PAPER NAME	AUTHOR
VECOS_2024_paper_32-8.pdf	Abdelhakim Baouya

WORD COUNT	CHARACTER COUNT
5819 Words	32428 Characters
PAGE COUNT	**FILE SIZE**
15 Pages	1.9MB
SUBMISSION DATE	**REPORT DATE**
Aug 4, 2024 7:05 AM GMT+2	Aug 4, 2024 7:05 AM GMT+2

● 27% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

- 21% Internet database
- 24% Publications database
- Crossref database
- Crossref Posted Content database
- 13% Submitted Works database

From Component-Based to Game-Theoretic Models: Analyzing Security within Drone-assisted Lifting Orchestration

Abdelhakim Baouya[1], Brahim Hamid[1], and Saddek Bensalem[2]

[1] IRIT, Université de Toulouse, CNRS, UT2, France
`abdelhakim.baouya@irit.fr`, `brahim.hamid@irit.fr`
[2] VERIMAG, Université Grenoble Alpes, CNRS, Grenoble, France
`saddek.bensalem@univ-grenoble-alpes.fr`

Abstract. Integrating diverse communication protocols presents a major hurdle for modern cyber-physical systems. While dedicated modeling formalism 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 rely on a modeling approach that leverages stochastic multi-player games (CSGs) with reward structures for both verification and strategy synthesis. We employ PATL (probabilistic alternating-time temporal logic with rewards) to express properties within these models. 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 and scalability of our approach by evaluating drone-assisted crane orchestration. This showcases our approach's capability to analyze systems exhibiting probabilistic, cooperative, and competitive interactions between concurrent components.

Keywords: Software architecture· Security· Formal Methods· Game Theory· PRISM

1 Introduction

Software architecture design is witnessing a trend towards multi-paradigm approaches, integrating diverse methodologies [29]. This includes Component-Based Development (CBD) [14,36], Model-Driven Engineering (MDE) [34], and formal methods [32]. 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 [31], CCM-like [30], and ADL-like [33] 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 exploring alternative methods to enhance the trustworthiness of communication in distributed systems at the design phase.

Formal verification using stochastic games [23] 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) [23,24] 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[25], their application to software architecture is lacking [10].

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 address the idea of using CSGs to specify and analyze security attacks as competitive or collaborative behaviors in the context of component-port-connector-based software architecture. 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. To validate our work, we study a set of representative threats based on Microsoft's STRIDE threat classification [3] against the components and the communication links in component-port-connector architecture views [14,5]. Finally, to evaluate our methodology's performance, scalability, and applicability, we present a case study from a European project CPS4EU [13] 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 CSG's formalism. Section 3 presents our software architecture modeling approach using the PRISM language with well-defined semantic rules. Section 4 presents an example of an attack scenario related to a representative threat from STRIDE categories. 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) [25] are based on the idea that players make choices concurrently in each state and then transition simultaneously. In CSGs, each

[3] STRIDE classifies threats into six categories: Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege.

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 [22]. The PRISM model is composed of a set of modules that can synchronize. Each module is characterized by a set of variables and commands (or transitions). The valuations of these variables represent the state of the module. A set of commands is used to describe the behavior of each module. A command takes the form: $[a_1, \ldots, a_n]\ g\ \rightarrow\ p_1 : u_1 + \ldots + p_n : u_n$ or, $[a_1, \ldots, a_n]\ g\ \rightarrow\ u$. This means that if the guard g is true, then an update u_i is enabled with a probability p_i for the conjunction of actions a_i, $i = 1, \ldots, n$. A guard is a boolean formula constructed from the variables of the module. 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 $v_i \in V$, with V being a set of local and global variables, and val_i are values evaluated via expressions denoted by θ such that $\theta : V \rightarrow \mathbb{D}$, where $\mathbb{D}$ is the domain of the variables.

The properties related to CSGs are expressed in rPATL [11] (reward Probabilistic Alternating Temporal Logic). For instance, for the following property: "*Players 1 and 2 have a strategy to ensure that the probability of system reboot occurring within 100 steps is less than 0.001, regardless of the strategies of other players*" is expressed in rPATL as: $\langle\langle 1,2\rangle\rangle \mathtt{P}_{<0.001}[\mathtt{F}^{\leq 100}\mathtt{reboot}]$. Here, " `reboot` " is the label that refers to the modeled system states. Using rewards structure, the following property expressed in natural language: "*What is the maximum cumulative reward r within 100 steps to reach* " `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 Revisiting the 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: $s \xrightarrow{g:\alpha!v} s'$ The module sends a value v , and

Pull: $s \xhookrightarrow{g:\alpha?v} s'$ The module 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*.

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

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

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).

Example 3.1

The system depicted in Fig. 1 models two components as players in producer-consumer mode. A connector facilitates the orchestration and recording of push-pull operations. Two ports, identified by the operation semantics rules in *Push/Pull*, are used: α and β that belongs to each component. Two components, producerConsumer1 and producerConsumer2, are responsible for transferring data. They concurrently access the connector, either pushing data in or pulling data out.

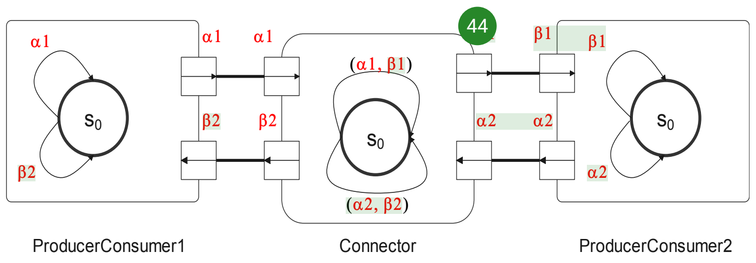

Fig. 1: Push and Pull Game Model in Component-Port-Connector formalism.

Listing 1.1: PRISM Code for Push/Pull on Connector

```
1   csg
2   // Players and their Roles
3   player p1  // Defines a player named p1
4     producerConsumer1  // Assigns the role "producerConsumer1" to player p1
5   endplayer  // End of player p1 definition
6
7   player p2  // Defines a player named p2
8     producerConsumer2  // Assigns the role "producerConsumer2" to player p2
9   endplayer  // End of player p2 definition
10
11  // Producer/Consumer Modules
12  module producerConsumer1  // Defines a module named "producerConsumer1"
13    v1 : int [0..10] init 1;  // Declares an integer variable "v1" with range
          0 to 10 and initializes it to 1
14    v2_prim : int [0..10] init 0;  // Declares an integer variable "v2_prim"
         with range 0 to 10 and initializes it to 0
15    [alpha1] true -> true;  // Transition labeled "alpha1" with  (stays in
         the same state)
16    [beta2] true -> (v2_prim'=v2);  // Transition labeled "beta2"  assigning
         "v2" to "v2_prim"
17  endmodule  // End of module "producerConsumer1"
18  module connector  // Defines a module named "connector"
19    win: : int [0..2] init 0;
20    [alpha1, beta1] true -> (win'=1);  // Allows both "alpha1" and "beta1"
         from producerConsumer1 and producerConsumer2 to happen together
21    [alpha2, beta2] true -> (win'=2);  // Allows both "alpha2" and "beta2"
         from producerConsumer1 and producerConsumer2 to happen together
22  endmodule  // End of module "connector"
23
24  module producerConsumer2  // Defines a module named "producerConsumer2"
25    v1_prim : int [0..10] init 0;  // Declares an integer variable "v1_prim"
         with range 0 to 10 and initializes it to 0
26    v2: int [0..10] init 1;   // Declares an integer variable "v2" with range
          0 to 10 and initializes it to 1
27    [alpha2] true -> true;  // Transition labeled "alpha2"
28    [beta1] true -> (v_prim'=v);  // Transition labeled "beta1" assigning "v"
          to "v_prim"
29  endmodule  // End of module "producerConsumer2"
```

The PRISM code in Listing 1.1 models a game with three modules (lines 12-29). Players represent the producerConsumer1 and producerConsumer2 (lines 3-9). Player p1 (producerConsumer1) has two integer variables: “v1” initialized to 1 and “v2_prim” initialized to 0 (lines 13-14). It has transitions labeled “alpha1” and "beta2" (lines 15-16). These transitions have no guard (they can always happen) and the variable "v" remains unchanged except for "v2_prim". Player p2 (producer-Consumer2) behaves similarly. It has two variables: "v1_prim" initialized to 0 and "v2" initialized to 1 (lines 16-17). It also has transitions labeled "alpha1" and "beta2" (lines 27-28) with similar behavior to player p1's transitions. The connector module (lines 18-22) allows both players' transitions to happen concurrently. Winning the game (win=1 or win=2) can be concurrently happens depending on the triggered actions.

When Player p1 participates in the game and completes the pushing task, the property can be expressed in rPATL as: $\langle\langle \mathtt{p1} \rangle\rangle \mathtt{P}_{>0.99} =?[\mathtt{F}\ \mathtt{win} = 1]$. Similarly, when player p2 participates in the game and completes the pushing task, the property can be expressed as $\langle\langle \mathtt{p2} \rangle\rangle \mathtt{P}_{>0.99} =?[\mathtt{F}\ \mathtt{win} = 2]$. Rewards can be modeled to capture the cost of observing specific actions on components or connectors. This is expressed as: $\langle\langle \mathtt{p1} \rangle\rangle \mathtt{R}_{\mathtt{max}} =?[\mathtt{C}^{\leq 100}]$. In other words, it represents the maximum cumulative reward achievable by player p1 upon winning the game.

4 A Representative Tampering Threat

This section explores a common threat to message integrity: tampering. As defined in [35,5,6], tampering refers to the unauthorized alteration of messages during transmission between communicating entities. In the context of message-passing communication style, this can manifest as a modification of message content (denoted as m) during data transfer.

$$\frac{[\![\mathscr{P}_{C_1}]\!] = s_i \xhookrightarrow{g_1:\alpha!v} s'_i \wedge \theta \models g_1 \wedge [\![\mathscr{P}_B]\!] = s_j \xhookrightarrow{g_2:\langle\alpha,\beta,\omega\rangle}_{\lambda} s'_j \wedge \theta \models g_2 \wedge \quad [\![\mathscr{P}_{C_2}]\!] = s_k \xhookrightarrow{g_3:\beta?v'} s'_k \wedge \theta \models g_3 \wedge [\![\mathscr{P}_{C_{Att}}]\!] = s_l \xhookrightarrow{g_4:\omega!x} s'_l \wedge \theta \models g_4}{\langle s_i, \ldots, s_j, \ldots, s_k, \theta\rangle \xrightarrow{\langle\alpha,\beta,\omega\rangle}_{\lambda} \langle s'_i, \ldots, s'_j, \ldots, s'_k,\ \theta'\rangle} \quad (\textit{Success})$$

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

$$\frac{[\![\mathscr{P}_{C_1}]\!] = s_i \xhookrightarrow{g_1:\alpha!v} s'_i \wedge \theta \models g_1 \wedge [\![\mathscr{P}_B]\!] = s_j \xhookrightarrow{g_1:\langle\alpha,\beta,\omega\rangle}_{1-\lambda} s'_j \wedge \theta \models g_2 \wedge \quad [\![\mathscr{P}_{C_2}]\!] = s_k \xhookrightarrow{g_3:\beta?v'} s'_k \wedge \theta \models g_3 \wedge [\![\mathscr{P}_{C_{Att}}]\!] = s_l \xhookrightarrow{g_4:\omega!x} s'_l \wedge \theta \models g_4}{\langle s_i, \ldots, s_j, \ldots, s_k, \theta\rangle \xrightarrow{\langle\alpha,\beta,\omega\rangle}_{1-\lambda} \langle s'_i, \ldots, s'_j, \ldots, s'_k,\ \theta'\rangle} \quad (\textit{Failure})$$

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

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: $\mathscr{P}_{C1}, and \mathscr{P}_{Att}$

who send data through the connector $\mathscr{P}_B$ to component $\mathscr{P}_{C2}$ 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 $\mathscr{P}_{Att}$ 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 $\mathscr{P}_B$.

5 Experiments

This scenario, illustrated in Fig. 2 [13], 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 our demonstration, the attack targets the communication channel as the vulnerable access point using a satellite system.

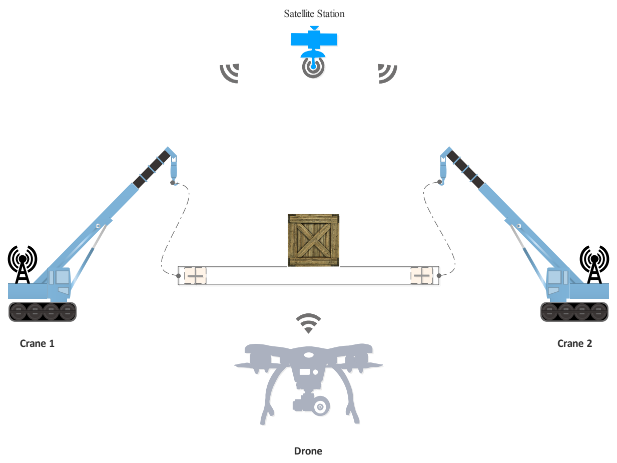

Fig. 2: Drone-assisted Lifting Orchestration [13].

Experimental setup. Within the set of functional and security properties, we have encoded properties in rPATL formalism. PRISM-games model checker 3.0 [24] 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 [16]) offering performance benefits for specific model structures. In addition, we have implemented the scenarios outlined in [38] 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

Listing 1.2: Connector Model Sensitive to an Attack (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
```

The composed model effectively captures the interplay between the drone, attacker, and cranes. A dedicated module, "ConnectorDroneCrane_1" specifically tracks the attack's effectiveness. This model, detailed in code snippet Listing 1.2, 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 [24] and a module (see Listing 1.3) 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.

Additionally, we introduce two reward structures in Listing 1.4 that align with each player's utility in the model. In this case, a reward of "cost" can be assigned for winning for the first and second players. To implement this, action lists or model states can once again be utilized to capture the assigned cost. To enable the inclusion

Listing 1.3: Rounds module to Represent Multiple Game Rounds.

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

Listing 1.4: 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
```

of additional attackers in the future, we have defined two essential formulas in lines 2 and 3.

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 attacker:

Properties in rPATL

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

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

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

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

5.3 Analyses

The verification results of (PRO1) and (PRO2) are depicted in Fig. 3 and Fig. 4, respectively. In Fig. 3, 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. 4 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. 5, 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. 6, 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.

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 six attackers. Each attacker can access the connector to tamper with data and potentially cause faults during the lifting operation.

In our study, we consider the verification cost and model construction function, which allow us to evaluate the model (as described in [37]). 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

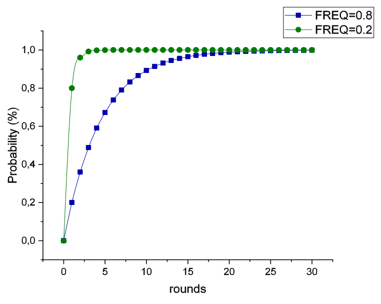

Fig. 3: Verif. PRO1.

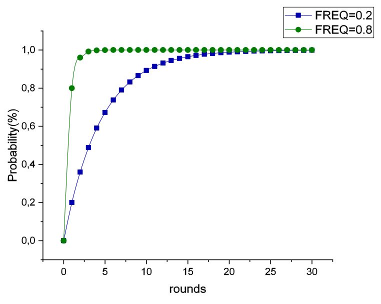

Fig. 4: Verif. PRO2.

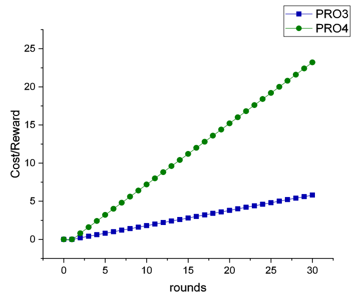

Fig. 5: Verif. PRO3 and PRO4 in Scenario1.

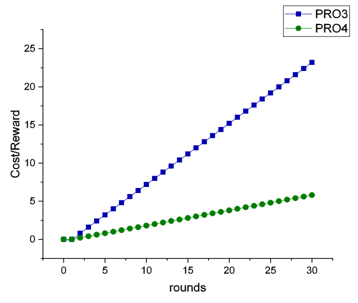

Fig. 6: Verif. PRO3 and PRO4 in Scenario2.

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.

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.

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 [24] require traversing the state space represented by the matrix diagram, which grows larger with increasing buffer size.

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.

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 [27], CSP [19], ACP [8], and π-calculus [28], LOTO [17]) and automata (like timed automata [3], I/O automata [21]). In our work, we focus on Probabilistic Automata[15,18], an extension of classical transition systems, for designing component behaviors. Transition systems (TS) are a widely used formalism to describe system behavior [4]. These systems can be visualized as directed graphs, where nodes represent states and edges represent transitions. Several component models like BIP [7], AUTOFOCUS AF3 [20], and UML (through activity and state machine diagrams) [31].

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 [9]. 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 [26] for analysis, while AUTOFOCUS AF3 utilizes NuSMV [12]. 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 [24], 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

In our work, we proposed an approach to security attack specification and analysis in component-based software architecture models via concurrent stochastic multi-player games (CSGs), where security properties in the model are defined as game goals. The proposed work uses formal methods, specifically probabilistic model checking with game semantics in PRISM games enhanced with rPATL logic for expressing properties adapted to CSG models. We illustrate the application of the proposed approach to through a set of representative threats based on Microsoft's STRIDE threat classification against the components and the communication links. This work demonstrates how formal methods, specifically probabilistic model checking with game semantics implemented in PRISM games, can be used to model and verify the security of collaborative and concurrent connector access for synchronized crane lifting operations guided by drone-generated orders. Additionally, we examined how connector design choices can affect scalability when considering scenarios with multiple attackers. 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.

References

1. Abdelhakim Baouya. Paper Artefacts Sources. https://acis-iot.github.io/vecos2024.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 and David L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 12(6):183–235, 1994.
4. Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. The MIT Press. OCLC: ocn171152628.
5. Abdelhakim Baouya, Brahim Hamid, and Saddek Bensalem. Modeling and analysis of data corruption attacks and energy consumption effects on edge servers using concurrent stochastic games. *Soft Computing*, 2024.
6. Abdelhakim Baouya, Brahim Hamid, Levent Gürgen, and Saddek Bensalem. Rigorous security analysis of rabbitmq broker with concurrent stochastic games. *Internet of Things*, 26:101161, 2024.
7. 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.
8. J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Control*, 60(1-3):109–137, 1984.
9. Simon Bliudze and Joseph Sifakis. The algebra of connectors—structuring interaction in BIP. 57(10):1315–1330.
10. PRISM Model Checker. Prism-games - publications. https://www.prismmodelchecker.org/games/publ.php.
11. 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.
12. 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.
13. 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-2024].
14. Ivica Crnkovic. Component-based software engineering for embedded systems. In *27th International Conference on Software Engineering*, ICSE '05, pages 712–713. ACM, 2005.
15. J. Andres Diaz-Pace, Rebekka Wohlrab, and David Garlan. Supporting the exploration of quality attribute tradeoffs in large design spaces. In Bedir Tekinerdogan, Catia Trubiani, Chouki Tibermacine, Patrizia Scandurra, and Carlos E. Cuesta, editors, *Software Architecture*, pages 3–19, Cham, 2023. Springer Nature Switzerland.
16. PRISM Development Team (eds.). Prism manual. https://www.prismmodelchecker.org/manual/ConfiguringPRISM/ComputationEngines. Accessed: March 27, 2024.
17. 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.
18. João M. Franco, Raul Barbosa, and M'rio Zenha-Rela. Automated reliability prediction from formal architectural descriptions. In *2012 Joint Working IEEE/IFIP Conference on*

Software Architecture and European Conference on Software Architecture, pages 302–309, 2012.

19. C.A.R. Hoare. Communicating Sequential Processes. *Commun. ACM*, 21(8):666–677, 1978.
20. Sudeep Kanav and Vincent Aravantinos. Modular transformation from af3 to nuxmv. In *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, 2017.
21. Dilsun K. Kaynar, Nancy Lynch, Roberto Segala, and Frits Vaandrager. The theory of timed i/o automata, 2011.
22. 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.
23. Marta Kwiatkowska, Gethin Norman, David Parker, and Gabriel Santos. Equilibria-based probabilistic model checking for concurrent stochastic games. In *Formal Methods – The Next 30 Years*, pages 298–315, Cham, 2019. Springer International Publishing.
24. Marta Kwiatkowska, Gethin Norman, David Parker, and Gabriel Santos. Prism-games 3.0: Stochastic game verification with concurrency, equilibria and time. In *Computer Aided Verification*, pages 475–487, Cham, 2020. Springer International Publishing.
25. 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.
26. 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.
27. Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980.
28. Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes part I. *Information and Computation*, 100(1):1–40, September 1992.
29. Richard N. Taylor and Nenad Medvidovic. *Software architecture: Foundation, theory, and practice*. Wiley, 2010.
30. OMG. CORBA Specification, Version 4.0. http://www.omg.org/spec/CCM, 2008. [Accessed: January-2023].
31. OMG. Unified Modeling Language (UML), Version 2.5.1. https://www.omg.org/spec/UML/2.5.1/About-UML, 2017. [Accessed: January-2023].
32. M. Rodano and K. Giammarco. A Formal Method for Evaluation of a Modeled System Architecture. *Procedia Computer Science*, 20:210–215, 2013.
33. SAE. Architecture Analysis & Design Language (AADL). https://www.sae.org/standards/content/as5506d/, 2009. [Accessed: January-2023].
34. B. Selic. The Pragmatics of Model-Driven Development. *IEEE Software*, 20(5):19–25, 2003.
35. A. Shostack. *Threat Modeling: Designing for Security*. Wiley, 2014.
36. 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.
37. Chao Wang, Gary D. Hachtel, and Fabio Somenzi. *Abstraction Refinement for Large Scale Model Checking*. Springer Publishing Company, Incorporated, 2014.
38. Quanyan Zhu and Tamer Başar. Dynamic policy-based ids configuration. In *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pages 8600–8605, 2009.

● 27% Overall Similarity

Top sources found in the following databases:

- 21% Internet database
- 24% Publications database
- Crossref database
- Crossref Posted Content database
- 13% Submitted Works database

TOP SOURCES

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Abdelhakim Baouya, Brahim Hamid, Levent Gürgen, Saddek Bensalem.... Crossref	3%
2	thesesups.ups-tlse.fr Internet	2%
3	Abdelhakim Baouya, Otmane Ait Mohamed, Samir Ouchani. "Toward a ... Crossref	1%
4	research.birmingham.ac.uk Internet	1%
5	University of Oxford on 2024-05-11 Submitted works	<1%
6	prismmodelchecker.org Internet	<1%
7	allhands.org.uk Internet	<1%
8	Samir Ouchani, Otmane Aït Mohamed, Mourad Debbabi. "A formal verif... Crossref	<1%

9	University of Newcastle upon Tyne on 2021-10-02 Submitted works	<1%
10	acis-iot.github.io Internet	<1%
11	dokumen.pub Internet	<1%
12	Loïc Thierry, Jason Jaskolka, Brahim Hamid, Jean-Paul Bodeveix. "Spe... Crossref	<1%
13	core.ac.uk Internet	<1%
14	pdfs.semanticscholar.org Internet	<1%
15	Abdel-Latif Alshalalfah, Otmane Ait Mohamed, Samir Ouchani. "A fram... Crossref	<1%
16	cse.yorku.ca Internet	<1%
17	tel.archives-ouvertes.fr Internet	<1%
18	ibisc.univ-evry.fr Internet	<1%
19	Quentin Rouland, Brahim Hamid, Jason Jaskolka. "Specification, detec... Crossref	<1%
20	patents.justia.com Internet	<1%

21	d-nb.info Internet	<1%
22	"Model and Data Engineering", Springer Science and Business Media L... Crossref	<1%
23	arxiv.org Internet	<1%
24	pure.tue.nl Internet	<1%
25	University of Glasgow on 2020-04-06 Submitted works	<1%
26	Nianyu Li, Mingyue Zhang, Jialong Li, Sridhar Adepu, Eunsuk Kang, Zhi ... Crossref	<1%
27	dblp.org Internet	<1%
28	groups.csail.mit.edu Internet	<1%
29	Abdelhakim Baouya, Brahim Hamid, Levent Gürgen, Saddek Bensalem.... Crossref	<1%
30	lcc.uma.es Internet	<1%
31	export.arxiv.org Internet	<1%
32	faculty.ece.vt.edu Internet	<1%

33	cse.msu.edu Internet	<1%
34	Asma Azzoug, Mustapha Anwar Brahami, Latéfa Ghomri, Belgacem Be... Crossref	<1%
35	discovery.researcher.life Internet	<1%
36	link.springer.com Internet	<1%
37	springerprofessional.de Internet	<1%
38	Abdelhakim Baouya, Djamal Bennouar, Otmane Ait Mohamed, Samir O... Crossref	<1%
39	Zhi Ma, Sheng Zhang, Zhiqi Chen, Tao Han, Zhuzhong Qian, Mingjun Xi... Crossref	<1%
40	drops.dagstuhl.de Internet	<1%
41	epdf.pub Internet	<1%
42	"Abstraction Refinement for Large Scale Model Checking", Springer Sci... Crossref	<1%
43	hdl.handle.net Internet	<1%
44	scribd.com Internet	<1%

45	scss.tcd.ie Internet	<1%
46	"Quantitative Evaluation of Systems", Springer Science and Business M... Crossref	<1%
47	"Software Architecture", Springer Science and Business Media LLC, 2023 Crossref	<1%
48	Higher Education Commission Pakistan on 2015-09-22 Submitted works	<1%
49	Mafijul Md. Islam, Aljoscha Lautenbach, Christian Sandberg, Tomas Ol... Crossref	<1%
50	hal.archives-ouvertes.fr Internet	<1%
51	Marwa Zeroual, Brahim Hamid, Morayo Adedjouma, Jason Jaskolka. "F... Crossref	<1%
52	Southern New Hampshire University - Continuing Education on 2023-1... Submitted works	<1%
53	"Game Theory and Machine Learning for Cyber Security", Wiley, 2021 Crossref	<1%
54	John Warsinkse. "CISSP: Certified Information Systems Security Profe... Crossref	<1%
55	Sudeep Kanav. "A modular approach to integrate verification tools in m... Crossref	<1%
56	University of Birmingham on 2021-05-06 Submitted works	<1%

57	ebin.pub Internet	<1%
58	fmt.ewi.utwente.nl Internet	<1%
59	veriware.org Internet	<1%
60	"Principles of Systems Design", Springer Science and Business Media ... Crossref	<1%