the guarantees must be verified independently (e.g., by formal analysis of a separate subsystem behavioral model or theorem proving in code). However, if the leaf component is a cognitive component, the behavior can be embedded as a Soar model. Formal analysis of a combined human-machine system can then be achieved by using the formal proof from the cognitive Soar model analysis to support the guarantees of system components up the system hierarchy. This approach is illustrated on two different use cases in the next section.

IV. Use Cases

We present two use cases to demonstrate the application of MATRICS to formally verify system models that include cognitive behavioral specifications.

A. Cybersickness Protection in Mixed-Reality Systems

We first apply our approach to a mixed-reality (MR) scenario in which a border guard uses a helmet-mounted display (HMD) in an observation tower to identify individuals carrying suspicious packages. A preliminary assessment indicates that an adversary might attempt to degrade the guard's performance by inducing cybersickness. For example, a carefully crafted software attack could overload the HMD's processing resources, increasing latency and reducing frame rate until the guard feels ill and removes the device, reducing surveillance capability.

1) Cognitive model in AADL with AGREE contracts: The initial AADL model, shown in Fig. 2, contains two main components: an `HMD` and an `Operator`. The `HMD` includes a `Display` subcomponent responsible for overlaying information into the operator's field of view. Both components are annotated with AGREE contracts. The `Display` is specified to produce a frame rate between 0 and 60 frames per second (fps), and the `Operator` component includes a guarantee stating that the operator will not become cybersick provided the frame rate stays at or above 40 fps. At the system level, we encode an informal mission requirement that the mission can only succeed if the operator does not reach a cybersick state.

Running AGREE on this initial model reveals two issues:

1) The assumption that a frame rate of at least 40 fps is sufficient to prevent cybersickness is not justified within AGREE; it depends on the behavior of the underlying cognitive model of the operator.
2) The operator's assumption that the frame rate will always be at least 40 fps is not guaranteed by the HMD component, which only promises that the frame rate lies within $[0, 60]$ fps.

These findings indicate that we must (1) analyze the Soar-based cognitive model to validate the 40 fps threshold and (2) refine the HMD architecture so that it can enforce the frame-rate assumption used in the operator's contract.

The Soar model (see Fig. 3) represents potential pathways to cybersickness that a user can exhibit. Therefore, this model demonstrates a proof of concept of human cognition when an operator's equipment is being attacked. Using existing literature, this simple cognitive model was built on three HMD hardware attributes: latency, frame rate, and optic flow [15], [30], [33]. These attributes have indicated its influence on inducing cybersickness. In virtual reality, latency refers to the delay between a user's action response, where the environment is updated to reflect the user's input [25]. In [18], leveraging virtual reality guidelines, acceptable latency was identified to be between 20-30 ms, while thresholds greater than 70 ms depict noticeable cybersickness. Frame rate is the number of frames displayed per second, which affects the fluidity and quality of motion perceived by users in the virtual environment [30]. In [12], when frame rates are between 30-90 fps, users reported significant discomfort that increased with lower frame rates especially. Optic flow is the patterns of motion in the visual field resulting from the movement of the observers or surrounding objects [15]. Zhang et. al [33] shows that when users are tasked with tracking moving objects, it results in extreme Simulator Sickness Questionnaire (SSQ) scores, and if these objects move at higher speeds, the users are more likely to be cybersick. The HMD hardware attribute thresholds used in our model are not verified, but are guided by existing literature. The main objective of this model is to demonstrate cognitive behaviors when these attributes surpass a particular threshold.

2) Property Verification: Once the models were generated in nuXmv, we began our verification process by establishing a key behavioral property: if the frame rate remains between 40 and 60 frames per second, the Operator should not experience incapacitating cybersickness. This property serves as the foundation for our analysis, ensuring that the system operates within safe physiological limits. However, verifying this condition alone is insufficient, as it risks a vacuous proof that does not confirm whether the Operator remains actively engaged in their task. To formally verify this behavioral constraint, we first encode it as a property in nuXmv to check whether the Soar model produces the correct system behavior before integrating it into the AGREE model. This is captured as the following nuXmv specification: `LTLSPEC G (cl != incapacitated-level-III);`

This specification ensures that, under normal operating conditions, cybersickness does not reach incapacitating levels when the frame rate remains within the designated range. By verifying this in nuXmv, we confirm that the Soar Annex correctly enforces this constraint at the state-transition level. Once verified, the property provides context for the AGREE model, guiding the inclusion of a monitoring mechanism that actively prevents frame rate drops below 40 fps.

To strengthen our verification, we introduce additional properties categorized into four key aspects: *ensuring system reachability, preventing unsafe states, maintaining expected operational transitions, and responding to critical events*. These properties, checked in nuXmv, validate that the system remains within valid operational states, cybersickness levels change only in response to specific violations, and the operator's task engagement is preserved.

1) *Reachability*: Ensures that the system always remains within valid states. `LTLSPEC G (state_name` $\neq$ `nil` $\wedge$ `state_name` $\neq$ `dm)`, where dm is demographic_monitor. Globally the state name is not nil as well as demographic monitor at the same time.
2) *Safety*: The system must never enter an unsafe or unintended state and must enforce constraints to prevent invalid cybersickness levels unless specific violations have occurred. Specifically, the system should ensure that an increased cybersickness level only occurs when its corresponding violation is detected. `LTLSPEC G (¬ p)` where p is a *bad state* specified as:
   - (op $\neq$ apply-latency-violated $\wedge$ cl $=$ incapacitated-level-III),
   - (op $\neq$ apply-framerate-violated $\wedge$ cl $=$ affected-level-II),
   - (op $\neq$ apply-opticflow-violated $\wedge$ cl $=$ affected-level-I),

   where op is state_operator_name, cl is state_io_cybersickness_level.
3) *Liveness*: The system should correctly follow expected transitions in normal conditions. The system should maintain constraints on cybersickness level transitions.
   - `LTLSPEC (cl` $\neq$ `level3) U (op = apply-latency-violated)`
   - `LTLSPEC G (cl = level3` $\rightarrow$ `X G (cl = level2))`

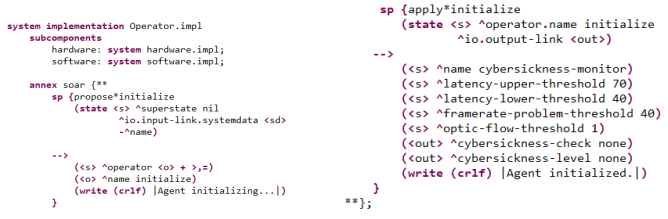

Fig. 3: Initializing Soar agent for the Operator in the AADL Soar annex.

where op is state_operator_name, cl is state_io_cybersickness-level, level2 is incapacitated-level-III and level3 is affected-level-II.
Remarks: There is a transition time between changes in cybersickness levels when optic flow violations occur in conjunction with framerate violations. As a result, the operator state may not be equal to *apply-opticflow-violated*, but the cybersickness level could still be *affected-level-I*.

4) *Response to Event Occurrence*: The system must respond appropriately to specific violations to maintain expected behavior. These are defined as: `LTLSPEC F (X (p → q))` where:
   - (s_op_name = apply-latency-violated → cl = incapacitated-level-III)
   - (s_op_name = apply-framerate-violated → cl = affected-level-II)
   - (s_op_name = apply-opticflow-violated → cl = affected-level-I)

   where s_op_name is operator name, cl is cybersickness-level.

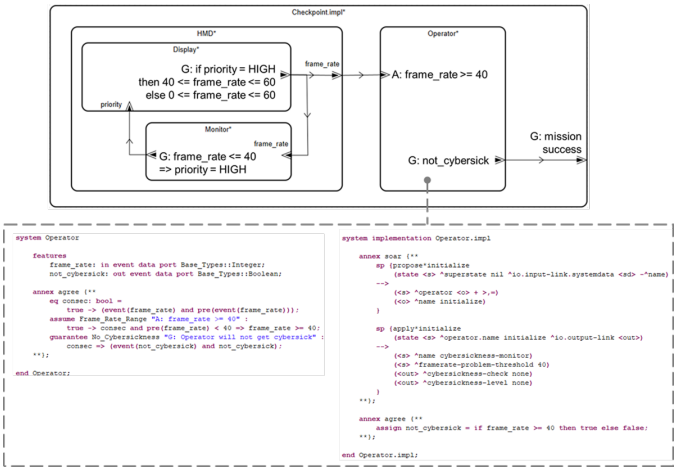

Fig. 4: Updated model with a frame rate monitor added to the HMD. Soar and AGREE annexes for the Operator are depicted below.

For the second AGREE violation, we must add a mitigation to the HMD model that will prevent the frame rate from dropping below 40 fps. In this case, we add a monitor that will increase the scheduling priority of the Display to ensure a sufficient frame rate. The updated model is shown in Fig. 4. When we run AGREE on this model, all contracts are now satisfied.

B. IAS with Autonomous Pilot Operator

Our second case study examines an increasingly autonomous aviation scenario. Here, an Autonomous Pilot Operator (APO) acts as a cognitive agent within the Assured Human-Machine Interface for Increasingly Autonomous Systems (AHMIIAS) framework [3], [13], [14], [22]. The APO monitors multiple onboard sensors, participates in contingency management, and supports coordination between the human pilot and the autonomous systems. The AHMIIAS work showed how such agents can be modeled and verified in an architecture description language, but the cognitive logic and the system architecture were developed as separate artifacts.

The APO serves as an intelligent cognitive agent that facilitates decision making based on multi-sensor input, fault detection, and dynamic role allocation between human pilots and increasingly autonomous systems (IAS). The AHMIIAS framework demonstrated a structured approach to the formal verification of cognitive agents within an architecture design language, ensuring compliance with operational safety and adaptability requirements. However, this framework required that the cognitive and architectural models be designed manually and *separately*. We address this limitation by creating the AADL Soar annex and a compositional reasoning workflow.

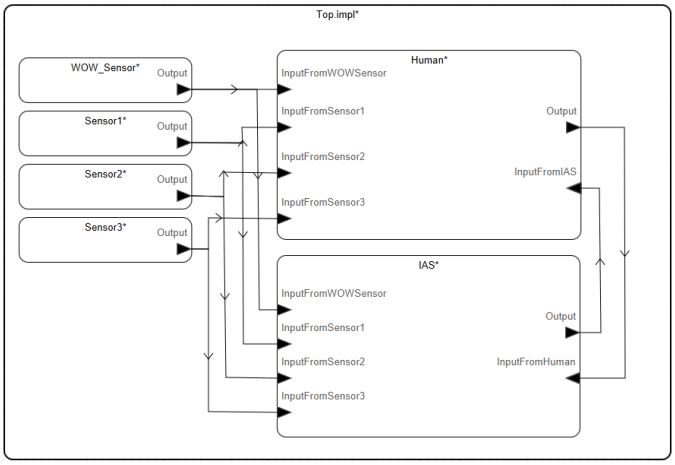

Fig. 5: AADL model of IAS with APO (human cognitive model).

1) IAS with cognitive model in AADL with AGREE contracts: To systematically validate the APO's decision-making process, the system is modeled using AADL (Fig. 5). The verification process incorporates assume-guarantee reasoning in AGREE, allowing for compositional verification of critical system functionalities, including sensor-based situational awareness, pilot interaction modeling, and failure mode verification. The APO continuously evaluates sensor reliability by identifying discrepancies in GPS, LIDAR, and IMU data streams. The system captures human input in fault acknowledgment and sensor selection, ensuring that the APO's decision-making aligns with pilot preferences.

Verification of the IAS with the APO is conducted using the AGREE framework. AGREE's assume-guarantee contracts facilitate compositional verification of critical system behaviors, ensuring adherence to operational safety constraints and effective human-machine collaboration. The APO's decision-making is governed by formalized assumptions regarding sensor reliability and human operator responses. Guarantees ensure that the APO's decision logic remains consistent with predefined safety properties, particularly in sensor failure

scenarios and human-machine interactions. Fig. 6 captures essential properties in AGREE.

The APO's decision logic is implemented in Soar as a rule-based cognitive model (Fig. 7). The agent begins by initializing flight parameters such as throttle setting, altitude, and sensor monitoring variables to represent a stable flight condition. During climb-out and cruise, Soar productions evaluate sensor data streams from GPS, LIDAR, and IMU to build an internal estimate of the aircraft state. Additional rules detect discrepancies between these sensors; when a discrepancy exceeds a defined threshold, the agent marks the corresponding sensor as suspect and proposes an operator to reassess or switch the active sensor.

Human input is modeled explicitly: when the APO reports that a sensor appears unreliable, Soar rules create an opportunity for the pilot to agree or disagree. If the pilot concurs and an alternative reliable sensor is available, the APO reallocates trust to that sensor. If the pilot does not respond within a specified time window, the agent may autonomously select a backup sensor to maintain situational awareness. Similar logic governs the landing phase, where the APO manages transitions between flight modes and can initiate recovery actions if landing conditions become unsafe. This style of modeling captures both the nominal cooperative behavior and the fallback strategies that are invoked when either sensors or pilot responses do not behave as expected.

The compositional verification framework guarantees that the sensor failure mitigation strategies and adaptive autonomy functions comply with the required safety and operational constraints. This use case presents a formally verified cognitive model of the APO, demonstrating its effectiveness in sensor-driven decision-making, adaptive pilot interaction, and safety-critical operations [3], [22].

2) Property Verification: To verify these properties, we specify key system behaviors using temporal logic. The formal specifications are categorized into four well-established formal verification areas to ensure system correctness in safety-critical domains: *reachability*, *safety (invariants)*, *liveness (ensuring normal execution)*, and *response to off-nominal situations*.

1) *Reachability*: The system must eventually satisfy a given condition, expressed as: `LTLSPEC F q` meaning that q will hold at some point in the future. Examples of q include:
   - (state_io_altitude $>$ 10000) (indicating altitude exceeds 10,000)
   - (s_op_name = transition) (indicating a transition in the operator state)

   where s_op_name is state_operator_name.
2) *Safety*: The system must never reach an unsafe state and must always maintain essential constraints. Safety properties are typically expressed as invariants to ensure that a critical condition always holds.
   - *Invariants*: Conditions that must always hold during system execution; captured by `INVARSPEC q`, where q is the invariant. Example q:
     - state_io_throttle $\leq$ 1.0 (throttle value never exceeds 1.0)
   - *Handling Off-Nominal Situations*: The system must appropriately transition during abnormal conditions. This is specified using: `LTLSPEC F(p → q)`, `LTLSPEC(p U q)`, or `LTLSPEC(p S q)`. For instance:
     - `LTLSPEC F(count_trans_lidar ≤ 5 → state_sensor = lidar)`
       Use the same Lidar sensor if error persists for less than 5 cycles.
3) *Liveness*: The system follows expected operational transitions under normal conditions.
   - *Handling Normal Operations*: The system must follow expected operational transitions, expressed similarly as `LTLSPEC F(p → q)`, `LTLSPEC (p U q)`, or `LTLSPEC (p S q)`. For example:
     - `LTLSPEC F(s_throttle < 1.00 → state_fm = horizontal)`

     where s_throttle is state_io_throttle and state_fm is state_flight-mode.
4) *Response to Event Occurrence*: When a specific event occurs, the system must respond accordingly. This is captured by: `LTLSPEC G(p → F q)` meaning that whenever p occurs, q must eventually follow. Example:
   - `LTLSPEC G((op = gps-error) → F(sensor-unreliable))`

   where op is state_operator_name, gps-error is gps-sensor-error-over-limit and sensor-unreliable is state_sensor-unreliable.

```
1  lemma "If Sensor 1 is both the active sensor and
       unreliable, it must be the
2    case that either the pilot disagreed with the IAS
         assessment
3    or the sensor just became unreliable on this timestep
4    or there was no reliable sensor available on the
         previous timestep":
5    ((IAS.Output.Active_Sensor = 1) and not
         IAS.Output.Sensor1_Reliable)
6    =>  ((Human.Output.Sensor1_Unreliable_Response =
         enum(Response, Disagree))
7        or prev(IAS.Output.Sensor1_Reliable, true)
8        or not prev(reliable_sensor_available, true));
```

Fig. 6: Human–IAS team property.

```
system implementation Human.impl
    annex soar{**

        sp {propose*initialize
            (state <s> ^superstate nil)
            (<s> ^io.input-link.flightdata <fd>)
            (<s> -^name)
            (<fd> ^throttle)

        -->
            (<s> ^operator <o> +)
            (<o> ^name initialize)
            (write (crlf) |Agent initializing...|)

        }
```

```
sp {apply*initialize
    (state <s> ^operator <o>)
    (<o> ^name initialize)
    (<s> ^io.output-link <out>)
-->
    (<s> ^name takeoff)
    (<s> ^flight-mode vertical)
    (<s> ^sensor-unreliable no)
    (<s> ^faulty-sensor none)
    (<s> ^landing no)
    (<s> ^notified-pilot no)
    (<out> ^target-altitude 5000)
    (<out> ^throttle 0.0)
    (<out> ^autoflaps off)
    (<out> ^air-brake 0.0)
    (write (crlf) |Agent initialized.|)
}
```

Fig. 7: Initialization Soar agent with default flight parameters and system states.

V. Conclusion

This paper presented a MBSE workflow for cyber–physical systems with human operators, in which the operator is modeled as a cognitive agent integrated into the system architecture. Using AADL with AGREE contracts and a Soar annex, we embed cognitive models as architectural components and automatically translate the combined model into a nuXmv transition system, enabling compositional verification where properties proved on the cognitive model support architecture-level guarantees.

We illustrated the approach on two use cases: a mixed-reality border-security scenario, where cybersickness-related behavior is linked to display properties, and an aviation scenario, where an Autonomous Pilot Operator (APO) participates in sensor selection and contingency management. In both cases, the unified treatment of system and cognitive models allows us to state and verify properties that connect architectural design choices to operator protection and mission success. Future work includes developing more compact cognitive abstractions, scaling to larger architectures, and incorporating probabilistic and performance aspects into the analysis.

References

[1] John R Anderson. *The adaptive character of thought*. Psychology Press, 2013.

[2] S. Bhattacharyya, N. Neogi, T. Eskridge, M. Carvalho, and M. Stafford. Formal assurance for cooperative intelligent agents. In *NASA Formal Methods Symposium LNCS*, volume 10811, 2018.

[3] Siddhartha Bhattacharyya, Jennifer Davis, Anubhav Gupta, Nandith Narayan, and Michael Matessa. Assuring increasingly autonomous systems in human-machine teams: An urban air mobility case study. 348:150–166, oct 2021.

[4] Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, and Stefano Tonetta. The nuxmv symbolic model checker. In *CAV*, pages 334–342, 2014.

[5] D. Cofer, I. Amundson, J. Babar, D. Hardin, K. Slind, P. Alexander, J. Hatcliff, Robby, G. Klein, C. Lewis, E. Mercer, and J. Shackleton. Cyberassured systems engineering at scale. *IEEE Security & Privacy*, 20(03):52–64, May 2022.

[6] Darren Cofer. Model checking: Cleared for take off. In *SPIN*, 2010.

[7] Darren Cofer and et. al. Run-time assurance for learning-enabled systems. In *NASA Formal Methods: 12th International Symposium, NFM 2020, Moffett Field, CA, USA, May 11–15, 2020, Proceedings*, page 361–368, Berlin, Heidelberg, 2020. Springer-Verlag.

[8] Moritz Eysholdt and Heiko Behrens. Xtext: implement your language faster than the quick and dirty way. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, pages 307–309, 2010.

[9] Peter Feiler. Open source aadl tool environment (osate). In *AADL Workshop, paris*, pages 1–40, 2004.

[10] Peter H. Feiler and David P. Gluch. *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language*. Addison-Wesley Professional, 1st edition, 2012.

[11] Angelo Ferrando, Louise A Dennis, Rafael C Cardoso, Michael Fisher, Davide Ancona, and Viviana Mascardi. Toward a holistic approach to verification and validation of autonomous cognitive systems. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 30(4):1–43, 2021.

[12] Jann Philipp Freiwald, Nicholas Katzakis, and Frank Steinicke. Camera time warp: compensating latency in video see-through head-mounted-displays for reduced cybersickness effects. In *Proceedings of the 24th ACM symposium on virtual reality software and technology*, pages 1–7, 2018.

[13] Parth Ganeriwala, Michael Matessa, Siddhartha Bhattacharyya, Randolph M. Jones, Jennifer Davis, Parneet Kaur, Simone Fulvio Rollini, and Natasha Neogi. Design and validation of learning aware hmi for learning-enabled increasingly autonomous systems. In *2025 IEEE International systems Conference (SysCon)*, pages 1–8, 2025.

[14] Parth Ganeriwala, Nandith Narayan, Randolph M Jones, Michael Matessa, Siddhartha Bhattacharyya, Jennifer Davis, Simone Fulvio Rollini, Hemant Purohit, and Natasha Neogi. Systems engineering with architecture modeling, formal verification, and human interactions for learning-enabled autonomous agent. *Systems Engineering*, 2025.

[15] James J Gibson. The perception of the visual world. 1950.

[16] Irene M Gregory and et al. Intelligent contingency management for urban air mobility. In *Dynamic Data Driven Applications Systems. DDDAS 2020. Lecture Notes in Computer Science*, 2020.

[17] ISO/IEC JTC 1. *Information technology — Vocabulary*, volume 1. International Standards Organization, May 2015.

[18] Andras Kemeny, Jean-Rémy Chardonnet, and Florent Colombet. Getting rid of cybersickness. *Virtual reality, augmented reality, and simulators*, 2020.

[19] J.E. Laird. *The SOAR Cognitive Architecture*. MIT Press, 2012.

[20] Vincent Langenfeld, Bernd Westphal, and Andreas Podelski. On formal verification of act-r architectures and models. In *CogSci*, pages 618–624, 2019.

[21] James Bret Michael, Richard Riehle, and Man-Tak Shing. The verification and validation of software architecture for systems of systems. In *2009 IEEE International Conference on System of Systems Engineering (SoSE)*, pages 1–6. IEEE, 2009.

[22] Nandith Narayan, Parth Ganeriwala, Randolph M. Jones, Michael Matessa, Siddhartha Bhattacharyya, Jennifer Davis, Hemant Purohit, and Simone Fulvio Rollini. Assuring learning-enabled increasingly autonomous systems*. In *2023 IEEE International Systems Conference (SysCon)*, pages 1–7, 2023.

[23] Natasha Neogi, Siddhartha Bhattacharyya, Daniel Griessler, Harshitha Kiran, and Marco Carvalho. Assuring intelligent systems: Contingency management for uas. *IEEE Transactions on Intelligent Transportation Systems*, 22(9):6028–6038, 2021.

[24] M. O'Connor, S. Tangirala, R. Kumar, S. Bhattacharyya, S. Sznaier, and L.E. Holloway. A bottom-up approach to verification of hybrid model-based hierarchical controllers with application to underwater vehicles. In *2006 American Control Conference*, pages 6 pp.–, 2006.

[25] Giorgos Papadakis, Katerina Mania, and Eftichios Koutroulis. A system to measure, control and minimize end-to-end head tracking latency in immersive simulations. In *Proceedings of the 10th International Conference on Virtual Reality Continuum and Its Applications in Industry*, pages 581–584, 2011.

[26] Maryam Shakeri, Abolghasem Sadeghi, and Soo-Mi Choi. Augmented reality-based border management. *Virtual Reality*, 26:1–21, 01 2022.

[27] Danielle Stewart, Jing (Janet) Liu, Darren Cofer, Mats Heimdahl, Michael W. Whalen, and Michael Peterson. Aadl-based safety analysis using formal methods applied to aircraft digital systems. *Reliability Engineering & System Safety*, 213:107649, 2021.

[28] Ron Sun. *Anatomy of Mind: Exploring Psychological Mechanisms & Processes, Clarion Cognitive Architecture*. Oxford University Press USA, 2016.

[29] William van Melle. Mycin: a knowledge-based consultation program for infectious disease diagnosis. *International Journal of Man-Machine Studies*, 10(3):313–322, 1978.

[30] Jialin Wang, Rongkai Shi, Wenxuan Zheng, Weijie Xie, Dominic Kao, and Hai-Ning Liang. Effect of frame rate on user experience, performance, and simulator sickness in virtual reality. *IEEE Transactions on Visualization and Computer Graphics*, 29(5):2478–2488, 2023.

[31] Timothy E. Wang, Chanwook Oh, Matthew Low, Isaac Amundson, Zamira Daw, Alessandro Pinto, Massimiliano L. Chiodo, Guoqiang Wang, Saqib Hasan, Ryan Melville, and Pierluigi Nuzzo. Computer-aided generation of assurance cases. In *10th International Workshop on Next Generation of System Assurance Approaches for Critical Systems (SASSUR)*, page 135–148, Berlin, Heidelberg, 2023. Springer-Verlag.

[32] M. W. Whalen, A. Gacek, D. Cofer, A. Murugesan, M. P. E. Heimdahl, and S. Rayadurgam. Your "what" is my "how": Iteration and hierarchy in system design. *IEEE Software*, 30(2):54–60, 2013.

[33] Jianing Zhang, Xiaoping Che, Enyao Chang, Chenxin Qu, Xiaofei Di, Haiming Liu, and Jingxin Su. How different text display patterns affect cybersickness in augmented reality. *Scientific Reports*, 14(1):11693, 2024.