





Safety-Aware Strategy Synthesis for Autonomous System of Systems with UPPAAL^{*}

Nazakat Ali^{}, Muhammad Naeem^{}, Julieth Patricia Castellanos Ardila^{}, and
Sasikumar Punnekkat^{}

School of Innovation, Design and Engineering, Mälardalen University, Sweden
{nazakat.ali, muhammad.naeem, julieth.castellanos,
sasikumar.punnekkat}@mdu.se

Abstract. Systems of Systems (SoS) in critical domains like construction require the coordination of independent and heterogeneous Constituent Systems (CS) to accomplish complex missions. To help with such coordination, an architectural approach, called orchestration, has been proposed. However, safety in such an approach remains unexplored. In this paper, we present a safety-aware strategy synthesis framework to fill this gap. It combines formal modeling of CS and shared resources as timed automata, integration of safety contracts to capture assumptions and guarantees, and Q-learning strategy generation by using Uppaal Stratego. As a result, the framework enables the synthesis of execution strategies that not only fulfill mission objectives but also ensure safety constraints. We demonstrate our method through a case study in autonomous construction operations, highlighting its ability to minimize unsafe interactions and to reduce resource conflicts and waiting times.

Keywords: Safety Strategy Synthesis · SoS · UPPAAL · Formal Models

1 Introduction

Systems of Systems (SoS) are increasingly deployed in critical domains such as construction, where multiple independent and heterogeneous Constituent Systems (CS) collaborate to achieve common goals [14]. To manage such coordination, an architectural strategy called orchestration has been proposed [16]. Originating from service-oriented computing, orchestration uses a centralized control mechanism, called orchestrator, that governs the interactions among distributed services [12]. Applied to SoS, this entity acts as a service hub enabling CS collaboration [6]. However, if not properly managed, the SoS may also experience unsafe interactions, resource contention, and timing constraint violations [2].

The orchestration of autonomous CS within a SoS could benefit from using formal modeling techniques as they can accurately represent functional behavior and time-critical constraints to facilitate safety verification [2, 9]. While several formal modeling tools exist [3], UPPAAL [17] distinguishes itself as a robust

^{*} This Research is supported by the Vinnova-funded project SIMCON and SAILS, a pre-study aimed at investigating safety assurance of artificial intelligence systems.

model-checking tool for real-time systems [8], thanks to its ability to represent timed automata and analyze behavior under strict temporal constraints. It also allows synthesizing strategies that facilitate efficiency under variable and uncertain execution conditions [1]. However, strategy synthesis approaches, which often emphasize resource optimization, commonly fall short in addressing safety awareness in dynamic and stochastic environments [13].

In this paper, we present a safety-aware strategy synthesis framework that integrates formal modeling, safety contracts, and Q-learning, a model-free, off-policy reinforcement learning algorithm that derives optimal actions by updating a Q-table iteratively [18]. In particular, we introduce a formal modeling approach that captures the time-constrained behaviors and interactions of CS and shared resources. By integrating safety contracts into both the individual CS models and the overall SoS orchestrator, our method ensures that assumptions and guarantees related to key operational parameters, such as battery levels, task synchronization, charging, and task conflicts, are considered during mission execution. Furthermore, we use UPPAAL STRATEGO’s inbuilt Q-learning-based strategy synthesis to generate optimal execution strategies that satisfy mission goals while ensuring safety. We validate our approach through a detailed case study in the construction domain, demonstrating how safety-aware strategies reduce waiting times and prevent unsafe behaviors during mission execution.

This paper is structured as follows. Section 2 introduces the necessary preliminaries. Section 3 presents the proposed approach. Section 4 describes the use case. Section 5 presents the System Model for SoS. Section 6 discusses the simulation results. Finally, Section 7 concludes the paper.

2 Background

2.1 System of Systems

SoS [11] consists of multiple autonomous CS that collaborate toward shared goals by forming coalitions known as constellations. These constellations are subsets of CS that are interconnected to exchange data to provide specific capabilities [4]. What distinguishes an SoS from a traditional monolithic system is the emergent behaviors, i.e., functionalities that arise from real-time collaboration, not present in individual CS [10]. However, such emergent behavior can also lead to unforeseen and possibly hazardous situations [5]. SoS types include directed (central control), collaborative (decentralized), virtual (minimal control) [14], and acknowledged, i.e., centrally guided while CS retain mission influence [7].

2.2 Strategy Synthesis using UPPAAL STRATEGO

UPPAAL STRATEGO is a powerful tool designed for modeling and strategy synthesis in the context of stochastic hybrid games (SHGs) [8]. It models systems as interconnected timed automata, where each location denotes a specific state, and transitions govern how the system progresses between states. Transitions can include guards, i.e., logical conditions must be satisfied to enable a transition and

invariants to limit how long the system can stay in a given location. Particularly, UPPAAL STRATEGO has two types of transitions: controllable, which are governed by the system's decision logic, and uncontrollable, which reflect uncertain environmental dynamics and are depicted with dotted lines. UPPAAL STRATEGO aims to synthesize optimal strategies toward a goal, accounting for uncertainty, using queries like the one below to guide system actions.

$$\text{strategy } s = \min E(\text{cost}), [\leq T], \text{ExpList1} \rightarrow \text{ExpList2} : \langle \rangle \text{ goal} \quad (1)$$

where $\min E(\text{cost})$ instructs the tool to find a strategy s that minimizes a cost metric within a bounded time T , or until the defined goal is achieved. ExpList1 comprises discrete state variables, whereas ExpList2 includes continuous ones. During simulation, UPPAAL STRATEGO collects trace samples that reflect observations of the system's behavior. These samples guide the Q-learning algorithm to refine a policy for making controllable decisions toward the system's goal.

3 Proposed Approach

In this section, we propose a strategy synthesis framework (Fig. 1) that uses UPPAAL to model the time-sensitive behaviors of autonomous CS, safety contracts containing domain-specific safety requirements, and synthesize optimal control strategies for safe mission execution even dynamic and uncertain conditions.

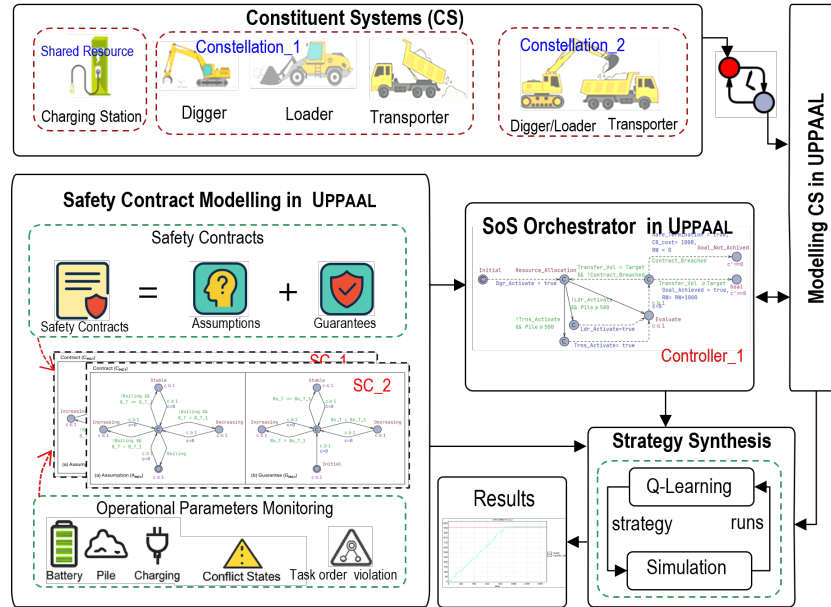


Fig. 1: Proposed Approach

3.1 Formal Modeling

The proposed framework adopts a two-tiered modeling structure: system and a SoS-level. At the system level, each CS (digger, loader, or transporter) is modeled as a network of timed automata. These automata represent operational cycles including task execution (e.g., excavation, loading, transporting), state transitions (e.g., idle, active, charging), and energy-aware behavior (e.g., battery thresholds, navigating to charging stations). Models include task-specific dynamics, such as pile volume generation, load capacities, and temporal execution constraints, along with synchronization channels to support inter-CS collaboration.

At the SoS level, the orchestrator is modeled as a supervisory control unit that interprets mission goals and coordinates the collective behavior of the CS constellation. It initiates task execution by dispatching commands to individual CS, synchronizes their progress through global variables and synchronization channels, and enforces global constraints on task sequencing, resource contention, and mission termination. This layer captures emergent SoS behaviors such as dynamic task reallocation, constellation, and centralized safety monitoring. The CS automata communicate with the orchestrator through channel synchronization and shared state variables, allowing CS-level decisions to be shaped by SoS-level reasoning and vice versa. This layered architecture preserves CS autonomy while enabling SoS-level collaboration and mission alignment.

3.2 Safety Contract Modeling

Formal modeling and integration of safety contracts into both the system and SoS-level automata is one of the main contributions of this paper. These contracts encode critical safety assumptions and guarantees in the form of operational constraints that directly influence the progression of execution traces. Unlike conventional techniques that treat safety verification as an external, post-synthesis activity, we model safety contracts as intrinsic system properties that are actively enforced throughout execution and learning. The significance of this contract framework is in its bidirectional integration. The safety contracts set rules for model behavior during execution and guide learning during strategy synthesis. This dual role establishes a continuous feedback loop between safety modeling and decision optimization.

3.3 Strategy Synthesis

Following the formal modeling of CS and integration of safety contracts, the framework advances to the synthesis of an execution strategy that balances mission efficiency with safety compliance (Fig. 6). The synthesis process is not a post hoc optimization overlay but a core design activity that directly interacts with the behavior models and embedded safety contracts. As illustrated in Fig.1, the strategy synthesis module receives formal automata models of the CS and orchestrator, along with the contract constraints, and iteratively learns an optimal control policy through simulation-guided reinforcement learning.

At its core, the strategy synthesis phase formulates the problem as a stochastic hybrid game, in which the system and its environment take turns in making decisions under partial observability and uncertainty. UPPAAL STRATEGO supports this interaction through the specification of controllable and uncontrollable transitions that allows the explicit modeling of decision points (e.g., when to dispatch a CS, when to initiate charging) as well as stochastic environmental responses (e.g., charging station availability or battery replacement).

The learning process is initiated through a series of randomized simulations that explore the state space of the composed SoS model. Each simulation run generates a trace composed of state-action pairs, where the actions reflect specific choices made at controllable transitions. These traces are then evaluated against a user-defined cost function. For each trace, a cumulative reward is calculated, which feeds into a Q-learning process that updates the expected utility of each action in a given state context. The integration of safety contracts into the model is important, as it ensures that unsafe trajectories are eliminated during the learning process. Transitions that would violate these safety contracts are blocked by guard conditions, and simulations that encounter such paths get penalty costs or experience early termination. As a result, the developed strategy is not only cost-optimal but also complies with formally specified safety policies that allows safe mission execution under uncertainties.

4 Use Case Description

Autonomous operations within construction environments offer substantial opportunities; however, they also present considerable challenges such as collaboration, safety, and efficiency. Tasks such as excavation, material transfer, and dumping involve multiple autonomous machines, each operating with its own local decision-making capabilities, but these machines must collaborate to achieve a shared mission goal. Unlike tightly integrated systems, these CSs exhibit operational independence, variable levels of autonomy, and asynchronous behavior, making safety and collaboration a complex challenge. Our industrial use case focuses on a mass removal operation at a construction site, where the goal is to transfer 2,000 tons of material from an excavation zone to a dump area. This operation is carried out by a constellation of three key autonomous machines: a digger, a loader, and a transporter. These machines must operate in a coordinated but decentralized manner, where each CS makes local decisions while collaborating with others to carry out its tasks safely and efficiently. The overall mission is managed by a central orchestrator, which receives the mission goal and determines the appropriate constellation of machines to execute it [16]. A task execution controller (**Executor**) within the orchestrator coordinates individual CS by assigning tasks and monitoring their progress. For example, the digger excavates and forms a pile, the loader transfers the material from the pile to the transporter, and the transporter delivers it to the dump site. This cycle repeats until the mission is complete. However, this setup introduces several coordination and safety challenges. Machines operate with limited energy, which

requires them to interrupt their tasks and return to a shared charging station when battery levels fall below a threshold. This introduces resource contention, as multiple CS may request charging simultaneously. Even more critically, unsynchronized task execution, such as the loader beginning to load while the transporter is still en-route or the transporter leaving before being fully loaded, can lead to operational hazards.

4.1 Safety Contracts

Embedding safety contracts at both levels of the model (i.e., system and SoS-level), we ensure safety guarantees are not only verifiable but operationalized during execution. The automata actively monitor contract compliance, and unsafe transitions are blocked or redirected to safe fallback states. This mechanism extends to the orchestrator, which interprets contract violation signals and dynamically adjusts task assignments, waits for preconditions to be restored, or reconfigures system execution to maintain contract satisfaction. We categorize the safety contracts into two groups: CS-Level Contracts (CS-C) and SoS-Level Contracts (SoS-C). Each contract is formulated as a pair (A, G) [15], where A represents the assumptions and G represents the guarantees. When multiple assumptions or guarantees exist, conjunctions are used: $A = \bigwedge_i A_i$ and $G = \bigwedge_i G_i$.

For instance, a CS may only proceed if its battery level exceeds a safe operational threshold. Another contract may enforce task execution order, e.g., loading does not begin until a sufficient pile is available, or transport is not initiated until the loader has completed loading operations. Charging station access is similarly governed by mutual exclusion policies encoded through shared variables. These constraints are realized through transition guards, synchronization mechanisms, and conditional invariants within the timed automata models.

CS-C1: Battery-Aware Operational Safety: Sudden battery depletion during operational phase can result in operational failures which can lead to unanticipated hazards. Hence the orchestrator must monitor battery levels of each CS and if the level drops below a specified threshold (say, 20%), it must enter a safe state and go to **HomeSite**.

$$A : \text{CS.operational}(t) = \text{true} \wedge \text{CS.battery}(t) < 20 \quad G : \text{CS} \rightarrow \text{SafeState}(t)$$

CS-C2: Charging Conflict Resolution: This contract ensures that multiple vehicles do not simultaneously wait for an occupied charging station, avoiding resource contention and potential unsafe queuing scenarios.

$$A : \neg \text{ChargingSlot} \quad G : \forall CS_i \in CS, \sum CS_i.Wait \leq 1$$

SoS-C1: Loader–Transporter Synchronization

SoS-C1.1: Pre-Loading Synchronization: The **Loader** shall only begin loading when the **Transporter** is correctly positioned and explicitly available (**T_Av1** = **true**). This ensures synchronized coordination between the systems, preventing unsafe or premature loading operations. If the **Loader** attempts to load

without transporter readiness, the system must detect this violation and trigger `Contract_Breached = true`.

$$\begin{array}{ll}
A_1 : \text{Ldr_Opr}(t) = \text{true} & G_1 : \text{T_Avl}(t) = \text{true} \\
A_2 : \text{Ldr_Opr}(t) = \text{true} \wedge \neg \text{T_Avl}(t) & G_2 : \text{Contract_Breached} \\
A_3 : \text{Ldr_Opr}(t) = \text{false} \vee \text{T_Avl}(t) = \text{true} & G_3 : \neg \text{Contract_Breached}
\end{array}$$

SoS-C1.2: Post-Loading Synchronization: Once the **Loader** has filled the **Transporter** to capacity ($\text{Load_vol} \geq \text{Max_LV}$), it must trigger a full signal. The **Transporter** is then required to exit the loading state within one time unit. If it remains in the loading state beyond this time ($\text{Trns_Opr} = \text{true}$ and $c \geq 1$), a contract breach is recorded by setting `Contract_Breached = true`.

$$\begin{array}{ll}
A_1 : \text{Load_vol} \geq \text{Max_LV} ; G_1 : \text{Trns_Opr}(t + \delta) = \text{false}, \delta \leq 1 \\
A_2 : \text{Load_vol} \geq \text{Max_LV} \wedge \text{Trns_Opr} = \text{true} \wedge c \geq 1 ; G_2 : \text{Contract_Breached} \\
A_3 : \text{Load_vol} < \text{Max_LV} \vee \text{Trns_Opr} = \text{false} ; G_3 : \neg \text{Contract_Breached}
\end{array}$$

SoS-C2: Exclusive Access to Pile: Simultaneous operation of the **Digger** and **Loader** at the pile site must be avoided to prevent physical collisions or interference during excavation and loading activities.

$$\begin{array}{l}
A : (\text{Dgr_Access_Pile} \vee \text{Ldr_Access_Pile}) \\
G : \neg(\text{Dgr_Access_Pile} \wedge \text{Ldr_Access_Pile})
\end{array}$$

SoS-C3: Safe Mission Termination: Upon mission success or contract violation, a system-wide safe termination must then be initiated and all vehicles return to **HomeSite**.

$$\begin{array}{ll}
A_1 : \text{Goal_Achieved}(t) = \text{true} \vee \text{Contract_Breached}(t) = \text{true} \\
G_1 : \forall CS_i : CS_i.\text{loc}(t) = \text{HomeSite} \\
G_2 : \text{Safe_Termination}(t) = \text{true}
\end{array}$$

5 System Model for SoS

5.1 Digger Model

The **Digger** (Fig.2) model represents an autonomous excavation vehicle that performs material digging and pile formation tasks at a designated excavation site. This model captures key behaviors such as task execution, power consumption, synchronization with other CSs, and dynamic charging strategies, using a network of timed automata in UPPAAL STRATEGO. The **Digger** starts at the **HomeSite** location, where all relevant variables such as **Pile**, **Dgr_Battery**, and **Dgr_cost** are initialized.

Upon receiving an activation signal from the **Executor** via the **Active!** synchronization channel, the **Digger** transitions to the **Travel2Site** location, representing its movement to the excavation zone. During this transition, energy consumption is modeled using a clock variable (**Battery**) that decreases at a

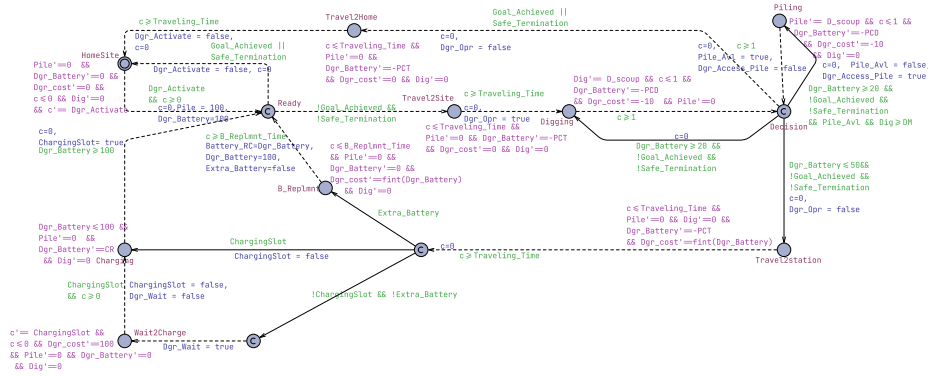


Fig. 2: Digger Model in UPPAAL

rate corresponding to the travel power consumption ($\text{Battery}' = -\text{PCT}$). Once the **Digger** reaches the excavation area, it enters the **Digging** state, where it begins the digging operation. The **Pile** clock accumulates at the rate of material excavation ($\text{Pile}' = \text{D_scoup}$) at **Piling** location and battery energy decreases due to the digging power consumption rate ($\text{Battery}' = -\text{PCD}$) both in **Digging** and **Piling** states.

The model actively monitors the **Digger**'s battery level, and when it drops below 20%, it initiates a transition to the charging station via the **Travel2Station** location. If the charging station is occupied (i.e., **ChargingSlot** == **false**), the **Digger** enters a waiting state (**Wait2Charge**) until the slot becomes available. Charging is carried out in the **Charging** state until the battery level reaches 100%. To improve charging efficiency and reduce system idleness, the model introduces controllable transitions that enable the **Digger** to either continue operating at low battery or preemptively initiate charging to avoid contention.

Additionally, the **Digger** includes a choice-based charging strategy where, depending on the availability of a fully charged spare battery (`Extra_Battery == true`), it may opt for a fast battery replacement instead of going for charging. These choices are encoded as controllable transitions, which are learned and optimized using RL during strategy synthesis. Upon completion of the assigned excavation goal (`Goal_Achieved == true`), the **Digger** transitions back to **Travel2Home**, concluding its operational cycle. This model enables energy-aware evaluation, CS coordination, and strategy optimization, central to the SoS constellation.

5.2 Loader Model

The **Loader** model (Fig.3) specifies the behavior of an autonomous machine responsible for transferring material from the pile, formed by the **Digger**, into the **Transporter**. It models task initiation, resource synchronization, battery-aware decisions, and loading dynamics as a timed automaton.

The **Loader** begins at the **HomeSite** location, with initial values set for relevant parameters such as battery and etc. Upon receiving the **Active!** synchro-

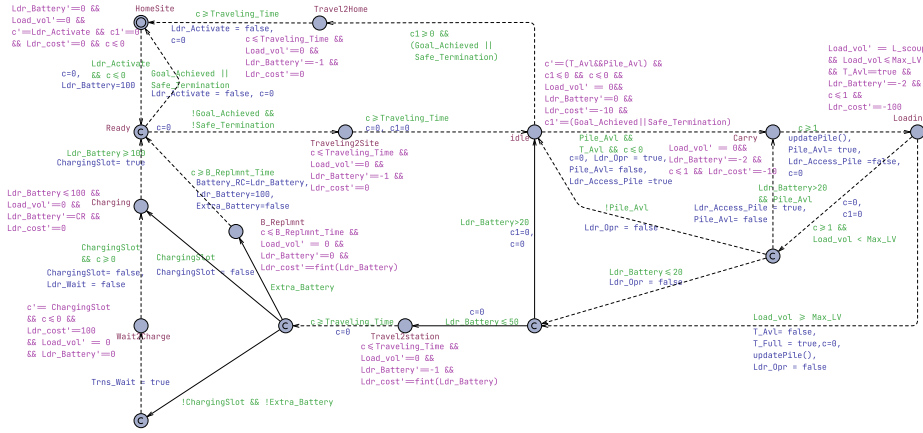


Fig. 3: Loader Model in UPPAAL

nization signal from the **Executor**, it transitions to the **Travel2-Site** location. Once it reaches the excavation site, the **Loader** enters a **Wait** state. In this location, the automaton checks for two preconditions: the availability of a sufficient pile volume $Pile \geq 2 \times L_scoup$ and the presence of an available **Transporter** ($T_Av1 == true$). Only when both conditions are satisfied does the **Loader** proceed to the **Loading** location. During the loading phase, the **Loader** transfers material to the **Transporter**. The material transfer is modeled via an accumulation variable ($Load_Vol'$) that increments at a fixed rate (L_scoup). Simultaneously, the pile volume is decremented, and the battery clock is updated to reflect task-related energy consumption. Loading continues until the **Transporter** reaches its full capacity i.e., $Load_Vol \geq Max_LV$.

The **Loader** model incorporates a dynamic charging mechanism. When the battery level falls below a predefined threshold, the **Loader** can initiate a transition to the **Travel2Station** location for recharging. If the charging station is occupied, it waits in the **Wait2Charge** location until the **ChargingSlot** becomes available. Once the slot is free, the **Loader** enters the **Charging** state and remains there until fully recharged ($Battery \geq 100$). Similar to the **Digger**, the **Loader** also supports a fast battery swap option governed by the availability of an extra battery. This choice, along with the charging decisions are modeled as controllable actions, allowing learning-based adaptation via UPPAAL STRATEGO.

The **Loader** concludes its cycle by returning to the **Travel2Home** location upon satisfaction of the global mission condition. The model synchronizes with both **Digger** and **Transporter** while optimizing energy and reducing idle time, aiding efficient SoS-level strategy synthesis.

5.3 Transporter Model

The **Transporter** model (Fig.4) governs the operation of an autonomous vehicle that carries excavated material from the loading area to the designated dumping site. It starts at **HomeSite**, awaiting the **Executor's Active!** signal to

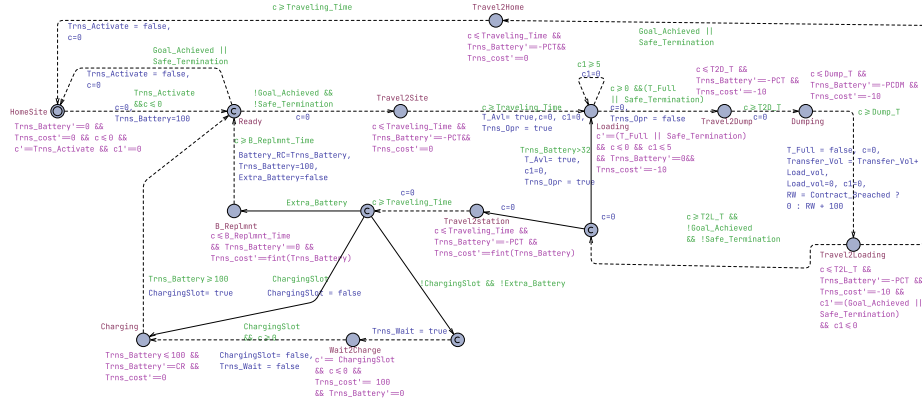


Fig. 4: Transporter Model in UPPAAL

begin its mission. Upon activation, it transitions to **Travel2Site** and reaches the loading area. Then, the **Transporter** enters the **Loading** location, sets the **T_Avl** flag, and signals the **Loader** to start loading. While in the **Loading** state, the **Transporter** receives material until it reaches full capacity ($\text{Load_Vol} = \text{Max_LV}$). Then, it moves to the **Travel2Dump** location to transit to the dumping area. The **Dumping** state models the material offloading process, which completes within a bounded time (**Dump_T**). After dumping, it goes back to the loading zone via **Travel2Loading** location, provided that the battery level supports another cycle. If the battery level is insufficient, the **Transporter** is diverted to the charging station by following the same charging logic as the **Digger** and **Loader**.

The **Transporter** model includes both waiting (**Wait2Charge**) and charging (**Charging**) states. It supports battery replacement if a fully charged spare battery is available same as **Digger** and **Loader**. The **Transporter** concludes its cycle by returning to the **Travel2Home** location upon satisfaction of the global mission condition.

5.4 Executor Model

The **Executor** acts as the central task orchestrator for our SoS use case that ensures synchronized mission execution among autonomous CS such as the **Digger**, **Loader**, and **Transporter**. The **Executor** model as shown in Fig.5, begins its operation in the **Resource_Allocation** location, a committed state where the system activates each CS through dedicated guards and assignments (e.g., $\text{Dgr_Activate} = \text{true}$). Once activation is completed, the **Executor** moves to the **Evaluate** location, where it continuously monitors mission parameters, such as **Transfer_Vol** and **Contract_Breached**. If the mission objective is achieved (i.e., $\text{Transfer_Vol} \geq \text{Target}$), the model transitions to the **Goal** location, and **Goal_Achieved** is set to **true** that signals all CS to halt operations. Alternatively, if any of the safety contracts breach, the system transitions to the **Goal_Not_Achieved** location that activates the **Safe_Termination** flag.



CS-C1: This contract (Fig.6, a) ensures that no CS (**Digger, Loader, Transporter**) continues operation below a minimum battery threshold. The model observes flags like **Dgr_Opr**, **Ldr_Opr**, and **Trns_Opr** in conjunction with battery variables. If any active CS is operating while its battery level falls below 20%, the model transitions from the **Safe** state to the **Unsafe** state and sets **Contract_Breached = true**. This contract prevents mid-task failures by enforcing energy-aware behavior across the SoS.

SoS-C1.1: SoS-C1.1 as shown in Fig.6 (c), guarantees that the **Loader** only operates when the **Transporter** is present and available for loading. If `Ldr_Opr = true` while `T_Avl = false`, the **Loader** is attempting to deposit material into an unready **Transporter**. This condition may lead to **Unsafe** state.

SoS-C2: SoS-C2 ensures orderly access to the shared resource **Pile** by the Digger and Loader. It ensures that only one of the **Dgr_Access_Pile** or **Ldr_Access_Pile** flags is active at any given time. Simultaneous pile access results in unsafe transitions. This contract (Fig.6 (e)) prevents physical collisions and ensures operational exclusivity over the shared material zone.

SoS-C3: Upon mission completion or detecting a contract violation, the **Executor** must initiate a safe termination process by setting **Safe_Termination** = **true**. This contract (Fig.6 (f)) ensures that termination signal is issued within a specified time window. The model enters a **Delay** state upon detecting mission completion or violation, and transitions to an **Unsafe** state if no response is triggered in time. This enforces graceful shutdown policies across the SoS.

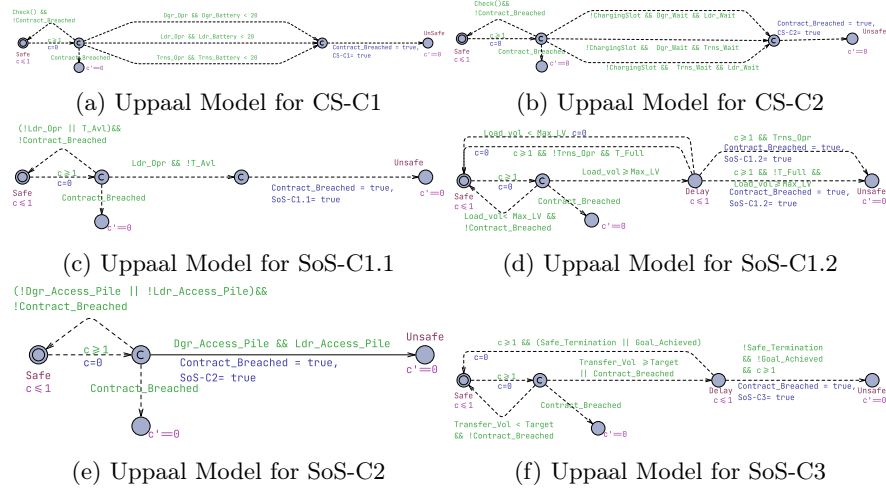


Fig. 6: Uppaal models for safety contracts

6 Simulation Results and Discussion

The simulation campaign begins with validation of mission completion under the synthesized strategy. As illustrated in the **Executor** model, the system successfully achieved the predefined transfer target of 2000 units. This goal is reached within approximately 600 time units, demonstrating that the synthesized strategy is not only safety-compliant but also mission-efficient. The orchestrator ensures performance goals are met without compromising safety.

To quantitatively evaluate the effectiveness of the synthesized safe strategy in ensuring safety-compliant execution of CS and SoS operations, statistical model checking was performed using UPPAAL STRATEGO. The evaluation used probabilistic queries of the following form:

$$\text{Pr}[\leq 3000](\langle \rangle \text{CS-x/SoS-x.Unsafe}) \text{ under Safe2}$$

where **CS-x/SoS-x.Unsafe** denotes the violation location associated with a specific safety contracts in the model, and **Safe2** represents the synthesized strategy obtained through Q-learning-based strategy synthesis using Eq.(1). The query estimates the probability that the system enters an unsafe state within a mission duration of 3000 time units while executing the **Safe2** strategy. A 95% confidence interval (CI) quantifies the estimate's reliability.

All six contracts were evaluated under the synthesized strategy. The probabilistic queries for each requirement consistently returned zero observed violations in 72 simulation runs per query. This result indicates that, with high statistical confidence, the probability of a contract violation under the synthesized strategy is low. In practical terms, policy **Safe2** ensured contract-level safety across all CS and coordination scenarios. Such consistent satisfaction across diverse safety contracts affirms the effectiveness of reinforcement learning-based

strategy synthesis in integrating contract logic into runtime behavior. The resulting strategy is not only performance-oriented but also tightly coupled with formal safety contracts that enables safe decision-making under shared-resource contention, temporal dependencies, CS-level and SoS-level interactions.

As an example, we present two (CS-C1 and SoS-C2) contract with and without safe strategy. **CS-C1** enforces battery-aware operation, ensuring that CS refrain from executing operational tasks when their battery levels fall below a predefined safety threshold. Without the safe strategy, the query:

$$\Pr[\leq 3000] (\langle \rangle \text{ CS-C1.Unsafe})$$

result in a confidence interval of: $\Pr \in [0.193363, 0.288507]$ based on 78 observed violations over 327 simulation runs. This indicates that, in the absence of strategic guidance, approximately 19% to 29% of mission traces resulted in CS operating unsafely with insufficient energy reserves. Such behavior risks mission failure and unsafe states, but under **Safe2**, violations dropped to zero.

SoS-C2 ensures mutual exclusion over the shared resource pile, preventing simultaneous access by the digger and loader, which could lead to physical collisions or interference. In the uncontrolled setting, the query:

$$\Pr[\leq 3000] (\langle \rangle \text{ SoS-C2.Unsafe})$$

resulted in a violation probability of: $\Pr \in [0.223363, 0.298507]$ suggesting that approximately one-quarter of mission traces failed to enforce exclusive access. This level of non-compliance indicates a high risk of unsafe behavior in autonomous operations where physical co-location must be carefully managed. Under the synthesized **Safe2** strategy, however, the probability of violation fell to 0 with no observed mutual access violations. This outcome demonstrates that the strategy successfully enforced serialization of pile access, coordinating the digger and loader to act in turn. The learning process incorporated safety constraints into scheduling decisions, ensuring operational exclusivity over the material zone throughout execution. The simulation results show that strategy synthesis in UPPAAL STRATEGO, guided by safety contract-based modeling yields safe, efficient policies for complex, resource and time constrained SoS.

7 Conclusion

This paper presented a safety-aware strategy synthesis framework for orchestrating autonomous SoS using UPPAAL STRATEGO. The proposed approach integrates formal modeling, safety contracts, and Q-learning-based synthesis to ensure safe and efficient coordination of CS within SoS orchestrations. The methodology avoids ad-hoc tuning or manual intervention by grounding decisions in both structural correctness and operational data, paving the way for scalable and certifiable deployment in safety-critical domains.

Future work includes integrating this framework with the SOSoS process introduced in [6], to incorporate systematic hazard analysis and variability man-

agement for more comprehensive safety assurance. In addition, we aim to evaluate larger and more complex SoS scenarios and explore alternative learning techniques to improve synthesis efficiency and robustness.

References

1. Ali, N., Naeem, M., Castellanos-Ardila, J.P., Punnekkat, S.: Formal modeling and strategy synthesis for resource optimization in system of systems. In: 20th Annual System of Systems Engineering Conference (2025)
2. Ali, N., Punnekkat, S., Rauf, A.: Modeling and safety analysis for collaborative safety-critical systems using hierarchical colored petri nets. *Journal of Systems and Software* **210**, 111958 (2024)
3. Armstrong, R.C., Punnoose, R.J., Wong, M.H., Mayo, J.R.: Survey of existing tools for formal verification. Tech. rep., Sandia National Lab., Livermore, USA (2014)
4. Axelsson, J.: A refined terminology on system-of-systems substructure and constituent system states. In: 14th Annual SoSE. pp. 31–36. IEEE (2019)
5. Beland, S.C., Miller, A.: Assuring a complex safety-critical systems of systems. *SAE Transactions* pp. 974–988 (2007)
6. Castellanos-Ardila, J.P., Ali, N., Punnekkat, S., Axelsson, J.: Making systems of systems orchestrations safer. In: European Safety and Reliability (ESREL) and Society for Risk Analysis Europe (SRA-E) (2025)
7. Dahmann, J.S., Baldwin, K.J.: Understanding the current state of us defense systems of systems and the implications for systems engineering. In: 2nd Annual IEEE Systems Conference (2008)
8. David, A., Jensen, P.G., Larsen, K.G., Mikučionis, M., Taankvist, J.H.: Uppaal stratego. In: Tools and Algorithms for the Construction and Analysis of Systems: 21st International Conference, TACAS-ETAPS , London, UK. Springer (2015)
9. Eddine, C., Hameurlain, N., Belala, F.: A maude-based formal approach to control and analyze time-resource aware missioned systems-of-systems. In: Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE). IEEE (2023)
10. Inocêncio, T.J., Gonzales, G.R., Cavalcante, E., Horita, F.E.: Emergent behavior in system-of-systems: A systematic mapping study. In: Proceedings of Brazilian Symposium on Software Engineering. pp. 140–149 (2019)
11. ISO/IEC JTC 1/SC 7: *ISO/IEC/IEEE 21841:2019. Systems and Software Engineering — Taxonomy of System of Systems*
12. Josuttis, N.M.: SOA in practice: the art of distributed system design. O'Reilly Media, Inc. (2007)
13. Lahijanian, M., Almagor, S., Fried, D., Kaviraki, L., Vardi, M.: This time the robot settles for a cost: A quantitative approach to temporal logic planning with partial satisfaction. In: AAAI Conference on Artificial Intelligence. vol. 29 (2015)
14. Maier, M.W.: Architecting Principles for Systems-of-Systems. *The Journal of the International Council on Systems Engineering* **1**(4) (1998)
15. Naeem, M., Seceleanu, C.: Contract-based verification of digital twins. In: International Conference on Engineering of Complex Computer Systems. No. 29th (2025)
16. Nordstrom, T., Sutfield, L.R., Besker, T.: Exploring different actor roles in orchestrations of system of systems. In: 19th System of Systems Engineering Conference. pp. 190–196 (2024)
17. Uppaal Team: Uppaal: Model Checking and Validation Tool (2025), <https://uppaal.org/>, accessed: 2025-03-12
18. Watkins, C.J., Dayan, P.: Q-learning. *Machine learning* **8**, 279–292 (1992)