From Component-Based to Game-Theoretic Models: Analyzing Security as Collaborative Behaviors in Cyber-Physical Systems

Abstract. Modern cyber-physical systems (CPS) pose significant challenges in integrating diverse communication protocols. While dedicated modeling formalisms based on component-port-connector paradigms can effectively represent these systems, they cannot often capture competitive or collaborative behaviors, a challenging architectural concern, particularly within attacks. We propose a novel modeling approach that leverages stochastic multi-player games (CSGs) with reward structures for both verification and strategy synthesis of system architectures to help in this respect. To express properties within these models, we employ rPATL (probabilistic alternating-time temporal logic with rewards). This allows for quantitative reasoning about the ability of player coalitions to reduce risks or attack success based on the probability of attacks or reward/cost measures. We demonstrate the performance, scalability, and applicability of our approach through the evaluation of drone systems. This showcases our approach's capability to analyze systems exhibiting probabilistic, cooperative, and competitive interactions between concurrent components.

Keywords: Software architecture· Components· Connectors· Game Theory

1 Introduction

Software architecture design is witnessing a trend towards multi-paradigm approaches, integrating diverse methodologies [27]. This includes Component-Based Development (CBD) [13,34], Model-Driven Engineering (MDE) [32], and formal methods [30]. This integration has led to the proposal of numerous description languages and formalism for modeling complex distributed systems [8]. However, standard specifications like UML-like [29], CCM-like [28], and ADL-like [31] may not fully address reliability and security concerns during design, particularly regarding communication channels. A key limitation lies in the potential lack of a robust "connector" concept within these languages. This necessitates the exploration of alternative methods to enhance the trustworthiness of communication in distributed systems at the design phase.

Formal verification using stochastic games [21] allows for the generation of quantitative correctness assertions about a system's behavior (e.g. "The object recognition system can correctly identify pedestrians with a probability of at least 95%, even in challenging lighting conditions".), where the required behavioral properties are expressed in quantitative extensions of temporal logic. The problem of strategy synthesis constructs an optimal strategy for a player, or coalition of players, to ensure a desired outcome (property) is achieved.

Concurrent stochastic multi-player games (CSGs) permit players to choose their actions concurrently in each state of the model. This approach captures the true essence of concurrent interaction, where agents make independent choices simultaneously without perfect knowledge of others' actions. However, although algorithms for verification and strategy synthesis of CSGs have been implemented in PRISM-games[23], their application to software architecture is lacking [9].

This paper presents a novel approach for modeling and verifying software architecture using concurrent stochastic multi-player games (CSGs), with a particular focus on incorporating security considerations. We leverage the rPATL logic for expressing properties adapted to CSG models. Subsequently, we employ the PRISM-games model checker, which implements a comprehensive suite of algorithms for constructing and verifying CSG models. Finally, to evaluate our methodology's performance, scalability, and applicability, we present a case study from a European project focusing on drone-assisted crane orchestration, where the system is vulnerable to malicious attacks. We demonstrate how attackers can exploit communication channels (connectors) as entry points to manipulate lifting levels.

Outline The remainder of this paper is organized as follows. Section 2 provides background material essential for understanding the CSGs formalism. Section 3 presents our software architecture modeling approach using the PRISM language with well-defined semantic rules. Section 4 focuses on attack scenarios relevant to the software architecture. Section 5 details a comprehensive set of experiments evaluating attacks within the context of drone-assisted crane orchestration. Section 6 discusses relevant research in this field. Finally, Section 7 concludes the paper and outlines potential directions for future work.

2 Preliminaries

Concurrent stochastic games (CSGs) [23] are based on the idea that players make choices concurrently in each state and then transition simultaneously. In CSGs, each player controls one or more modules, and the actions that label commands within a player's modules must only be used by that specific player.

To express the coalition game, we rely on PRISM [20]. The PRISM model is composed of a set of modules that can synchronize. A set of variables and commands characterizes each module. The variable's valuations represent the state of the module. A set of commands is used to describe the behavior of each module (i.e., transitions). A command takes the form: $[a_1, \ldots, a_n]g \to \lambda_1 : u_1 + \ldots + \lambda_n : u_n$ or, $[a_1, \ldots, a_n]g \to u$, which means, for actions " $a_1, \ldots, a_n$ " if the guard " g " is true, then, an update " u_i " is enabled with a probability " λ_i ". A guard is a logical proposition consisting of variable evaluation and propositional logic operators. The update " u_i " is an evaluation of variables expressed as a conjunction of assignments: $v_i' = val_i + \ldots + v_n' = val_n$ where " v_i " are local variables and val_i are values evaluated via expressions denoted by " θ " such that $\theta : V \to \mathbb{D}$. CSGs are augmented with reward structures [23], which assign each state and action tuple to a real value accumulated when the action tuple is selected.

The properties related to CSGs are expressed in the temporal logic rPATL [10] (reward Probabilistic Alternating Temporal Logic). The property grammar is based on CTL [5] extended with coalition operator $\langle\langle\mathscr{C}\rangle\rangle$ of ATL [3] and probabilistic operator $\mathtt{P}$ of PCTL [16]. For instance, for the following property expressed in natural language: "*Players 1 and 2 have a strategy to ensure that the probability of system reset occurring within 100 steps is less than 0.001, regardless of the strategies of other players*" is expressed in rPATL as: $\langle\langle 1,2\rangle\rangle \mathtt{P}_{<0.001}[\mathtt{F}^{\leq 100}\mathtt{reset}]$. Here, " $\mathtt{reset}$ " is the label that refers to the system states. Concerning rewards structure, the property expressed in natural language: "*What is the maximum commutative reward r within 100 steps to reach "* $\mathtt{reboot}$ *" for both Players 1 and 2 for a selected strategy ?*" is expressed in rPATL as $\langle\langle 1,2\rangle\rangle \mathtt{R}_{\mathtt{max}=?}[\mathtt{C}^{\leq 100}]$

3 A game-theoretic model of the software architecture

This section focuses on defining the semantics of the component-port-connector (CPC) paradigm within PRISM, a language well-suited for expressing CPC concepts as modules. We will start by establishing formal definitions for components, connectors, and the overall CPC architecture. Following that, we delve into the specific definition of a stochastic component.

Definition 1 (Component). A component in CPC is a PRISM module $\mathscr{P}_C = \langle s_0, S, P, C, \vartheta, L\rangle$ labeled with ports, where:

- s_0 is the initial state,
- $S = \{s_1, \ldots, s_k\}$ is a set of states,
- P is a set of ports referred to as PRISM actions in synchronized components,
- ϑ is a set of PRISM variables including state variables S ,
- $Cm : S \times P \times Const(\vartheta) \longrightarrow Dist(S)$ is a probabilistic PRISM command function assigning for each $s \in S$ and $\alpha \in P$ a probabilistic distribution $\mu \in Dist(S)$, θ is a set of valuations on a set of PRISM model variables ϑ . Formally, we distinguish two types of commands :

  Push: $s \xhookrightarrow{g:\alpha!v} s'$ The module sends a value v , and

  Pull: $s \xhookrightarrow{g:\alpha!v} s'$ The mosule receives a value v .
- $L : S \longrightarrow 2^{AP}$ is a labeling function that assigns each state $s \in S$ to a set of atomic propositions taken from the set of atomic propositions (AP).

The semantics of connectors are derived from the semantics of components, with the connectors themselves not being considered as *players*. The connectors connect the components based on the component's ports as we use the operator γ . Expressing algebraically, the connections are modeled as composition such that $\gamma_{p_1,\ldots,p_n}(\mathscr{P}_{C_1}, \ldots, \mathscr{P}_{C_n})$ is determined by the connectors according to the operational semantics rule *Push/Pull*.

$$\frac{[\![\mathscr{P}_{C_1}]\!] = s_i \xhookrightarrow{g_1:\alpha!v} s_i' \wedge \theta \models g_1 \wedge [\![\mathscr{P}_B]\!] = s_j \xhookrightarrow{g_1:\langle\alpha,\beta\rangle} s_j' \wedge \theta \models g_2 \wedge \quad [\![\mathscr{P}_{C_2}]\!] = s_k \xhookrightarrow{g_3:\beta?v'} s_k' \wedge \theta \models g_3}{\langle s_i, \ldots, s_j, \ldots, s_k, \theta\rangle \xrightarrow{\langle\alpha,\beta\rangle} \langle s_i', \ldots, s_j', \ldots, s_k',\ \theta'\rangle} \quad (\textit{Push/Pull})$$

$$\theta' = \theta[v' = v]$$

So, Connectors are characterized by a set of ports that label the commands within them. The formal definition of a connector is presented below:

Definition 2 (Connector). A connector $\mathscr{P}_B$ is a PRISM module such that $\mathscr{P}_B = \langle s_0, S, P_1 \times \ldots \times P_n, C, \vartheta, L\rangle$, where:

- s_0 is the initial state,
- $S = \{s_1, \ldots, s_k\}$ is a set of states,
- ϑ is aset of PRISM variables including state variables S ,
- $Cm : S \times P_1 \times \ldots \times P_n \times Const(\vartheta) \longrightarrow Dist(S)$ is a probabilistic PRISM command assigning for each $s \in S$ and a set of ports $\alpha, \ldots, \beta \in P$ a probabilistic distribution $\mu \in Dist(S)$, the behavior of a set of commands follows the formal command: $s \xhookrightarrow{g:\langle\alpha,\beta\rangle} s'$, and
- $L : S \longrightarrow 2^{AP}$ is a labeling function that assigns each state $s \in S$ to a set of atomic propositions taken from the set of atomic propositions (AP).

4 An attack scenario within the software architecture

This section explores a common threat to message integrity: tampering. As defined in [33], tampering refers to the unauthorized alteration of messages during transmission between communicating entities. In the context of communication style, this can manifest as a modification of message content (denoted as m) during data transfer.

To better express successful or failed modifications, we use the operational semantics rules presented in *Success* and *Failure* that inherit the operational semantics rule *Push/Pull*. To illustrate these rules, we consider three players: $\mathscr{P}_{C1}, \mathscr{P}_{C2}, and \mathscr{P}_{Att}$ who send data through the connector $\mathscr{P}_B$ α to perform a push (!) action (played by s_i), and one receiver player who is in pull mode (?).

According to operational semantics rule *Success*, the attacker identified by the command in $\mathscr{P}_{Att}$ attempts to tamper with the message x at rate λ . The rule specifies that the attacker successfully modifies the received variable v' to x , indicated by the success variable $win = 1$. However, operational semantics rule *Failure* captures the scenario where the attack fails with rate $1 - \lambda$. This rule shows that $\mathscr{P}_{C1}$ successfully transmits its local data to $\mathscr{P}_{C2}$ through $\mathscr{P}_B$.

$$\frac{\begin{array}{c}[\![\mathscr{P}_{C_1}]\!] = s_i \xhookrightarrow{g_1:\alpha!v} s_i' \wedge \theta \models g_1 \wedge [\![\mathscr{P}_B]\!] = s_j \xhookrightarrow{g_2:\langle\alpha,\beta,\omega\rangle}_{\lambda} s_j' \wedge \theta \models g_2 \wedge \\ [\![\mathscr{P}_{C_2}]\!] = s_{lk} \xhookrightarrow{g_3:\beta?v'} s_k' \wedge \theta \models g_3 \wedge [\![\mathscr{P}_{C_{Att}}]\!] = s_l \xhookrightarrow{g_4:\omega?x} s_l' \wedge \theta \models g_4\end{array}}{\langle s_i,\dots,s_j,\dots,s_k,\theta\rangle \xrightarrow{\langle\alpha,\beta,\omega\rangle} \langle s_i',\dots,s_j',\dots,s_k',\ \theta'\rangle} \quad (\textit{Success})$$

$$\theta' = \theta[v' = x, win = 1]$$

$$\frac{\begin{array}{c}[\![\mathscr{P}_{C_1}]\!] = s_i \xhookrightarrow{g_1:\alpha!v} s_i' \wedge \theta \models g_1 \wedge [\![\mathscr{P}_B]\!] = s_j \xhookrightarrow{g_1:\langle\alpha,\beta,\omega\rangle}_{1-\lambda} s_j' \wedge \theta \models g_2 \wedge \\ [\![\mathscr{P}_{C_2}]\!] = s_{lk} \xhookrightarrow{g_3:\beta?v'} s_k' \wedge \theta \models g_3 \wedge [\![\mathscr{P}_{C_{Att}}]\!] = s_l \xhookrightarrow{g_4:\omega?x} s_l' \wedge \theta \models g_4\end{array}}{\langle s_i,\dots,s_j,\dots,s_k,\theta\rangle \xrightarrow{\langle\alpha,\beta,\omega\rangle} \langle s_i',\dots,s_j',\dots,s_k',\ \theta'\rangle} \quad (\textit{Failure})$$

$$\theta' = \theta[v' = v, win = 0]$$

5 Experiments

This scenario, illustrated in Fig. 1 [12], involves a drone collaborating with cranes to lift a platform. To maintain stability, the drone continuously transmits lifting measurements to the cranes at each time instant (t). The platform has two markers that the drone scans to determine their distance from the ground. This calculated lifting distance is then sent as a payload message to the cranes until the platform reaches its optimal position. However, as discussed previously, drones are vulnerable to attacks that could manipulate the lifting data. In case of our demonstration the attacks considers the communication channel as the vulnerable access point.

Experimental setup. Within the set of functional and security properties, we have encoded properties in rPATL formalism. PRISM-games model checker 3.0 [22] is utilized to perform verification. These experiments were conducted on a Ubuntu-I7 system equipped with 32GB RAM. Multiple engines can be selected (refer to documentation [14]) offering performance benefits for specific model structures. In addition, we have implemented the scenarios outlined in [**?**] to accurately model attack frequencies.

Artefacts. The source code for the experiments described in this section is publicly available on a GitHub repository[1]. The website provides comprehensive instructions on how to replicate the experiments.

5.1 Modeling

The composed model effectively captures the interplay between the drone, attacker, and cranes. A dedicated module, “ConnectorDroneCrane_1” specifically

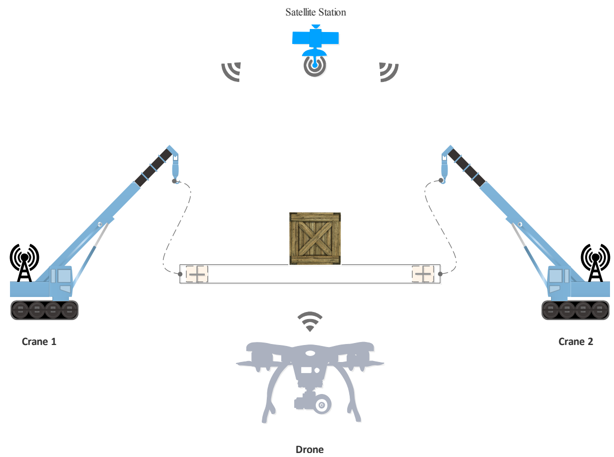

Fig. 1: Drone-assisted Lifting Orchestration [12].

Listing 1.1: Human Intervention in PRISM Code (non-player module)

```
1   module ConnectorDroneCrane_1
2   win : [0..1] init -1;
3   data : [-1..1] init -1;
4   //Probabilistic attack with frequency FREQ
5   [writeDrone, readCrane_1 , readCrane_2 ,writeAttack_1]   true  -> (1-FREQ)
         :(data'=Drone_Lift) & (win'=0) +(FREQ):(data'=Attack_1_Lift)& (win'=1);
6   endmodule
```

tracks the attack's effectiveness. This model, detailed in code snippet Listing 1.1, uses commands represented by lists of actions reflecting the choices available to each player. For instance, the command on line 5 depicts probabilistically successful attacks based on their frequency "FREQ" while the actions belonging to components are available. The component can be enhanced to record more actions, allowing for a more comprehensive observation of phenomena. The order of these action lists is irrelevant as the actions of each player are independent.

5.2 Properties as game goals

We enhance the model by incorporating an integer constant as in [**?**] and a module (see Listing 1.2) to keep track of the number of rounds. In this case, as the commands are unaffected by the players' choices, they are considered unlabeled with empty action. Consequently, these commands are executed regardless of the actions taken by the players. Furthermore, the module is deterministic due to the disjoint guards present in both commands.

Listing 1.2: Rounds module to Represent Multiple Interventions.

```
const k; // number of rounds
// Module to count rounds
module rounds
  rounds : [0..k+1];
  // Transition: increment rounds if not at max
  [] rounds <= k -> (rounds' = rounds + 1);
  // Stay at max rounds
  [] rounds = k+1 -> true;
endmodule
```

Additionally, we introduce two reward structure in Listing 1.3 that aligns with each player's utility in the model. In this case, a reward of "cost" can be assigned for winning, while "-cost" can represent losing scenarios. To implement this, action lists can once again be utilized to capture the different choices made by the players. To enable the inclusion of additional attackers in the future, we have defined two essential formulas in lines 2 and 3.

Listing 1.3: Reward structure to Express the Utility Function

```
1  //PRISM formula to generalize when multiple attackers
2  formula win1 = win=1 ;
3  formula win0 = win=0 ;
4
5  const double cost; //The reward/cost of winning/losing
6  // reward structure for successful attacks
7  rewards "utility1"
8          win1 :  cost; //successful Attacks
9  endrewards
10 // reward structure for non-successful attacks
11 rewards "utility2"
12         win0 :  cost;// non successful Attacks
13 endrewards
```

Considering the updated model, we can express the properties related to the modeled system in natural language:

PRO1: *What is the minimum probability that a* ***drone can successfully broadcast the lifting level*** *regarding attacks in at least one of the k rounds?*.
PRO2: *What is the minimum probability that* ***the attacks are successfully realized*** *regarding drones in the broadcasting mode in at least one of the k rounds?*.
PRO3: *What is the expected cumulative utility of* ***successful drones brodcast*** *over k rounds?*.
PRO4: *What is the expected cumulative utility of* ***successful attack realizations*** *over k rounds?*.

We can express the properties that are subject to analysis, including the probability of winning at least one round out of k rounds and the expected cumulative utility over k rounds using rPATL, each component is identified as a player, except for the connector. For example, p1 refers to the drone, p2 refers to crane1, p3 refers to crane2, and p4 refers to the attacker:

Properties in rPATL

$$<<\texttt{p1},\texttt{p2},\texttt{p3}>> \texttt{Pmin} =?[\texttt{F}\ (\texttt{win} = \texttt{0}\ \&\ \texttt{rounds} <= \texttt{k})],\ \texttt{k} = \texttt{1} : \texttt{30} : \texttt{1} \quad \text{(PRO1)}$$

$$<<\texttt{p2},\texttt{p3},\texttt{p4}>> \texttt{Pmin} =?[\texttt{F}\ (\texttt{win} = \texttt{1}\ \&\ \texttt{rounds} <= \texttt{k})],\ \texttt{k} = \texttt{1} : \texttt{30} : \texttt{1} \quad \text{(PRO2)}$$

$$<<\texttt{p1},\texttt{p2},\texttt{p3}>> \texttt{R}\{\texttt{"utility1"}\}\texttt{max} =?[\texttt{F}\ \texttt{rounds} = \texttt{k}],\ \texttt{k} = \texttt{1} : \texttt{30} : \texttt{1} \quad \text{(PRO3)}$$

$$<<\texttt{p2},\texttt{p3},\texttt{p4}>> \texttt{R}\{\texttt{"utility2"}\}\texttt{max} =?[\texttt{F}\ \texttt{rounds} = \texttt{k}],\ \texttt{k} = \texttt{1} : \texttt{30} : \texttt{1} \quad \text{(PRO4)}$$

5.3 Analyses

The verification results of (PRO1) and (PRO2) are depicted in Fig. 2 and Fig. 3, respectively. In Fig. 2, the drone p1 achieves a successful broadcast rate exceeding 80% in scenario 1 (attack frequency = 0.2) after only 2 rounds. Conversely, it takes 10 rounds to reach an 80% success rate with a higher attack frequency of 0.8. Similarly, Fig. 3 shows that the attacker p4 can tamper the connector with a success rate exceeding 80% in just 2 rounds for a higher attack frequency (scenario 2). However, at a lower attack frequency, it takes 10 rounds to reach the same success rate.

Meanwhile, the reward queries acknowledge that the frequency of attacks significantly impacts winning the game. For example, in Fig. 4, when the attack frequency is low, the drone can maintain the lifting level even though attacks are present. Furthermore, we observe that the winning time increases as the number of rounds increases. This suggests that the drone is continuously learning to reach maximum equilibrium in the game, making the attacker less impactful. In contrast, Fig. 5, where the query PRO4 focuses on the attacker, shows a chance for attackers to progress in tampering as the number of rounds increases. This implies that the attacker can continuously learn to overcome the drone in the competition.

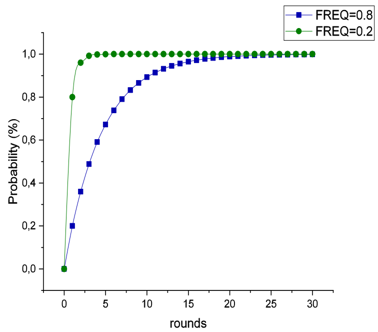

Fig. 2: Verif. PRO1.

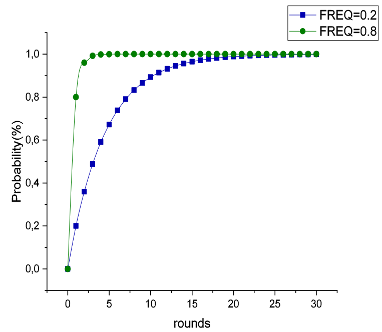

Fig. 3: Verif. PRO2.

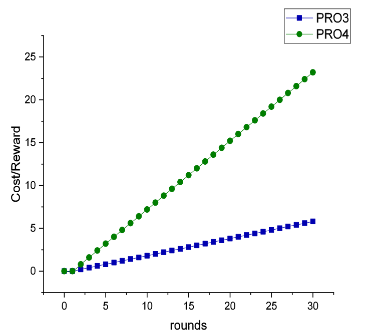

Fig. 4: Verif. PRO3 and PRO4 in Scenario1.

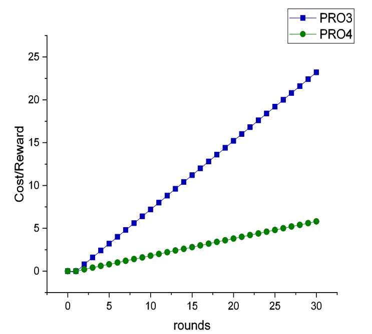

Fig. 5: Verif. PRO3 and PRO4 in Scenario2.

5.4 Computational aspects of model scalability

Since model checking takes longer with larger state spaces, it's valuable to evaluate the scalability of the model regarding the number of attackers. In our experiments, we considered scenarios with two, four and fsix attackers. Each attacker can access the connector to tamper with data and potentially cause faults during the lifting operation.

NB. Attackers	States	Transitions	Model Construction	Model Checking
1	123	246	0.004 seconds	0.005 seconds
2	123	246	0.08 seconds	0.004 seconds
4	123	246	0.007 seconds	0.05 seconds
6	123	246	0.01 seconds	0.08 seconds

Table 1: Computational Aspects of Models Scalability.

In our study, we consider the verification cost and model construction function, which allow us to evaluate the model (as described in [35]). PRISM model checking provides information related to both parameters. Therefore, Table 1 portrays the results relative to both computations. The computation shows stable model construction times between 0.004 and 0.01 seconds, and model checking times ranging from 0.005 to 0.08 seconds. This efficiency is attributed to the single variable used for configuring broadcasting, requiring only one push and one pull operation. However, to model asynchronous communication between components, we extend the model to include two attackers and introduce a buffer that increments as data is pushed. The verification focuses on PRO4 and the results are presented in Table 2.

Queue size	States	Transitions	Model Construction	Model Checking
300	1201	246	0.028 seconds	1.415 seconds
600	2401	4796	0.039 seconds	5.607 seconds
1200	4801	9596	0.068 seconds	23.501 seconds
2400	9601	19196	0.116 seconds	103.183 seconds

Table 2: Computational Aspects of Models Scalability with Asyn.Mode.

Model checking time increases with increasing buffer size, ranging from 1.45 to 103.183 seconds. However, model construction remains low, between 0.028 and 0.116 seconds. This is because model-checking algorithms [22] require traversing the state space represented by the matrix diagram, which grows larger with increasing buffer size.

5.5 Threats to validity

Our current model does not fully account for the impact of drone-assisted lifting hardware on system verification. Specifically, the reliability and availability of the hardware component haven't been incorporated. This omission is significant because component degradation can directly affect the overall system's reliability. Additionally, the model doesn't address strategies for replacing hardware components to maximize lifespan. Including these aspects would provide a more comprehensive understanding of the system and its robustness.

6 Related work

Allen's work [2] introduced a comprehensive methodology for specifying Component-Based Architectures (CBA) using formal semantics to define architectural connectors. Formal languages play a significant role in software engineering for designing and analyzing systems. Examples include process algebra (like CCS [25], CSP [17], ACP [7], and π-calculus [26], LOTO [15]) and automata (like timed automata [4], I/O automata [19]). In our work, we focus on Probabilistic Automata, an extension of classical transition systems, for designing component behaviors. Transition systems (TS) are a widely used formalism to describe system behavior [5]. These systems can be visualized as directed graphs, where nodes represent states and edges represent transitions. Several component models like BIP [6], AUTOFOCUS AF3 [18], and UML (through activity and state machine diagrams) [29].

The formalism used by each component model in TS remains consistent across different component models. The exception lies at the connector level, where variations arise. In the BIP component model, connectors act as stateless entities performing a "gluing" operation in a publish-subscribe pattern when ports are stimulated and available [8]. Conversely, UML and AUTOSAR AF3 abstract connectors, delegate complex computations to components rather than connectors themselves. This approach can hinder the modeling of intricate systems where connectors manage concurrency access and stochastic behavior. In such scenarios, concurrent and stateful connectors are often necessary. One promising formalism for modeling such systems is Concurrent Stochastic Multi-Player Games (CSGs). CSGs leverage the probabilistic automata model, allowing components to concurrently select their actions within each state. This can be interpreted as components operating concurrently, making their choices without prior knowledge of what other components' actions.

To ensure the correctness of software architecture, component models often rely on specialized verification tools. These tools consume the component model description and translate it into a format compatible with the chosen verification engine. For example, the BIP component model leverages the BIP statistical model checker [24] for analysis, while AUTOFOCUS AF3 utilizes NuSMV [11]. The verification process involves assessing various properties of the system, such as functional and non-functional properties like performance or security. These properties are typically expressed using formalisms designed for specific use cases. In the case of Concurrent Stochastic Multi-Player Games (CSGs), verification of probabilistic properties expressed in temporal logic is supported by tools like PRISM-games checker [22], which implements specialized algorithms for this purpose.

This paper tackles a crucial shortcoming in modeling secure component-based software architectures: the absence of robust semantic support for stateful connectors as collaborative interaction points. Stateful connectors facilitate concurrent and cooperative interactions between components. Existing research on component-based systems has largely overlooked this gap, particularly when it comes to security considerations.

7 Conclusion

We have demonstrated how formal methods, specifically probabilistic model checking with games semantics in PRISM games, can model and verify the performance of the collaborative and concurrent connector access for synchronizing crane lifting operations based on orders originating from drones. Formal verification can be a productive design tool for software architecture and communication protocols. By formally modeling the protocol, designers can assess various parameters and options before turning to simulation.

Furthermore, we investigated how connector design choices can impact scalability in relation to design requirements. Implementing a connector with buffers for concurrent data exchange in push and pull modes can lead to state space explosion. In such scenarios, abstraction techniques can be beneficial by reducing computational overhead. However, this abstraction might come at the cost of losing details about the model, such as the buffer structure.

We propose extending our approach beyond software architecture specification and functional verification. From a design perspective, we propose the development of a dedicated grammar specifically tailored to model competitive systems. This grammar would bridge the gap between engineer and researcher viewpoints by offering a high level of abstraction that aligns with the engineer's perspective while avoiding excessive detail and complex mathematical elements that may hinder adoption. From a hardware perspective, our goal is to encompass non-functional properties, specifically clock de-synchronization between collaborating drones during lifting tasks. This integration will enable us to model environmental factors and hardware degradation that can impact message quality. By incorporating these considerations, we aim to generalize tampering operations on connectors to encompass a wider range of threats, including both human-induced and environmental.

Data Availability

The artifacts related to this article are available on the GitHub repository [1] . The repository respects the anonymity required for the double-blind review process, as the content does not contain any references to the authors' names or institutions.

References

1. **. Paper Artefacts Sources. https://blindreviewforwcj.github.io/ecsa2024.html.
2. Robert Allen and David Garlan. A Formal Basis for Architectural Connection. *ACM Trans. Softw. Eng. Methodol.*, 6(3):213–249, 1997.
3. Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, sep 2002.
4. Rajeev Alur and David L. Dill. A Theory of Timed Automata. *heoretical Computer Science*, 12(6):183–235, 1994.
5. Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. The MIT Press. OCLC: ocn171152628.
6. Ananda Basu, Saddek Bensalem, Marius Bozga, Jacques Combaz, Mohamad Jaber, Thanh-Hung Nguyen, and Joseph Sifakis. Rigorous component-based system design using the BIP framework. 28(3):41–48.
7. J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Control*, 60(1-3):109–137, 1984.
8. Simon Bliudze and Joseph Sifakis. The algebra of connectors—structuring interaction in BIP. 57(10):1315–1330.

9. PRISM Model Checker. Prism-games - publications.
10. Taolue Chen, Vojtěch Forejt, Marta Kwiatkowska, David Parker, and Aistis Simaitis. Automatic verification of competitive stochastic systems. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 7214, pages 315–330. Springer Berlin Heidelberg.
11. Alessandro Cimatti, Edmund M. Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. Nusmv 2: An open-source tool for symbolic model checking. In *Proceedings of the 14th International Conference on Computer Aided Verification*, CAV '02, page 359–364, Berlin, Heidelberg, 2002. Springer-Verlag.
12. CPS4EU Project. CPS Tool Evaluation. D5.6. https://cps4eu.eu/wp-content/uploads/2022/11/CPS4EU_D5.6_CPS-Tool_Evaluation_V1.0.pdf, 2022. [Online; accessed 19-March-2023].
13. Ivica Crnkovic. Component-based software engineering for embedded systems. In *27th International Conference on Software Engineering*, ICSE '05, pages 712–713. ACM, 2005.
14. PRISM Development Team (eds.). Prism manual. https://www.prismmodelchecker.org/manual/ConfiguringPRISM/ComputationEngines. Accessed: March 27, 2024.
15. Andrea Ferrara. Web services: A process algebra approach. In M. Aiello, M. Aoyama, F. Curbera, and M.-P. Papazoglou, editors, *2nd international conference on Service oriented computing - ICSOC '04*, pages 242–251, New York, NY, USA, 2004. ACM Press.
16. Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. 6(5):512–535.
17. CAR. Hoare. Communicating Sequential Processes. *Commun. ACM*, 21(8):666–677, 1978.
18. Sudeep Kanav and Vincent Aravantinos. Modular transformation from af3 to nuxmv. In *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, 2017.
19. Dilsun K. Kaynar, Nancy Lynch, Roberto Segala, and Frits Vaandrager. The theory of timed i/o automata, 2011.
20. Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.
21. Marta Kwiatkowska, Gethin Norman, David Parker, and Gabriel Santos. Equilibria-based probabilistic model checking for concurrent stochastic games. In Maurice H. ter Beek, Annabelle McIver, and José N. Oliveira, editors, *Formal Methods – The Next 30 Years*, pages 298–315, Cham, 2019. Springer International Publishing.
22. Marta Kwiatkowska, Gethin Norman, David Parker, and Gabriel Santos. Prism-games 3.0: Stochastic game verification with concurrency, equilibria and time. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification*, pages 475–487, Cham, 2020. Springer International Publishing.
23. Marta Kwiatkowska, Gethin Norman, David Parker, and Gabriel Santos. Automatic verification of concurrent stochastic systems. *Formal Methods in System Design*, 58(1):188–250, Oct 2021.
24. Braham Lotfi Mediouni, Ayoub Nouri, Marius Bozga, Mahieddine Dellabani, Axel Legay, and Saddek Bensalem. SBIP 2.0: Statistical model checking stochastic real-time systems. In *Automated Technology for Verification and Analysis*, volume 11138, pages 536–542. Springer International Publishing.
25. Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980.

26. Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes part I. *Information and Computation*, 100(1):1–40, September 1992.
27. Richard N.Taylor and Nenad Medvidovic. *Software architecture: Foundation, theory, and practice*. Wiley, 2010.
28. OMG. CORBA Specification, Version 4.0. http://www.omg.org/spec/CCM, 2008. [Accessed: January-2023].
29. OMG. Unified Modeling Language (UML), Version 2.5.1. https://www.omg.org/spec/UML/2.5.1/About-UML, 2017. [Accessed: January-2023].
30. M. Rodano and K. Giammarc. A Formal Method for Evaluation of a Modeled System Architecture. *Procedia Computer Science*, 20:210–215, 2013.
31. SAE. Architecture Analysis & Design Language (AADL). https://www.sae.org/standards/content/as5506d/, 2009. [Accessed: January-2023].
32. B. Selic. The Pragmatics of Model-Driven Development. *IEEE Software*, 20(5):19–25, 2003.
33. A. Shostack. *Threat Modeling: Designing for Security*. Wiley, 2014.
34. Joseph Sifakis, Saddek Bensalem, Simon Bliudze, and Marius Bozga. A theory agenda for component-based design. In *Software, Services, and Systems*, pages 409–439. Springer, 2015.
35. Chao Wang, Gary D. Hachtel, and Fabio Somenzi. *Abstraction Refinement for Large Scale Model Checking*. Springer Publishing Company, Incorporated, 2014.