The Journal of Systems and Software 226 (2025) 112416

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Modeling and verifying resources and capabilities of ubiquitous scenarios for Unmanned Aerial Vehicle swarm☆

Manqing Zhang [a], Yunwei Dong [a,*], Tao Zhang [b], Kang Su [a], Zeshan Li [a]

[a] School of Software, Northwestern Polytechnical University, Xi'an, China
[b] School of Computer Science and Engineering, Macau University of Science and Technology, 999078, Macao Special Administrative Region of China

ARTICLE INFO

Keywords:
Unmanned Aerial Vehicle swarm
Application scenarios
Meta-level theory
Formal verification

ABSTRACT

Unmanned Aerial Vehicle (UAV) swarms are increasingly deployed in both military and civilian sectors due to their ability to manage numerous resources, execute complex functionalities, and operate under strict spatiotemporal constraints in task-driven environments. However, existing task description methods are often restricted to specific operations and lack the flexibility to represent dynamic and intricate scenarios. To overcome these limitations, we introduce a meta-level theory-based UAV swarm application scenario model. This approach abstracts three primary meta-models: mission meta-model, resource meta-model, and constraint meta-model. We developed the UAV Swarm Application Scenario Modeling Language (ASML), which enables formal scenario descriptions and analysis. Additionally, we establish a set of transformation rules to convert ASML representations into timed automata. To validate the effectiveness of the proposed approach, we apply it to a highway inspection scenario and utilize the UPPAAL model checking tool to verify the correctness of the model. The experimental results from the highway inspection scenario show that our approach significantly enhances the accuracy of UAV swarm scenario modeling while improving adaptability to dynamic environments. Moreover, the results also demonstrate the model's correctness, reinforcing the reliability of our framework.

1. Introduction

The modeling of resources and capabilities in the context of Unmanned Aerial Vehicle (UAV) swarms in ubiquitous scenarios forms the foundation for building efficient, intelligent, and collaborative systems. With the rapid development of UAV technology, UAV swarm (Zhou et al., 2020) systems are gradually becoming a primary approach for multi-task collaboration, with application scenarios spanning aerial surveillance (Wu et al., 2020; Javaid et al., 2023), search and rescue (Arnold et al., 2020; Lomonaco et al., 2018), logistics delivery (Wen et al., 2018; Zhong et al., 2023; Khan et al., 2020), and disaster rescue. In these scenarios, systems are required to execute tasks in complex environments, which demands UAV swarms to utilize limited resources efficiently and possess the ability to adapt dynamically. To achieve this, it is essential to precisely model the resources of UAV swarms (such as loading capabilities, communication bandwidth, energy consumption, etc.) and reasonably assess and verify their task execution capabilities.

Existing research only focuses on specific mission scenarios, such as only applicable to regional monitoring (Bozhinoski et al., 2015) or search and rescue (Molina et al., 2017) mission scenarios. However, the broad adoption of UAV swarms introduces challenges in more complex scenarios, characterized by task diversity and environmental uncertainty. Existing task modeling methods cannot achieve reuse and accurately describe complex application scenarios when dealing with diverse application scenarios.

To overcome this limitation, we propose a comprehensive method for modeling UAV swarm application scenarios. Our approach involves using meta-level theory to abstract UAV swarm scenarios into mission, resource, and constraint meta-models. By leveraging these meta-models, we can effectively characterize diverse UAV swarm scenarios through the development of corresponding mission, resource, and constraint models. We further formalize the meta-model into a dedicated modeling language, complete with defined lexical and syntactic rules. Additionally, we translate the modeled scenarios into UPPAAL timed automata – a tool developed by Uppsala University and Aalborg University – to verify scenario correctness. In summary, this paper offers the following contributions:

- We propose a UAV swarm application scenario modeling method based on meta-level theory. This modeling method can accurately

☆ Editor: Wong W. Eric.

* Corresponding author.
E-mail address: yunweidong@nwpu.edu.cn (Y. Dong).

https://doi.org/10.1016/j.jss.2025.112416
Received 28 September 2024; Received in revised form 9 January 2025; Accepted 25 February 2025
Available online 8 March 2025

describe complex and diverse UAV swarm application scenarios in detail.

- We design and implement a UAV swarm Application Scenario Modeling Language (ASML). We have developed a graphical tool interface that enables low-code and flexible programming.
- We define the conversion rules from ASML to timed automata, which can realize the formal verification of the constructed application scenarios.
- We verify the feasibility of the proposed modeling method and language in a logistics handling scenario.

The rest of this paper is organized as follows. In Section 2, we provide an overview of the background knowledge relevant to our study. Section 3 details our scenario model and the corresponding modeling language. Section 4 describes the verification method for the UAV application scenario. Section 5 presents the simulation experiment and verification results for a logistics handling scenario. Section 6 discusses related work, and Section 7 concludes the paper.

2. Background

2.1. Timed automata

Timed automata are widely used in formal modeling, formal verification, simulation, automatic code generation and automatic model repair of sequential systems (Kölbl et al., 2020). Timed automaton is an extension of finite automaton, adding a clock and having the semantics of continuous real numbers. A timed automata is a five-tuple $TA = (L, l_0, C, E, I)$. Among them, L is a finite set of *locations*. $l_0 \in L$, and represents the starting position. C is a finite set of clock variables. $E \subseteq L \times G(C) \times 2^C \times L$ C is a set of *transition*. I is the set of *invariants*.

The state of timed automata can be represented by a tuple (l, v), where l represents a *location* of the timed automata, and v represents the clock value that satisfies the *locationinvariant* of l. Given a clock variable x, $v(x)$ represents the value of x at state (l, v). All clock variable values increase synchronously as time passes. Suppose (l_1, g, r, l_2) is a transition on the time automata. When the current location of the time automata is l_1 and the time guard g is satisfied, the transition may occur. After the transition occurs, the current location of the timed automaton becomes l_2, the values of the clock variables in the set r are reset to 0, and the values of other clock variables remain unchanged.

2.2. UPPAAL toolbox

UPPAAL is a toolbox for real-time system modeling and verification, which uses a timed automata network to simulate the system's behavior, and uses Timed Computational Tree Logic (TCTL) to describe the properties of the system.

In UPPAAL, each timed automata is called a Template, and a network composed of multiple concurrent timed automata becomes a timed automaton network $TAN \equiv TA_1 \| TA_2 \cdots \| TA_n$. A template consists of *Locations* and *Edges*. *Locations* can be constrained with location *invariants*, and the automaton can remain at that location as long as the clock value satisfies the invariant condition for that location. *Edges* contains four optional types of elements: selections, guards, synchronizations and updates. *Selections* non-deterministically binds the given identifier to a value within the given range. *Guard* indicates whether the state can be transitioned. *Synchronization* defines the channels used for synchronization between networks of timed automata. *Update* indicates that when the state transitions, the expression on the transition's edge will update its variable value.

UPPAAL uses TCTL to define the syntax of the property verification specification language. Verifiers only need to write corresponding query statements to verify the properties of the system. The specific syntax and meaning of the UPPAAL property query language are shown in Table 1.

Table 1
Property description language syntax.

Formula	Description
$A[]\ p$	p always holds for all states of all paths.
$A <>\ p$	For all paths, p eventually holds.
$E[]\ p$	There exists a path for which all states p always hold.
$E <>\ p$	There is a path such that p eventually holds.
p imply q	If p satisfies, then q also satisfies.
$p \rightarrow q$	Whenever p holds, q will eventually hold.

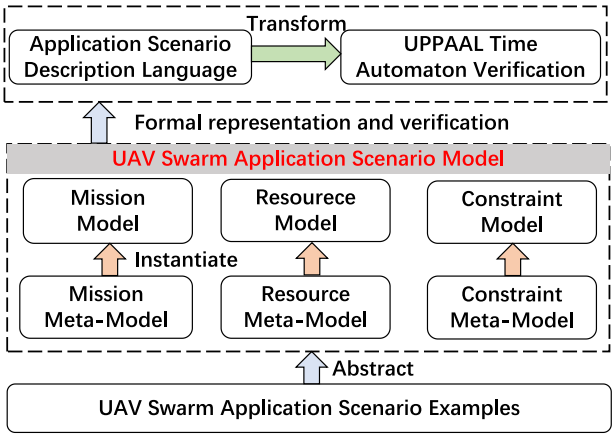

Fig. 1. Overview of UAV swarm scenario modeling.

3. Modeling method

3.1. Overview

To thoroughly abstract and describe the UAV swarm application scenario, we employ a modeling method based on meta-level theory. Specifically, as shown in Fig. 1, we begin by analyzing various examples of UAV swarm scenarios and distill them into three primary models: mission, resources, and constraints. By identifying common features across these scenarios, we develop a mission meta-model, a resource meta-model, and a constraint meta-model, which serve as the foundation for constructing detailed scenario models. Metamodel (Kühne, 2006) can be regarded as the "model" of the model, which can be used to define the model concept and create corresponding elements for the model. The metamodel is equivalent to the model's template, and the model can be regarded as an instance of the metamodel. Through the template definitions and concepts provided by the metamodel, models can be abstracted and interactive relationships between models can be described, providing high scalability and flexibility. The application scenario model ultimately consists of a task model, a resource model, and a constraint model.

Additionally, existing Domain-Specific Languages (DSLs) often have limited expressive power (Fowler, 2010) and are insufficient for comprehensively addressing multi-scenario requirements such as tasks, resources, and environmental constraints. To overcome this limitation, we propose and develop an application scenario modeling language (ASML) specifically for UAV swarms. This language is designed to provide a formal method for describing UAV swarm scenarios and to represent the application scenario model in detail. To ensure the accuracy of the developed scenario, we convert the scenario model into a timed automaton and employ the model-checking tool UPPAAL to validate its properties.

The scenario model is a triplet $Scenario = \langle Mission, Resource, Constraint \rangle$, where (1) The mission meta-model describes the specific tasks the UAV aims to accomplish within the scenario, (2) The resource meta-model defines the capabilities and limitations of the UAV's resources throughout the mission, and (3) The constraint meta-model

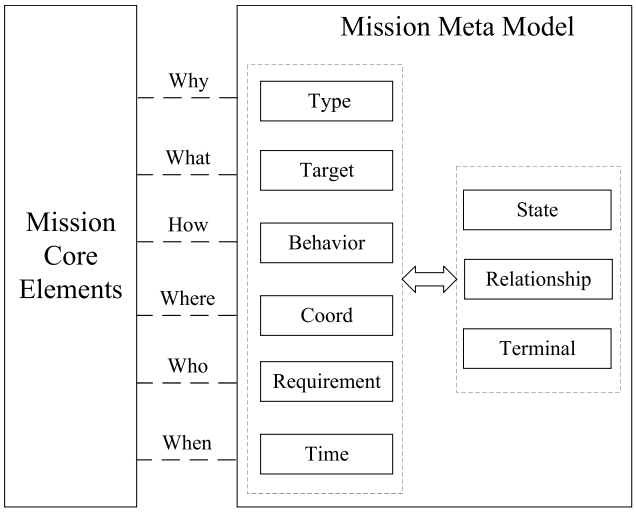

Fig. 2. Mission metamodel element extraction model.

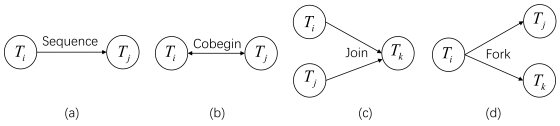

Fig. 3. Temporal relationship constraints on mission.

addresses limitations relationships among tasks and capabilities of UAV's resources, such as the mission environment, weather conditions, obstacles, and other factors, allowing the UAV to effectively adapt to varying environmental conditions. It is described in detail following.

3.2. Mission meta-model

We employ the 5W1H abstract domain ontology and its associated relationships to define mission information within the mission meta-model precisely. Laswell proposed the "5 W analysis method" (Wenxiu, 2015; Parton et al., 2009) and gradually formed a complete "5W1H" model (Yu and Bi, 2010). 5W1H has been used to model domain ontologies conceptually (Yang et al., 2011). Specifically, as shown in Fig. 2, we derive UAV mission ontology elements based on six aspects: purpose (Why), execution entity (Who), location (Where), timing (When), resources or events (What), and methods (How). Additionally, we incorporate mission status, mission relationships, and terminal flags associated with each mission.

The mission meta-model abstracts the shared attributes among various missions. The mission meta-model is ultimately composed of 5-tuples, $Mission = \langle ID, Name, Description, Property, Include \rangle$, where (1) The ID represents the identifier of the scenario mission, and (2) the Name represents the name of the scenario mission. (3) The Description represents the description of the scenario mission, (4) The Property is a set that represents all properties of the scenario mission, and (5) The Include lists the lower-level tasks and their IDs that are associated with this mission layer. For instance, the scenario tasks are organized into a hierarchical structure with up to three levels: Mission, task, and subtask. A mission can encompass one or more tasks, and each task may further include multiple subtasks.

The property is also defines a 9-tuple $Property = \langle Type, State, Time, Coord, Behavior, Relationship, Requirement, Target, Terminal \rangle$, which corresponds to the mission. It will discuss in detail the meaning of each element of the task property:

(1) Type: The mission type denotes the task level, corresponding to a mission, task, or subtask. By defining mission types, we can effectively manage the inclusion relationships between tasks, identify the specific type of each task, and determine whether a task is at the terminal level.

Table 2
UAV mission behavior.

Keywords	Parameters	Description
TAKE_PHOTO	shooting location	shooting
FILM_VIDEO	Shooting location and duration	Video
LIGHTING	open sign	illuminate
PATROL	starting point and ending point	Inspection
SEARCH	Task target information	search region
LOAD	Cargo location	Loading and unloading cargo
TRANSPORT	starting point and ending point	transport cargo
DROP	placement	unload location

(2) State: As shown in Formula (1), the term State refers to the task's allocation status, which can be one of three possible states: Allocated, Unallocated, or Partially Allocated. Notably, terminal tasks do not include the partially allocated state. The allocation status of tasks follows a bottom-up approach, where the allocation of terminal tasks or subtasks influences the status of their parent tasks. Let us assume the task's assignment status is represented by State; its variables can include both non-terminal and terminal tasks.

$$\begin{cases} X \in \{NonTerminal\}, Y \in \{Terminal\} \\ State(X) \in \{Allocated, UnAllocated, PartiallyAllocated\} \\ State(Y) \in \{Allocated, UnAllocated\} \end{cases} \quad (1)$$

(3) Time: Time represents the time information of the mission. Formula (2) defines the constraint relationship of task time. Time sub-elements include $StartTime, EndTime\ and\ EstimatedTime$. The default value of the start time is the current time, and the value needs to be greater than or equal to the current time. The end time T_{end} of the same mission needs to be greater than or equal to the start time T_{start}. The estimated time $T_{estimate}$ needs to be less than or equal to the difference between the end time T_{end} and the start time T_{start}.

$$\begin{cases} Time = \langle StartTime, EndTime, EstimatedTime \rangle \\ T_{end} \geq T_{start} \geq T_{current} \\ T_{estimate} \leq T_{end} - T_{start} \end{cases} \quad (2)$$

(4) Coord: Coord is composed of two sets of triples, representing the longitude, latitude, and altitude of both the entry and exit coordinate points during task execution. The entry coordinate point marks the starting position where the UAV begins its mission, entering the designated mission area. Conversely, the exit coordinate point indicates the position where the UAV leaves the mission area upon completing its task.

(5) Behavior: Behavior refers to the specific actions the UAV performs during a mission, and it is a distinct element of terminal missions. This behavior outlines how the UAV operates within the scenario when executing a terminal mission. As shown in Table 2, Behavior defines the specific actions the UAV needs to perform, such as TAKE_PHOTO, RECORD_VIDEO, and TRANSPORT.

(6) Relationship: Relationship defines the temporal connections between missions. As shown in Fig. 3, these relationships include parallel execution (Cobegin), sequential execution (Sequence), branching execution (Fork), and converging execution (Join). Notably, when a mission is a terminal task—representing the smallest atomic unit—parallel execution is not applicable.

$Sequence(T_i, T_j)$ represents the linear execution relationship between tasks, Task T_j can only start execution after task T_i has been executed. $Cobegin(T_i, T_j)$ represents the relationship between task T_i responsible and task T_j starting at the same time. $Join(T_i, T_j, T_k)$ represents that task T_k can only be executed after task T_i and task T_j have been executed. $Fork(T_i, T_j, T_k)$ represents that after the execution of task T_i is completed, task T_j and task T_k will be executed.

(7) Requirement: Requirement refers to the resources necessary to complete a task and is used to align with the capabilities outlined

Table 3
The meaning of task property elements and their relationship with terminal tasks.

Element	Description	Terminal Tasks	Non-terminal Tasks
State	Task State	✓	✓
Type	Task Type	✓	✓
Time	Task Time	✓	✓
Coord	Task Coord	✓	✓
Relationship	Subtask collaboration relationship	×	✓
Requirement	Task resource requirements	✓	✓
Target	Task Target	✓	×
Behavior	Task Behavior	✓	×

in the resource model. For instance, executing night missions would necessitate UAVs equipped with lighting capabilities.

(8) Target: The Target element defines the specific characteristics of a task's objective and is unique to terminal tasks. It includes attributes such as TargetLocation, TargetType, TargetShape, TargetEvent, TargetSize, and TargetCoords.

(9) Terminal: Terminal indicates whether a task is at its final stage, determining if it qualifies as a terminal task. A terminal mission is an indivisible unit and represents the smallest task a drone can perform. Similar to a leaf node in a tree, it has an out-degree of 0, making it a terminal node. The task meta-model uses the Termination attribute, with a value of 0 or 1, to specify if a task is terminal. A task is classified as terminal if no further tasks follow its completion. Depending on whether a task is terminal, the elements within the task meta-model are categorized into two distinct scenarios. Table 3 outlines the meanings of the task meta-model elements and their relationships to terminal tasks.

3.3. Resource meta-model

UAV resources refer to resources such as drones and related equipment, technology and knowledge needed to complete specific missions. The resource metamodel is defined by six tuples $Resource = \langle Info, State, Domain, Service, Log, Performance \rangle$, where (1) $Info$ refers to the basic details of the resource, encompassing four attributes: $id, name, type$ and $description$. These correspond to the unique identifier, name, type, and description of the resource's fundamental information, respectively. (2) $State$ represents the status information of the resource, which is $idle, inuse, abnormal$ or $invalid$. (3) $Domain$ specifies the domain in which the resource is applicable, with values that include areas like logistics and transportation, search and rescue, and highway inspection. (4) $Service$ provides details about the resource's service capabilities, including the attributes Number and Time, as well as the sub-element TaskList, representing the resource's quantity, available time, and the set of tasks it can perform. Finally, (5) Log contains the resource's historical information, with the attributes Time and Note corresponding to specific past times and their associated records. (6) $Performance$ element represents the performance information of the UAV resources, that is, the UAV's resources, payload resources, etc. that can be called when the UAV performs tasks.

We conducted a thorough manual analysis of the information provided on the UAV's official website and distilled the Performance elements into five key attributes $Performance = \langle Move, Communication, Sense, Payload, Endurance \rangle$. These attributes capture the capabilities demonstrated by UAV resources. Therefore, as shown in Table 4, we summarized the five types of capabilities that UAVs need to have based on the performance of UAVs.

3.4. Constraint meta-model

Constraints during the mission execution of a UAV swarm are crucial for ensuring mission safety and effectiveness. Properly detailing these constraints is essential for optimal scheduling and operation of the UAV swarm. To provide a precise description of the constraints in UAV swarm scenarios, we have identified two primary types: constraints related to UAV resources and constraints imposed by the environment.

Table 4
UAV performance keywords and descriptions.

Performance name	Keywords	Description
MoveAbility	maxAscendingSpeed	maximum ascent speed
	maxDescendingSpeed	maximum descend speed
	maxHorizontalSpeed	maximum horizontal flight speed
	maxWindResistance	maximum wind resistance speed
	maxTakeoffAltitude	maximum takeoff altitude
	maxTiltAngle	maximum tilt angle
	maxRotationSpeed	maximum rotation angular speed
	maxHoveringTime	maximum hover time
CommunicationAbility	GNSS	global navigation satellite system
	workingFrequency	working frequency
	imageQuality	image transmission quality
	imageDelay	image transmission delay
	maxsIgnalRange	maximum signal range
SenseAbility	senseabilityType	sensing system type
	obstacleAvoidance	obstacle avoidance
PayloadAbility	emptyWeight	bare metal weight
	maxPayload	maximum payload
	gimbalQuantity	number of UAV gimbals
EnduranceAbility	maxFilghtRange	maximum flight range
	flighttime	theoretical flight time
	currentFlightTime	current flight time
	remainingFlightTime	remain flight time
	workingTemperature	working temperature
	protection	protection level

(a) Resource constraint: Resource constraints govern the allocation and scheduling of resources during UAV swarm operations, enhancing the accuracy and efficiency of task allocation and path planning. Resource Constraints consist of five-tuples $Resource\_Constraint = \langle Move_c, Communication_c, Sense_c, Payload_c, Endurance_c \rangle$. These constraints are closely tied to the resource model's performance capabilities. The resource model outlines the performance limits of each resource. For instance, if three UAVs have endurance capacities of 2 h, 3 h, and 5 h respectively, and a task requires 3.5 h to complete, the corresponding resource constraint would be 3.5 h. To meet this constraint, we would select a UAV with a flight time of at least 5 h to ensure the task can be successfully executed.

(b) Environment constraint: Environmental constraints provide a detailed representation of the geographical and environmental conditions in which the UAV swarm operates, ensuring safe and efficient performance under specific conditions. Environmental constraints are defined as quadruples $Environment\_Constraint = \langle Obstacle, NoFlyZone, SafeDistance, Weather \rangle$. The following sections will provide a detailed explanation of each environmental element.

(1) Obstacle: An obstacle refers to objects encountered by the drone during flight that cannot be navigated horizontally but can be circumvented by altering the drone's altitude. As defined by the Formula (3), Obstacle elements include coordinates (coord), minimum height (minAlt), and maximum height (maxAlt). Assume that the coordinates of the UAV during the flight are (U_x, U_y, U_z) and the height

range of the obstacles is (Alt_{min}, Alt_{max}). When encountering an obstacle, the UAV needs to lower or raise its height to avoid the obstacle. The z-axis coordinates U_z of the UAV need to be lower than the lowest height of the obstacle ($U_z < Alt_{min}$) or higher than the highest height of the obstacle ($U_z > Alt_{max}$).

$$\begin{cases} Obstacle = \langle coord, minAlt, maxAlt \rangle \\ U_z < Alt_{min}, U_z > Alt_{max} \end{cases} \tag{3}$$

(2) NoFlyZone: A NoFlyZone refers to designated airspace, areas, or specific locations where drones are prohibited from entering or flying over. These no-fly zones can be categorized into three types: rectangular areas (Rectangle), circular areas (Circle), and polygonal areas (Polygon). UAVs must adhere strictly to these restrictions, ensuring they do not enter or fly within the specified longitude, latitude, and altitude limits of the no-fly zone.

(3) SafeDistance: SafeDistance defines the minimum safe distance and altitude limits that must be maintained between the UAV and both external objects and its own flight environment. It is characterized by three parameters: minDis (the minimum safe distance), minAlt (the minimum altitude), and maxAlt (the maximum altitude). The distance between the drone and other objects in the environment is Dis, the current height of the drone is Alt, the minimum safe distance of the drone is $minDis$, and the safe height limits are $minAlt$ and $maxAlt$, then the safety distance constraints $Safe_{distance}$ need to satisfy the following formula:

$$Safe_{distance} = \begin{cases} Dis \leq minDis \\ minAlt \leq Alt \leq maxAlt \end{cases} \tag{4}$$

(4) Weather: Weather conditions can impact various aspects of UAV operations, including flight direction, altitude, and stability. Properly planning for these weather constraints can enhance both the efficiency and safety of the mission. In adverse weather conditions, UAVs may be unable to operate, rendering certain weather-affected areas impassable. Specifically, Weather is defined as a five-tuple $Weather = \langle Type, Duration, Temperature, Covered\,Area, Wind \rangle$, where (1) $Type$ indicates the type of weather conditions encountered by the UAV group during its mission, such as high temperature, low temperature, heavy fog, thunder and lightning, rainfall, strong wind, etc. (2) $Duration$ represents the duration range of the weather. (3) $Temperature$ represents the lowest temperature to the highest temperature, which is related to the UAV's working environment. (4) $Covered\,Area$ represents the weather coverage area. (5) $Wind$ represents the wind level and wind speed under the current weather, which is related to the wind resistance speed in the UAV resource capability.

3.5. Application scenario modeling language

Traditional Domain-Specific Languages (DSLs) are often limited in their ability to address the diverse requirements of UAV swarm scenarios, including missions, resources, and constraints. To overcome these limitations, we have developed an Application Scenario Modeling Language (ASML) designed to describe UAV swarm application scenarios formally. The structure of ASML is based on the meta-models of UAV swarm scenarios.

Specifically, ASML translates the mission meta-model, resource meta-model, and constraint meta-model into Backus–Naur Form (BNF) to establish the corresponding grammatical rules. BNF was selected for its ability to precisely define formal grammar and validate syntax, making it an ideal choice for domain-specific languages like ASML. Its structured approach ensures that the language's syntax is both rigorous and unambiguous, which is critical for the reliable execution of UAV swarm operations. While BNF may pose a steep learning curve for developers unfamiliar with formal grammar specifications, its advantages in providing a clear and consistent framework for language definition outweigh these challenges. Furthermore, comprehensive documentation and examples have been provided to mitigate this barrier.

Table 5
Primitive data types.

Name	Declaration	Value example
Integer	*int*	0,1,−1,77
Identifier	*identifier*	abc,_bc
Double-precision floating-point	*double*	0.0,1.5,−3.14
String	*string*	'abc',"bcd"
Time	*time*	2023-11-23T12:30:45
3D coordinates	*coord3*	(122.4294, 37.7649, 50)
3D coordinate array	*coord3Array*	(0,0,2),(3,4,7)

Additionally, to enhance the language's understandability, scalability, and reusability, ASML incorporates design principles from extensible markup language (XML). XML was chosen over alternatives such as JSON or YAML due to its versatility, robust schema validation support, and hierarchical structure. These features make XML particularly well-suited for complex UAV swarm scenarios, where ensuring strict data integrity and compatibility is paramount. For instance, XML's schema validation capabilities enable precise definitions of allowable structures and constraints, reducing the likelihood of errors during mission planning and execution. Additionally, its hierarchical structure aligns naturally with the nested relationships often found in UAV swarm configurations.

While XML's verbosity and complexity may increase development overhead compared to more lightweight alternatives, these trade-offs are acceptable given its reliability and ability to represent extensive metadata. The decision to use XML ensures that ASML can effectively handle the intricacies of UAV swarm coordination while maintaining a high level of flexibility and extensibility. To further support the adoption of ASML, we have made the lexical and syntax rules, along with usage guidelines, publicly available (Renliang, 2024) for reference.

3.5.1. Lexical rules

ASML defines basic lexical and grammatical rules. Lexical rules include components, data types, keywords, etc. The components clarify the meaning and composition relationship of each part of the scene modeling language. The data type represents the different types and formats of data and clarifies the storage form of the data. By designing the corresponding vocabulary, ASML keywords obtain the part-of-speech tags and their meanings of application scenarios by designing corresponding vocabulary lists.

ASML is composed of three parts: Mission Description Language, Resource Description Language, and Constraint Description Language. A scenario represents a situation, which refers to the description of missions, resources, and constraints of a drone swarm under specific conditions (such as highway inspection). 'Mission' refers to the global tasks within the scenario. 'Resource' refers to the resources such as payloads and drones that the drone swarm possesses while performing tasks in this scenario. 'Constraint' refers to the environmental constraints that the drones face while executing tasks in this scenario. An application scenario is defined as follows:

$$\langle Scenario \rangle ::= \langle Mission \rangle \langle Resource \rangle \langle Constraint \rangle$$

Data types are used to define the nature and value rules of data, determining the range of values that data can store. ASML includes seven data types, with Table 5 showing the specific type names and their corresponding examples:

3.5.2. Syntax rules

Syntax rules use the Backus–Naur Form (BNF) to define syntax rules for scene elements, their tasks, resources, and constraint sub-elements and attributes. BNF provides a more formal grammatical description structure system. As a metalanguage specifically used to define languages, it has the characteristics of concise syntax, and clear expression, and is conducive to syntax analysis and compilation. BNF can express

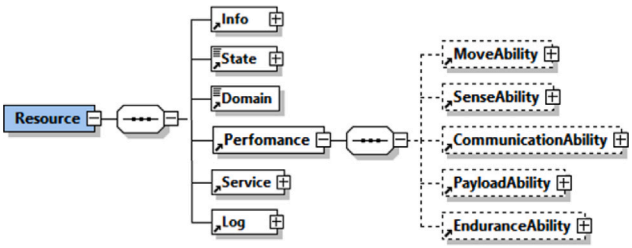

Fig. 4. Resource XML Schema.

grammar rules canonically, and the grammar it presents does not depend on a specific context. ASML uses XML Schema diagrams to intuitively express each element's attributes and sub-element composition in the scene model.

Application Scenario Mission Syntax Rules: In ASML, a scenario mission is defined using the ⟨*Mission*⟩ element. The ⟨*Mission*⟩ element includes basic information such as id, name, and describe, as well as sub-elements for scenario mission attributes ⟨*MissionProperty*⟩ and a task set ⟨*Task*⟩, where the ⟨*Task*⟩ element can contain one or more instances. The ⟨*MissionProperty*⟩ element includes the sub-elements ⟨*MissionState*⟩, ⟨*MissionType*⟩, ⟨*MissionTime*⟩, ⟨*MissionCoord*⟩, ⟨*MissionRelationship*⟩, and ⟨*MissionRequirement*⟩, each of which must appear exactly once. The values of the elements and attributes within <Mission> are restricted by syntax rules. Specifically, the value of ⟨*MissionState*⟩ must be of string type and limited to one of the following: "Allocated", "Unallocated", or "PartiallyAllocated". The value of ⟨*MissionType*⟩ must be of string type and restricted to one of the following: "Logistics", "Agriculture", "Rescue", or "HighwayPatrol". The <MissionTime> element includes three sub-elements: ⟨*StartTime*⟩, ⟨*EndTime*⟩, and ⟨*EstimatedTime*⟩. The ⟨*MissionCoord*⟩ element includes two sub-elements: ⟨*EnterPoint*⟩ and ⟨*LeavePoint*⟩. The value of ⟨*MissionRelationship*⟩ must be one of the following: ⟨*Sequence*⟩, ⟨*CoBegin*⟩, ⟨*Fork*⟩, or ⟨*Join*⟩. The ⟨*MissionRequirement*⟩ element contains one or more ⟨*ResourceType*⟩ sub-elements.

$$\langle Mission\rangle ::= \langle MissionProperty\rangle\langle Task\rangle^{+}$$
$$\langle MissionProperty\rangle ::= \langle MissionState\rangle\langle MissionType\rangle\langle MissionTime\rangle$$
$$\langle MissionCoord\rangle\langle MissionRelationship\rangle\langle MissionRequirement\rangle$$
$$\langle MissionState\rangle ::= Allocated \mid Unallocated \mid PartiallyAllocated$$
$$\langle MissionType\rangle ::= Logistics \mid Agriculture \mid Rescue \mid HighwayPatrol$$
$$\langle MissionTime\rangle ::= \langle StartTime\rangle\langle EndTime\rangle\langle EstimatedTime\rangle$$
$$\langle MissionCoord\rangle ::= \langle EnterPoint\rangle\langle LeavePoint\rangle$$
$$\langle MissionRelationship\rangle ::= \langle Sequence\rangle \mid \langle CoBegin\rangle \mid \langle Fork\rangle \mid \langle Join\rangle$$
$$\langle MissionRequirement\rangle ::= \langle ResourceType\rangle^{+}$$

Resource Syntax Rules: In ASML, a resource is defined using the ⟨*Resource*⟩ element. The resource syntax inherits the structure of the resource metamodel. A resource description consists of 6 elements. As shown in Fig. 4, the resource's XML Schema defines the legal building blocks of the XML document and defines the performance sub-elements. The specific syntax rules of resources are as follows:

$$\langle Resource\rangle ::= \langle Info\rangle\langle State\rangle\langle Domain\rangle\langle Performance\rangle\langle Service\rangle\langle Log\rangle$$

Constraint Syntax Rules: Constraints include two types of constraints: resource constraints and environment constraints. Resource constraints describe the resource capabilities required by the UAV to perform its mission. Environmental constraints describe the corresponding mission environment information when the UAV performs its mission. The specific constraint syntax is described as follows:

$$\langle Constraint\rangle ::= \langle Resource\_Constraint\rangle|\langle Environment\_Constraint\rangle$$
$$\langle Resource\_Constraint\rangle ::= \langle Move\rangle\langle Communication\rangle\langle Sense\rangle\langle Payload\rangle\langle Endurance\rangle$$
$$\langle Environment\_Constraint\rangle ::= \langle WayPoints\rangle\langle Obstacle\rangle\langle NoFlyArea\rangle\langle SafeDistance\rangle\langle Weather\rangle$$

Table 6
Error message field and number.

Error message field	Number	Description
NOT_FOUND_MISSION	1000001	Mission element not found.
MULTIPILE_MISSION	1000002	There are multiple Mission elements.
UNKNOWN_ELEMENT	1000003	Element is undefined.
EMPTY_ATTRIBUTE	1000004	The attribute value is empty.
INVALID_CONTENT	1000005	The content is invalid.
EMPTY_CONTENT	1000006	The content is empty.
INVALID_FORMAT	1000007	The format is invalid.
MULTI_ELEMENT	1000008	There are multiple elements present.
MULTI_ATTRIBUTE	1000009	Multiple properties exist.
NOT_FOUND_ATTRIBUTE	1000010	Property not found.
NOT_FOUND_ELEMENT	1000011	Element not found.

In addition, in order to reduce the difficulty of writing the language, we developed a graphical interface tool for the application scenario modeling function. The graphical interface can realize low-code programming and can generate and parse application scenario modeling language according to user configuration. This tool is based on the Qt Creator integrated development environment, uses the C++ programming language combined with the MSVC compiler and provides support for tool interface design through Qt Design. However, the application scenario model for interface drawing may have errors when parsed into ASML. For this reason, as shown in Table 6, we have defined common parsing error types to facilitate developers to locate and fix errors promptly.

3.6. Parse model

The scenario parsing module is used for parsing the modeling language of application scenarios for drone swarms. This includes the parsing of the expected element set and the expected attribute set. The expected element set comprises all possible element types that may appear during parsing, while the expected attribute set is formed by the collection of attributes for each element in the expected element set. The parsing steps are as follows: first, parse all elements to form the expected element set; next, parse each element in the expected element set to obtain all attributes of the elements, forming the expected attribute set; then, determine whether the content of the elements meets the format requirements and store the information; finally, based on the parsing results, return the successful parsing result or error information as the scenario parsing data.

Algorithm 1 illustrates the file parsing process of the application scenario modeling language. **Step 1: Global Parsing**. First, verify whether the drone swarm application scenario modeling file complies with XML syntax standards, including basic rules such as tag closure and element name matching. **Step 2: Element and Attribute Parsing**. Validate whether the file conforms to the semantics and specifications of the application scenario modeling language designed in this study, ensuring that the elements or attributes belong to the expected element set or expected attribute set. **Step 3: Content Parsing.** Ensure that the element content exists and meets the corresponding format requirements. For example, the time within the ⟨*Time*⟩ tag must comply with the ISO8601 date-time format. **Step 4: Sub-element Parsing**. Repeat the above parsing steps to continue parsing the sub-elements of each element until all elements are parsed and validated.

4. Verification method

The UAV swarm verification ensures the safety, efficiency, and compliance of task execution through formal methods. Fig. 5 shows the verification process of the drone swarm application scenario. First, the system begins with an application scenario model that captures the behavior and interaction patterns of the drone swarm in specific tasks, such as flight path planning, task allocation, and communication modes. Subsequently, this scenario model is parsed to extract key scenario elements,

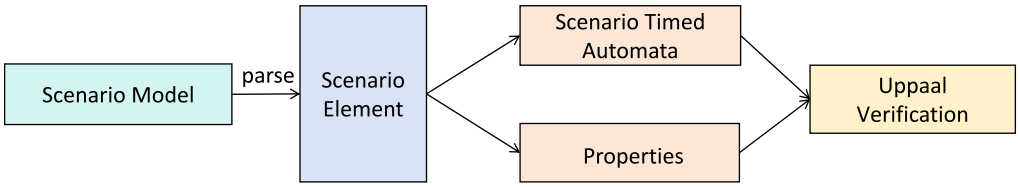

Fig. 5. Scenario verification method framework.

Algorithm 1 File parsing process of the application scenario modeling language.

Input: XMLElement: The XML document node
Output: Parse Result

```
if XMLElement == NULL then
    return Results
end if
if !isVaild(XMLElement) then
    return Errors
end if
if XMLElement → name ∉ propertyset then
    return Errors
end if
for attribute in XMLElement → attributes do
    if attribute ∉ propertyset then
        return Errors
    end if
end for
if isVaild(XMLElement → content) then
    return Errors
end if
for element in XMLElement → elements do
    if !findFunction(element → name) then
        return Errors
    else
        parseXML(element)
    end if
end for
```

including the states of the drones, dependencies between tasks, and time constraints associated with those tasks. These elements are then transformed into a timed automata model, which accurately describes the state transitions of the drones during task execution along with their time-related limitations.

Next, a set of properties is defined to constrain the behavior of the drone swarm. These properties typically encompass temporal requirements (e.g., task completion deadlines), safety requirements (e.g., minimum safety distances between drones), and performance requirements (e.g., communication latency and resource utilization efficiency). These properties are expressed in formal logic to ensure that the task execution of the drone swarm aligns with expectations across various scenarios.

Using the Uppaal verification tool, the framework conducts formal verification of the timed automata. Uppaal can execute temporal logic queries to verify whether the drone swarm can complete tasks within specified time limits, whether task allocation is reasonable, whether communication is synchronized in a timely manner, and whether potential collisions or task failures can be avoided. This entire process provides robust safety and correctness assurances for the task execution of the drone swarm.

4.1. Scenario element extraction

The extraction of scenario elements is a critical step in translating high-level scenario models into precise, actionable components for formal verification. The goal is to decompose the abstract description of the drone swarm's tasks, interactions, and environmental constraints into concrete elements that can be used to construct a timed automata model. This process ensures that the various aspects of drone behavior, such as task execution, state transitions, and inter-drone communications, are accurately captured and formally represented.

First, task decomposition breaks down complex mission objectives into smaller, more manageable sub-tasks. Each sub-task is treated as an independent element, with clearly defined start and end conditions, as well as specific time and resource requirements. For instance, a mission involving environmental monitoring might be decomposed into subtasks such as entering the monitoring zone, performing the monitoring, and exiting the zone. Each of these stages becomes a distinct element in the model, enabling more precise control over task execution.

Next, the state of each drone is extracted to capture the various stages of task execution, such as "idle", "in-flight", or "performing task". These states are critical for accurately modeling the dynamic behavior of the drones within the swarm. Additionally, interaction and synchronization events between drones are identified, especially in scenarios that require collaboration, such as data sharing or coordinated task execution. These interactions are formalized as events in the model to ensure proper synchronization and communication during mission execution.

Another crucial component of scenario element extraction is the identification of time constraints. Timely execution is often essential in drone swarm missions, and these constraints are represented through timed automata's clock variables and guards. For example, if a monitoring task must be completed within a certain time frame, this constraint is captured and enforced in the model.

Finally, external conditions and environmental interactions are extracted. Drones not only interact with each other but also with the external environment, which may include obstacles, weather conditions, or dynamic mission parameters. These elements are modeled to represent external influences on the swarm's task execution. Additionally, dependencies between tasks are identified and extracted, ensuring that task sequencing and synchronization are accurately reflected in the formal model.

4.2. Transformation rules for timed automata

Scenario element extraction provides the foundation for constructing timed automata by decomposing high-level system behaviors—such as tasks, drone states, interactions, time constraints, and environmental conditions—into precise, formal components. These elements are mapped directly onto the timed automata model, where tasks are represented as states, interactions as synchronization events, and time constraints as clock variables and guard conditions. Task dependencies and state transitions are formalized into automaton transitions, ensuring that the dynamic behavior and temporal requirements of the drone swarm are accurately captured. This structured extraction-to-conversion process ensures that the system's real-world behavior is faithfully modeled, enabling thorough analysis during formal verification while maintaining consistency between the scenario model and the timed automata. As shown in Fig. 6, the three component models of the application scenario – task model, resource model, and constraint model – can be converted into corresponding timed automaton networks according to the conversion rules.

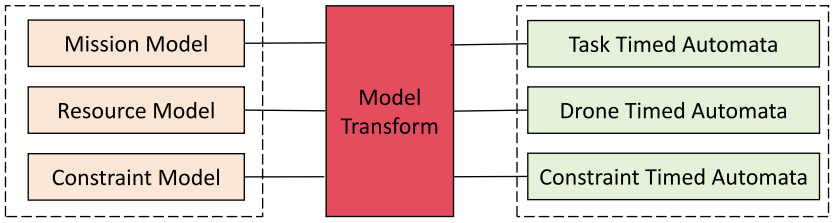

Fig. 6. Timed Automata Transformation Rules.

The timed automaton network is composed of Task timed automata, drone timed automata and constrained timed automata connected to each other. The state of each timed automaton is abstracted from the behavior of the system. The transition conditions of task automata are mainly composed of the time consumption between tasks, so the transition conditions of task time automata are extracted and converted according to the time-related attributes of the Mission model. Transitions in the drone automaton are determined by the power-related properties in the resource model. In contrast, the constraint model's endurance governs transitions in the constraint automaton on time and power.

4.2.1. Model transformation rules

Conversion of mission model to task-timed automata. The mission model describes the order, dependencies, duration and other information of tasks. Convert the mission model into a task time automaton, mainly by defining the various states of the task and their transformation relationships.

For the mission model, it contains a task set $\{t_1, t_2, \ldots, t_n\}$, each of these tasks contains the following properties: task start state, task execute state, task end state, task duration and task dependencies relationship. The conversion process from a task model to a task-time automaton first requires initializing the task-time automaton. The initialization of the task-time automaton is to create the state of the task-time automaton for each task t_i. These states include the task's state attributes, start state, execution state and end state. Next, the task's time constraints and dependencies are converted into the transition of task time automata. If the execution time of the task is within the time window constraint, the migration is from task start to task execution. If the completion of the task is still less than the end time specified by the task, it means that the task has been completed—transition from task execution state to task end state. Dependencies between tasks are realized through the synchronization of task time automata. If there is a timing relationship (T_i, T_j) between task T_i and task T_j, then the task time automaton start state of T_j must wait until the end of task T_i before it can start execution.

Conversion of resource model to drone-timed automata. The drone model captures the resource dynamics, such as drone availability, task execution, and battery management. The transformation process generates Drone Timed Automata (DTA) for each drone, ensuring that drone states align with task execution needs. The resource model includes a UAV set $D = \{d_1, d_2, \ldots, d_n\}$, each characterized by Idle state, Cruising state and Inspecting state. These states initialize the drone timed automaton as the state of each drone timed automaton. Next, the power constraints of the resource model are extracted as the transition conditions of the UAV time automaton. If the power can meet the task execution requirements, the drone will migrate from idle state to cruising state. In the cruising state, if the drone's power is sufficient to meet the mission execution requirements, it will reach the mission execution state.

Conversion of constraint model to constraint-timed automata. The constraint model defines temporal and dependency restrictions on task execution. The transformation generates Constraint Timed Automata (CTA) ensuring tasks are executed within defined limits. The resource model includes constraint set $C = \{c_1, c_2, \ldots, c_n\}$. These constraints include time, battery power, and no-fly zone limits. For each constraint c_i, corresponding states and transitions are created according to different constraint types. For time limit type and no fly zone constraints c_i, add time constraints in task automata and constrained automata: if a task t_i must be completed within time t_{limit}, add a transition time constraint. If the task fails to be completed within the specified time, then trigger an Error state. For the power type constraint c_i, add power constraints to the drone automaton and constrained automata: if a drone must be within the power p_{limit} to complete the task, add a transition power constraint. If the drone fails to complete the task, If the task is executed under the specified power, the Error state will be triggered.

4.2.2. Synchronization rules for timed automata.

Task-Drone Synchronization. Task Start: For each task t_i in the Task timed automata, ensure synchronization with the corresponding drone state d_j. When the task automaton is in the starting state, the drone automaton needs to be synchronized to the Cruising state. Upon task completion, synchronize the transition to the drone's idle state,

Task-Constraint Synchronization, The execution of the task is controlled by time and no-fly zone constraints, and the task automata and the constraint automata must trigger the transition at the same time. The execution of the task is controlled by time constraints, and the task automaton and the constrained automaton must trigger the transition at the same time. When the task automaton is in the starting state, the constrained automaton is also in the running state. If the task is not completed within the time limit, the task automaton enters the Error state and the constraint automaton enters the TimeoutError state. No-fly zone constraints and time constraints have similar synchronization rules.

Drone-Constraint Synchronization. If the drone's status is limited by battery power, etc., it must be synchronized to the status of the battery error in the constrained automaton.

4.3. Property extraction and verification with UPPAAL

The process of property extraction is crucial for defining the system's behavioral and temporal constraints, which serve as formal specifications for verification. Properties typically encompass timing requirements, safety conditions, and performance metrics, all of which are derived from the drone swarm's mission and operational context. These properties are expressed in temporal logic, allowing them to be rigorously tested using model-checking tools like UPPAAL. By formalizing these requirements, the system's expected behaviors—such as completing tasks within deadlines, maintaining safe distances between drones, and ensuring synchronized communication—are made explicit and ready for verification.

In the drone swarm verification framework, several key properties must be extracted and formally verified to ensure safe, efficient, and correct system behavior. These properties, expressed in temporal logic, fall into two main categories: safety properties, liveness properties. Each type of property addresses a specific aspect of the system's operation, and their verification guarantees that the system adheres to the required constraints in various mission scenarios.

Safety Properties. Safety properties ensure that the drone swarm operates without violating critical safety constraints, such as preventing collisions or maintaining safe operational boundaries. These properties define the conditions under which tasks can be executed safely, such as ensuring drones maintain a minimum distance from each other to avoid collisions, or ensuring that critical system checks are completed before initiating flight. Safety properties are crucial for preventing hazardous states in the system, and are typically expressed as invariants that must always hold during mission execution.

In addition to ensuring collision avoidance and system consistency, safety properties in the drone swarm framework can include constraints on time windows and operational conditions under low power. For instance, safety properties can enforce that tasks are completed within a specified time window, preventing drones from running tasks that extend beyond their operational deadlines. Furthermore, low-power safety properties ensure that drones do not engage in or continue tasks when their battery levels fall below a predefined threshold, thereby preventing failures due to power depletion. These properties are crucial for maintaining system robustness, ensuring that drones operate within safe temporal and energy constraints, and avoid scenarios that could lead to mission failure or unsafe behavior.

Liveness Properties. Liveness properties guarantee that the system will eventually progress and complete its tasks, avoiding deadlock or indefinite waiting. These properties ensure that tasks, once started, will be completed, and that the system will continue to transition between states as expected. For example, a liveness property might specify that a task will eventually be finished, ensuring that the drone swarm does not get stuck in an incomplete or idle state. Liveness properties are essential for ensuring that the system remains functional and capable of achieving its objectives.

Once the properties are extracted, the UPPAAL verification process begins. UPPAAL, a model-checking tool for timed automata, is used to validate that the system satisfies the extracted properties under all possible scenarios. This is done by encoding the system model and its properties into UPPAAL's query language, where temporal logic expressions are used to check the model's adherence to the specified requirements. For instance, a timing property might be checked by querying whether a specific task is always completed within its deadline, while safety properties can be verified by ensuring that drones never violate the minimum separation distance during their operations.

UPPAAL's model-checking algorithm exhaustively explores all potential states of the system model, analyzing whether the system can violate any of the specified properties under any conditions. If a violation is detected, UPPAAL provides a counterexample, illustrating the sequence of events leading to the failure. This feedback allows designers to refine the system model or modify its properties, thereby improving the overall design. The verification process ensures that the system meets its timing, safety, and performance requirements, providing strong assurances of its reliability and correctness before deployment.

5. Case study

5.1. Highway inspection scenarios

Highway inspection refers to the process of surveying and monitoring highways and their associated facilities to evaluate their condition. Fig. 7 shows a scenario of a highway inspection mission. The highway inspection scenarios can be categorized into two types: road inspection and slope inspection. These tasks have a sequential relationship and are performed sequentially. This scenario contains five tasks, namely three road inspections and two slope inspections. In addition, this scenario also includes a no-fly zone outside the inspection mission, which represents the area that the drone needs to avoid entering when performing its mission. Due to the wide distribution and complex environmental structure of highways, a single UAV faces challenges such as long operation times and low efficiency during inspections. Therefore, in practice, a multi-UAV cooperative approach is required for highway inspections. In this paper, we developed detailed models of highway inspection scenarios and rigorously validated the flight mission planning results. Our work emphasizes the accuracy and effectiveness of the proposed planning strategies, ensuring that the models align with real-world conditions and deliver optimal performance in practical highway inspection operations.

Table 7
Comparison of different modeling languages.

Language	Task	Resource	Resource constraint
GML (Jia et al., 2022)	✓	×	×
SL4U (Zhao et al., 2024)	✓	×	×
ASML	✓	✓	✓

5.1.1. Language comparison

We compare the descriptive capabilities of the ASML UAV scene modeling language with those of two other general UAV swarm modeling languages. As illustrated in Table 7, ASML offers detailed descriptions of resource capabilities and constraints. In contrast, the other two languages are limited to describing task execution within logistics scenarios and lack the ability to provide a granular depiction of resource capabilities and constraints.

5.1.2. Mission

This mission model outlines a drone-based highway patrol mission, which is scheduled to begin at 5:00 a.m. on August 28, 2024, with a planned duration of 1.5 h. The drone will navigate along a pre-defined route while carrying a camera for inspection purposes. It will execute the assigned tasks in a specified sequence. As illustrated in Fig. 8, the inspection paths for the road and slope inspection tasks overlap in certain areas. To resolve this overlap, a sequential dependency is introduced between the tasks. Specifically, task 3 can only be initiated once task 0 has been completed, and task 4 depends on the successful completion of task 2. This approach ensures that there is no task interference during the mission and that resources are allocated efficiently.

Table 8 provides a comprehensive summary of the key components of the highway inspection scenario. The overall mission consists of three road inspection tasks and two slope inspection tasks. The road inspection tasks are classified as line-type tasks, where the drone follows linear paths along the highway, while the slope inspection tasks are surface-type tasks, requiring a broader coverage of sloped areas adjacent to the highway. The road inspection tasks are estimated to take approximately 25 min each, whereas the slope inspection tasks are projected to last around 30 min each. Collectively, these tasks are designed to be completed within the allocated 1.5-hour mission window, ensuring that the drone efficiently patrols and inspects both road and slope areas without exceeding the time constraints.

5.1.3. Resource

Fig. 9 illustrates the comprehensive resource model essential for the drone highway inspection mission. This model provides a detailed overview of the status and performance capabilities of the quadcopter UAV, referred to as UAV0. The drone is equipped with a maximum horizontal flight speed of 25 meters per second, enabling efficient coverage of long highway sections within a short time frame. Its communication system supports a maximum signal range of 10.5 kilometers, ensuring that the drone remains in constant contact with the control center or monitoring personnel over a significant operational radius.

In terms of payload, UAV0 can carry up to 1.5 kilograms, which is sufficient to support the necessary mission equipment, including a high-definition camera for continuous monitoring and filming. The drone's endurance is another critical factor, with a flight duration of up to 45 min on a single charge or fuel cycle. This endurance is aligned with

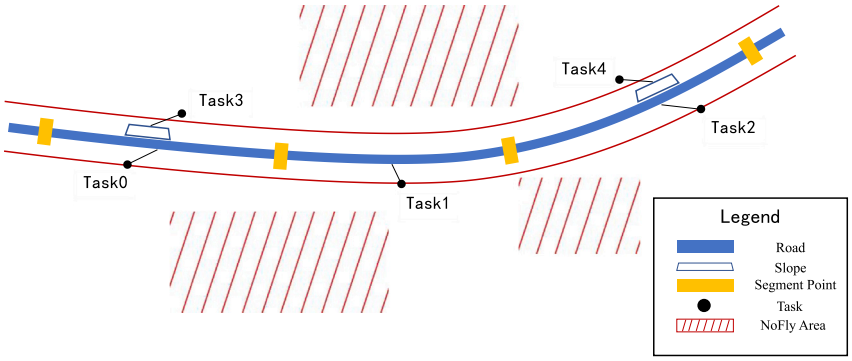

Fig. 7. Set up for highway inspection scenario.

```
<Mission id="0" name="mission_highway" describe="Patrols of highway">
    <MissionProperty>
        <MissionState>Allocated</MissionState>
        <MissionType>HighwayPatrols</MissionType>
        <MissionTime>
            <StartTime>2024-08-28T05:00:00Z</StartTime>
            <EndTime>2024-08-28T06:00:00Z</EndTime>
            <EstimatedTime>1.5h</EstimatedTime>
        </MissionTime>
        <MissionCoord>
            <EnterPoint longitude="108.629" latitude="34.089" altitude="413.956" />
            <LeavePoint longitude="108.658" latitude="34.097" altitude="414.608" />
        </MissionCoord>
        <MissionRelationship>
            <Sequence Tasks="task0,task3" />
            <Sequence Tasks="task2,task4" />
        </MissionRelationship>
        <MissionRequirements>
            <ResourceDomain>HighwayPatrols</ResourceDomain>
            <ResourcePayload>Camera</ResourcePayload>
        </MissionRequirements>
    </MissionProperty>
</Mission>
```

Fig. 8. ASML for mission.

Table 8
Composition of highway inspection scenario tasks.

Task attribute	Task0	Task1	Task2	Task3	Task4
Task State	allocated	allocated	allocated	allocated	allocated
Task Type	Road	Road	Road	Slope	Slope
Task Time	05:00-05:25	05:25-05:50	05:00-05:25	06:00-06:30	06:00-06:30
Task Terminal	1	1	1	1	1
Task Requirement	Camera	Camera	Camera	Camera	Camera
Task Behavior	TAKE_PHOTO	TAKE_PHOTO	TAKE_PHOTO	TAKE_PHOTO	TAKE_PHOTO
Task Target	line	line	line	plane	plane

the planned inspection tasks, allowing the drone to perform substantial portions of the highway patrol without the need for frequent recharging or refueling.

The service window for the drone is also a vital consideration in mission planning. UAV0 is available for deployment between 4:00 a.m. and 4:00 p.m. daily, providing a 12-hour operational window. This schedule ensures that all mission-critical tasks can be carried out during daylight hours, optimizing visibility for both road and slope inspections. The resource model is designed to facilitate efficient utilization of UAV0 within these time constraints, ensuring that the drone's capabilities are maximized while adhering to the required time frame.

5.1.4. Constraint

The drone highway inspection constraint model is composed of three critical components: endurance, waypoints, and safe distance. These constraints are essential to ensure the successful and safe execution of drone-based highway inspection missions, providing a framework for mission planners to optimize resource utilization, enhance operational efficiency, and mitigate potential risks.

The endurance constraints outline the drone's operational limits in terms of flight time and power consumption for different inspection tasks. Specifically, the model defines distinct endurance parameters for road inspection and slope inspection missions. As shown in Fig. 10, for

```
<Resource>
    <Info id="0" name="uav0" type="Quadcopter" description="uav0 resource" />
    <State>Available</State>
    <Domain>HighwayPatrols</Domain>
    <Performance>
        <MoveAbility>
            <maxHorizontalSpeed>25.0</maxHorizontalSpeed>
        </MoveAbility>
        <CommuniactionAbility>
            <maxSignalRange>10.5km</maxSignalRange>
        </CommuniactionAbility>
        <PayloadAbility>
            <maxPayload>1.5kg</maxPayload>
            <missionPayload>Camera</missionPayload>
        </PayloadAbility>
        <EnduranceAbility>
            <maxFlightTime>45min</maxFlightTime>
        </EnduranceAbility>
    </Performance>
    <Service sid="0" time="04:00:00-16:00:00" />
</Resource>
```

Fig. 9. Resource information for UAV0.

road inspections, the maximum flight time is limited to 25 min, with a corresponding power consumption of 8000 mA. In contrast, slope inspections allow a slightly longer flight duration of 30 min, with a power consumption of 9000 mA. These endurance parameters are crucial for mission planning, as they directly impact task scheduling and battery management. Since the drone's power reserves are finite, careful planning is necessary to ensure that the tasks are completed within the available flight time and energy limits. If these constraints are not properly managed, there is a risk of mission failure due to the drone running out of power mid-flight, which would not only disrupt the mission but also potentially endanger the drone's integrity.

Waypoint constraints define the geographical path the drone must follow during the inspection mission. Each waypoint is associated with specific longitude, latitude, and altitude coordinates that mark the key points along the drone's route. Additionally, the connectivity matrix for each waypoint specifies its link to other waypoints, ensuring the drone follows a structured and logical flight path. This connectivity ensures that the drone moves seamlessly between pre-defined waypoints, reducing the chances of deviating from the mission path or encountering obstacles. Furthermore, the connectivity matrix allows for flexible path planning, where the drone can adapt to specific mission requirements or environmental factors, such as avoiding restricted areas or adjusting the flight route in response to real-time data. The waypoint constraints also facilitate efficient coverage of the inspection area, allowing for more comprehensive monitoring of both the road surface and surrounding infrastructure, such as slopes and embankments.

The safe distance constraints play a critical role in ensuring the drone's operational safety during the mission. The model defines the minimum horizontal distance and the minimum and maximum flight altitudes the drone must maintain throughout its flight. The minimum horizontal distance is set at 0.5 m, ensuring that the drone remains at a safe distance from obstacles, such as buildings, bridges, or other vehicles on the highway. Additionally, the model specifies a minimum altitude of 5 meters and a maximum altitude of 200 m, which ensures the drone stays within safe airspace boundaries while avoiding interference with ground activities and other aerial operations. These altitude constraints are particularly important in urban or semi-urban highway environments, where airspace can be congested, and infrastructure, such as overpasses or power lines, pose potential hazards. By adhering to these safe distance parameters, the drone can operate without risking collisions or violating aviation regulations.

5.2. Experiment platform and result

To verify the validity of the scenarios constructed by ASML, this paper configures a UAV simulation environment and conducts corresponding real-aircraft testing. The experiment's hardware and software setup includes an Intel(R) Core(TM) i7-9700 CPU, 16 GB of memory, Ubuntu 20.04 as the operating system, Robot Operating System (ROS) Noetic as the robotic middleware, and Gazebo 11.11.0 for the simulation environment.

Additionally, we leverage the concept of digital twins to simulate UAV models, ensuring that the simulation models closely approximate real UAVs. Specifically, we utilize the P600-Allapark2-RTK-G1-S3 UAV model, independently developed by the AMOVLAB. The modeling process comprehensively considers various aspects of the real UAV, including geometry, physics, flight control, flight performance, and endurance capabilities. This approach enables highly accurate replication of the real UAV's behavior within the simulation environment.

For the experimental task scenarios, we selected a section of a highway for inspection and modeled it to ensure task diversity. Following an on-site survey, we modeled inspection scenarios for three highway segments and two slopes. Additionally, we employed a multi-UAV collaborative inspection strategy to enhance the operational complexity of the UAVs, ensuring a realistic and challenging simulation environment.

The simulation results are presented in Fig. 11, the UAV team (UAV1, UAV2 and UAV3) will work together to complete the mission and eventually return to the designated point. As depicted in Fig. 11, for the inspection of a highway, two drones are located at different locations to inspect the road and slopes. Two drones will not simultaneously perform slope and road inspections of the same segment to prevent collisions. After flying through the entire simulation planning process, the drone successfully completed the assigned scene tasks without collision.

5.3. Verification

5.3.1. UPPAAL models

We convert the logistics UAV swarm application scenario model into a UPPAAL timed automaton network. The Timed Automata Network of the entire drone application scenario system (TANDAS) can be represented as the inner product of timed automata processes. Based on

```
<Constraint>
    <Endurence>
        <time type="road">25min</time><power type="road">8000mA</power>
        <time type="slope">30min</time><power type="slope">9000mA</power>
    <Endurence>
    <WayPoints>
        <WayPoint id="WP001" longitude="108.629" latitude="34.089" altitude="413.956">
            <Connectivity>0,1,0,0,1,0,0,0</Connectivity>
        </WayPoint>
        <WayPoint id="WP002" longitude="108.640" latitude="34.090" altitude="414.967">
            <Connectivity>1,0,1,0,0,1,0,0</Connectivity>
        </WayPoint>
        <WayPoint id="WP003" longitude="108.649" latitude="34.093" altitude="412.137">
            <Connectivity>0,1,0,1,0,0,1,0</Connectivity>
        </WayPoint>
        <WayPoint id="WP004" longitude="108.658" latitude="34.097" altitude="414.608">
            <Connectivity>0,0,1,0,0,0,0,1</Connectivity>
        </WayPoint>
        <WayPoint id="WP005" longitude="108.633" latitude="34.089" altitude="413.460">
            <Connectivity>1,0,0,0,0,1,0,0</Connectivity>
        </WayPoint>
        <WayPoint id="WP006" longitude="108.635" latitude="34.089" altitude="414.850">
            <Connectivity>0,1,0,0,1,0,0,0</Connectivity>
        </WayPoint>
        <WayPoint id="WP007" longitude="108.653" latitude="34.095" altitude="410.912">
            <Connectivity>0,0,1,0,0,0,0,1</Connectivity>
        </WayPoint>
        <WayPoint id="WP008" longitude="108.654" latitude="34.095" altitude="411.977">
            <Connectivity>0,0,0,1,0,0,1,0</Connectivity>
        </WayPoint>
    </WayPoints>
    <SafeDistance id="sd1" name="Safe Zone">
        <minDis>0.5</minDis>
        <minAltitude>5.0</minAltitude>
        <maxAltitude>200.0</maxAltitude>
    </SafeDistance>
</Constraint>
```

Fig. 10. ASML for constraint.

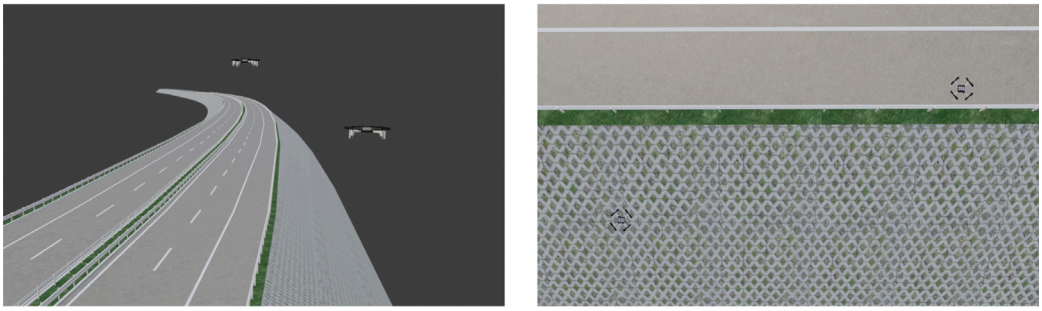

Fig. 11. Schematic diagram of highway scenario simulation.

the understanding and abstraction of UAV swarm application scenarios, the task planning and execution process of a UAV application scenario can be described as a timed automaton network. As shown in Formula (5), TANDAS consists of a drone network composed of three timed automata: the task timed automata, the drone timed automata, and the constraint timed automata.

$$TANDAS = TaskTA \| DroneTA \| ConstraintTA \tag{5}$$

5.3.2. Task timed automata

The task timed automaton simulates the process of task execution in the drone application scenario. As shown in Fig. 12, the task timed automata consists of three states: Ready, Executing, and Error. The running process of the automaton is that the tasks have been assigned first and are initially in the Ready state waiting to be executed. When a task is executed, it enters the Executing state, and the task execution time continuously increases. If the task completion time is less than the task end time, it returns to the Ready state to proceed with the next task. Otherwise, if the task exceeds its execution deadline, it enters the error state due to timeout, and interacts with the constraint automaton, sending a timeout error signal. Below we introduce task-time automata's state and the meaning of transitions.

S1: Ready State: A state in which task assignment is completed but execution has not yet started. In this state, the mission planning result

Table 9
Transition descriptions of task-timed automata.

Number	Transition	Description
1	S1→S1	This transition involves the initialization of task attributes, including task allocation results and task execution time.
2	S1→S1	This transition checks task dependencies to ensure they are satisfied.
3	S1→S2	Once execution conditions are met, the task transitions to the execution state.
4	S2→S1	The task is completed by DroneTA within the specified time window and returns to the Ready state.
5	S1→S1	After all tasks in the UAV group application scenario are completed, ConstraintTA resets the tasks, and TaskTA reinitializes them to evaluate the planning results for the next application scenario.
6	S1→S3	During the waiting process, if the task remains in a Ready state and the overall execution time exceeds the defined time window (endTime), a timeout error occurs, leading to the Error state and triggering a timeout exception signal to ConstraintTA.
7	S2→S3	If the task execution time exceeds the endTime during its operation, the system transitions to the Error state and sends a timeout exception signal to ConstraintTA.
8	S1→S3	The system receives an exception signal broadcast by ConstraintTA, indicating a drone-related issue, which interrupts task execution and causes the system to enter the Error state.
9	S2→S3	The system receives an exception signal broadcast by ConstraintTA, indicating a drone-related issue, which interrupts task execution and causes the system to enter the Error state.
10	S3→S1	In the event of an Error state, the system resets and re-executes the scenario task from the beginning.

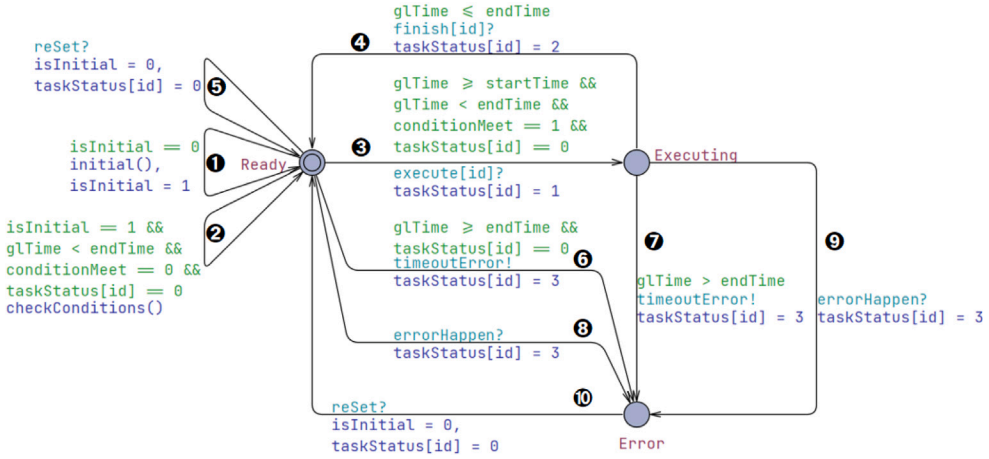

Fig. 12. Highway inspection scenario task timed automata.

assigns a specific UAV to the task, and the task will wait for its own time constraints and dependency constraints to be satisfied.

S2: Executing State: The task is being executed by the specified drone. In this state, it indicates that the mission is proceeding normally and the drone is performing operations according to the mission requirements.

S3: Error State: The state of abnormal arrival when the scene is running. In this state, it indicates that there is a problem with the task planning, and the specific error cause is triggered from the migration condition and broadcast to the constrained automaton.

Task time automata state transitions can be classified into two main categories: those that occur during normal task execution and those that occur during abnormal execution. Transitions 1–5 are associated with normal task execution, and transitions 6–10 represent relevant transitions that occur when task execution times out. The specific transition meaning is shown in the Table 9.

5.3.3. Drone timed automata

The drone timed automata simulates the dynamic process of battery power changes during the drone's task execution. To perform tasks, UAVs need to go through three states: Idle, Cruising, and Inspecting. As shown in Fig. 13, the drone is first in the Idle state, waiting to take off to perform its mission. Then the drone enters the cruising state and reaches the mission location that the drone needs to perform. Finally, the drone performs the assigned tasks in the inspection state. During the flight of the UAV, the battery power changes dynamically with the flight. When the UAV is in low battery, it enters the Error state and synchronously sends a low battery error signal to the constrained timed automata. The detailed description of the state and transition of the drone time automaton is as follows:

S1: Idle State: The task has been assigned to the drone, but the drone has not started taking action. In this state, the mission planning results assign the drone a specific list of tasks to be performed. After starting the action, the drone is ready to take off.

S2: Cruising State: The drone is flying along a preset path in the air. In this state, it indicates that the drone is cruising between tasks,

Table 10
Transition descriptions of drone-timed automata.

Number	Transition	Description
1	S1→S1	The system initiates the drone's attribute parameters, including power levels and other essential configurations.
2	S1→S2	The drone ascends and transitions into a cruising state.
3	S2→S3	Upon completing the cruise, the drone reaches the task execution location, where it sends an execution signal and updates its battery status simultaneously.
4	S3→S2	After the current task is completed, the task list is updated, preparing the UAV to execute the subsequent assigned task.
5	S2→S2	Update the position, time consumption, and power consumption of the UAV based on predefined waypoints, and determine whether the current location is within a no-fly zone to update the inNoFlyArea flag accordingly.
6	S2→S1	Upon completion of all tasks, the system returns to the Idle state and broadcasts a check signal to ConstraintTA to verify the completion of the overall plan.
7	S1→S1	Once the plan is confirmed as completed, ConstraintTA initiates a system reset, leading to the reset and reinitialization of DroneTA.
8	S2→S4	Upon receiving a system exception signal broadcast by ConstraintTA, the system transitions to the Error state. A task timeout will cause the entire system to terminate.
9	S2→S4	When inNoFlyArea == 1, indicating that the current location is within a no-fly zone, the system enters the Error state and sends a no-fly error signal to the ConstraintTA.
10	S2→S4	If the drone's battery power falls below the LowPower threshold after task execution, the system transitions to the Error state, and a low power exception signal is sent to ConstraintTA.
11	S3→S4	If the battery power is below the LowPower threshold after completing the cruise, the system enters the Error state, and a low power anomaly signal is transmitted to ConstraintTA.
12	S3→S4	Upon receiving a system exception signal broadcast by ConstraintTA, the system transitions to the Error state. A task timeout will cause the entire system to terminate.
13	S4→S1	In the Error state, ConstraintTA initiates a system reset, which synchronously resets DroneTA and triggers reinitialization.

that is, from the execution area of one task to the execution area of another task.

S3: Inspecting State: The drone is performing a mission. This state indicates that the drone is performing the tasks assigned by the mission planning.

S4: Error State: The state reached when the drone operates abnormally.

The transition of DroneTA illustrates the changes in the drone's power state as it executes its mission. Table 10 shows the implications of drone automaton transition. Transitions 1–6 represent the state transformations that occur during normal mission execution by the UAV. In contrast, Transitions 7–11 describe the state transitions triggered by abnormal conditions, such as low battery, encountered during the mission.

5.3.4. Constraint timed automata

The constraint timed automata describe the operation of the entire drone swarm application scenario. As shown in Fig. 14, the drone swarm is first in the Running state. If all tasks are finally completed, the constrained automaton reaches the Finish state. Otherwise, if a timeout or low power error occurs during task execution, it will enter the Timeout state and LowPower state respectively. The meaning of each state and transaction is introduced below.

S1: Running State: The drone and mission are in normal operating conditions. The UAV swarm application scenario task has been assigned to the UAV, and the UAV is performing the task normally.

S2: Finish State: A state in which the drone successfully completes all tasks. In this state, the drone completed all application scenario tasks according to the planned results, and no errors occurred.

S3: Timeout State: The task is not completed within the specified time. Each task has a specified start time and end time. If the task is completed beyond the end time, a timeout error will occur.

S4: NoFlyArea State: If a flight deviation occurs during the UAV's approach to the task execution point, it will enter a no-fly zone.

S5: LowPower State: The battery of the drone is insufficient to continue the mission. Under normal circumstances, a drone should have sufficient power to complete its assigned tasks. However, in abnormal situations, the drone may encounter insufficient power, resulting in unfinished tasks.

As shown in Table 11, the timed automata effectively manage the drone's various states during the inspection mission through state transitions, including normal operation, task completion, and error handling. This ensures the smooth execution of the mission and timely recovery from any faults or errors.

5.3.5. Property verification

We use UPPAAL's model validator to verify the properties of logistics drone swarm temporal automata networks. UPPAAL is one of the most widely used model checking tools for timed automata. UPPAAL uses TCTL to define the syntax of the property verification specification language. Verifiers only need to write corresponding query statements to verify the properties of the system.

Specifically, we verify two major types of properties: safety, and liveness. All verification properties are represented using TCTL. We cannot verify the collaborative properties of multiple UAVs, such as two UAVs moving an object simultaneously. Below we describe in detail the properties that require verification.

Safety: Safety properties mean that something bad will never happen. In the UAV swarm application scenario, due to the dependencies

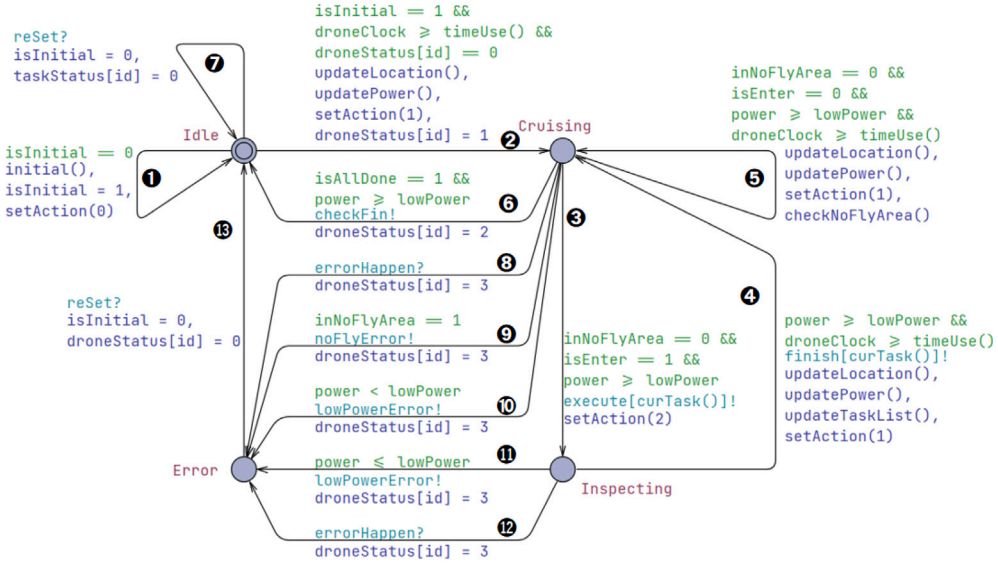

Fig. 13. Highway inspection scenario drone timed automata.

Table 11
Transition descriptions of constraint-timed automata.

Number	Transition	Description
1	S1→S1	The system receives a check signal from DroneTA to verify whether all planned tasks have been completed.
2	S1→S2	If all tasks are successfully completed, the system transitions to the Finish state, indicating the successful execution of all plans.
3	S2→S1	If not, the system resets, continues executing the newly planned application scenario, and broadcasts the reset signal to both DroneTA and TaskTA.
4	S1→S3	Upon receiving a timeout exception signal from TaskTA, the system transitions to the Timeout state, indicating task execution has timed out.
5	S3→S3	The system broadcasts an errorHappen signal upon detecting an abnormal situation, causing the entire system to transition to the error state and halt the current application scenario.
6	S3→S1	After an exception, the system resets and re-executes the application scenario automata network.
7	S1→S4	Upon receiving the no-fly exception signal from DroneTA, the system enters the NoFlyArea exception state.
8	S2→S2	The errorHappen signal is broadcast, and the entire system transitions to the Error state.
9	S4→S1	In the exception state, the ConstraintTA resets the system.
10	S1→S5	If DroneTA signals a low battery, the system enters the error state.
11	S5→S5	The system broadcasts an errorHappen signal upon detecting an abnormal situation, causing the entire system to transition to the error state and halt the current application scenario.
12	S5→S1	After an exception, the system resets and re-executes the application scenario automaton network.

between tasks and execution time window constraints, if the execution of the previous task exceeds the planned execution time, the start execution time of subsequent tasks will not be met, resulting in a time conflict. Therefore, Formula (6) verifies whether there is a timeout during task execution, thereby determining whether task planning is proceeding as expected without time conflicts causing collisions.

$$E\langle\rangle\ ConstraintTA.Timeout \tag{6}$$

Formula (7) verifies that the UAV will not be in a low power state during mission execution. During the flight of the drone, the power is continuously consumed as the task is performed. When the drone is allocated, the power is used to evaluate whether it can complete the task. Therefore, the battery of the drone should not be low during the mission, which will lead to safety issues such as being unable to return.

$$E\langle\rangle\ ConstraintTA.LowPower \tag{7}$$

Formula (8) verifies that the drone will not enter the no-fly zone during its mission. This constraint is crucial to ensure the drone's operational safety, particularly when performing inspections or other tasks near sensitive or hazardous areas. The no-fly zone, often established due to regulatory restrictions, hazardous conditions, or potential risks to the drone or surrounding environment, must be avoided at all times.

$$E\langle\rangle\ ConstraintTA.NoFlyArea \tag{8}$$

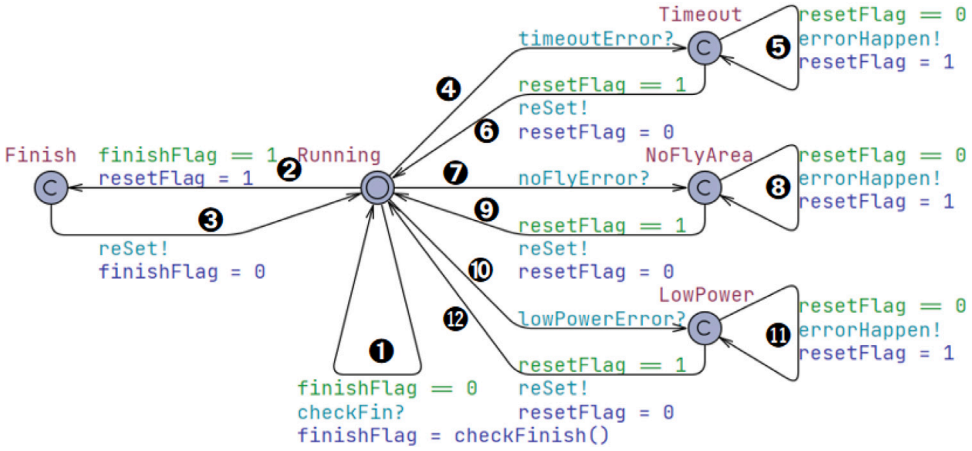

Fig. 14. Highway Inspection scenario constraint timed automata.

Table 12
UPPAAL verification results.

Property type	Formula number	Verification results
Safety	(15)	not Satisfy this property
	(16)	not Satisfy this property
	(17)	not Satisfy this property
Liveness	(18)	Satisfy this property

Liveness: Liveness properties indicate that something good will eventually happen. Formula (9) indicates that for all tasks planned in the UAV swarm application scenario, the UAV swarm can eventually complete it.

$$E\langle\rangle\ ConstraintTA.Finish \tag{9}$$

Table 12 shows the verification results of the above properties. Since the verification of security is to set an unsafe state in the timed automaton, it is judged whether the state is reachable by traversing the automaton. If it is unreachable, the system is safe. For the low-power and timeout unsafe states in the automaton, verification shows that the properties are not met, that is, the state is unreachable. This shows that drone scenario planning is safe. For the liveness properties, the verifier returned a result that satisfies the attribute, indicating that the UAV group scenario mission planning can be finally completed.

6. Related work

6.1. Single UAV mission description

A UAV swarm consists of numerous individual UAVs, making the description of each UAV's mission fundamental to the swarm as a whole. To capture the wide range of tasks carried out by individual UAVs across various applications, researchers often employ specific task descriptions to outline the roles and actions of these UAVs.

The individual task description was originally framed using the waypoint list method (Brumitt and Stentz, 1998), which treats each task as a waypoint that the robot must pass through, along with specifying the action to be performed at that location. However, this approach is rigid and struggles to adapt to dynamic mission environments, especially when obstacles are present. To address this, Konolige (1997) developed the COLBERT robot control language, utilizing a Finite State Machine (FSM) structure with iteration, sequencing, and conditional logic. Building on this concept, Tousignant et al. (2012) introduced the XRobots language, which employs hierarchical state machines and defines states as robot behaviors. This approach resolves the scalability and maintenance issues inherent in FSMs as the complexity and number of robot behaviors increase.

To enhance the adaptability of UAVs in dynamic environments, Molina et al. (2017) developed a task-based mission specification language (TML) specifically for search and rescue operations. TML organizes tasks hierarchically using a tree structure. Building on this concept, Lan et al. (2018) adopted behavior trees as the core framework for representing and executing complex task plans. Similarly, Li et al. (2019) introduced a real-time sensing UAV mission system integrated with the ROS, where behavior trees are employed as decision-making mechanisms to manage both flight behaviors and contextual changes in real-time.

In single-drone tasks, task description languages often exhibit rigidity in expression, hindering the drone's ability to adapt to changing task environments and leading to limitations such as inflexible route adjustments. In contrast, our proposed description language is specifically designed for UAV swarms, providing enhanced capabilities for complex task descriptions along with comprehensive resource and environmental representations.

6.2. UAV swarm mission description

The UAV swarm mission description language provides a framework for defining collaborative tasks among multiple UAVs, detailing the actions each UAV must perform, as well as the interdependencies between these actions. Approaches for describing UAV swarm missions can be categorized into three types: quasi-programming languages, declarative markup languages, and graphical interfaces.

Programming-like language: Programming-like languages resemble traditional programming languages in their syntax and structure but are designed for purposes other than general computer programming. These languages allow users to define domain-specific rules, logic, or procedures, enabling the programmatic expression of complex tasks or operations. Merino Merino et al. (2005) introduced a framework for collaborative fire detection using heterogeneous UAV formations, applied within the multi-UAV project COMETS, though UAVs had to be adapted to function within this framework. Dantu et al. (2011) developed Karma, a system for programming and managing micro-UAV swarms. Dedousis and Kalogeraki (2018) presented PaROS (PROgramming Swarm), a framework for programming both UAV swarms and individual UAVs, offering developers abstract swarm programming primitives to simplify drone swarm control and reduce the complexity of low-level programming. Similarly, Mottola et al. (2014) proposed the VOLTRON team-level programming model, which dynamically allocates tasks to UAVs based on mission requirements. Pinciroli and Beltrame (2016) introduced Buzz, a language designed for large-scale, heterogeneous robot clusters. While programming-like languages are highly expressive and feature-rich, they are not ideal for rapid deployment in dynamic environments.

Declarative markup language: In the aviation sector, XML has increasingly become the standard for data exchange, particularly in the SESAR air traffic management modernization program across Europe and the United States (SESAR Joint Undertaking et al., 2019). Consequently, declarative markup languages for task description are predominantly implemented using extensible markup language (XML) (Bray et al., 1997). Doherty et al. (2010) developed a task specification language based on Task Specification Trees (TST), which was applied to UAV collaborative systems. Additionally, Bozhinoski et al. (2015) introduced domain-specific languages for drone swarms, such as the Monitoring Missions Language (MML) and Quadrotor Behavior Language (QBL).

Silva et al. (2014) created a set of languages for describing multi-robot tasks, including the XML-based Mission Description Language (MDL). MDL focuses on specifying mission areas, actions, sequences, time constraints, and UAV requirements. To address potential disruptions in multi-robot tasks, such as human or environmental interference, Silva et al. (2016) later introduced the Disturbance Description Language (DDL). Additionally, Castro Silva et al. (2017) proposed two more XML-based languages: Scenario Description Language (SDL) and Team Description Language (TDL), which serve as static representations of scene and task knowledge. SDL defines the physical environment and global operational constraints, while TDL outlines vehicle teams and their specific constraints. To accommodate the demands of UAV swarms in
multi-task scenarios, Jia Jia et al. (2022) developed a UAV swarm mission model for dynamic tasks across various environments and introduced the XML-based Group Mission Language (GML) for cluster task descriptions. Moreover, Zhao et al. (2024) proposed SL4U, a UAV swarm description language that categorizes UAV scenarios into environments and tasks. Although declarative markup languages offer excellent readability, they tend to be less adaptable across different platforms and application areas.

Graphical interface: Graphical interface tools are often used to define drone missions and create basic flight plans, such as setting waypoints for a drone to follow a predetermined route. UAV manufacturers like Parrot and DJI have developed proprietary graphical tools (Glossner et al., 2021), but these tools are limited to their own products and are incompatible with drones from other brands. For more complex mission planning, FlyMASTER (Lamping et al., 2018) offers a software platform aimed at researchers working on UAV swarm systems, enabling rapid development, flexible integration, and use. However, its technical complexity makes it accessible mainly to domain experts, with limited usability for non-technical users. Ruscio et al. (2016) introduced MML, a monitoring task language designed for non-experts, and implemented FLYAQ, a platform with a graphical interface for defining monitoring tasks. Nevertheless, FLYAQ lacks features such as automatic detection of regions with complex geometries and the ability to visualize or plan three-dimensional flight paths (Besada et al., 2018). To address these limitations, Besada et al. (2018) developed a mission definition system that supports both pre-flight mission visualization and trajectory prediction. Although graphical interfaces provide intuitive and user-friendly interaction, they are often tightly coupled with specific applications and can be difficult to adapt for other use cases.

While these tools demonstrate excellent performance in specific applications, they are often tightly coupled with their original contexts, which limits their flexibility in adapting to diverse usage scenarios and requirements. This constraint becomes evident when addressing complex and dynamic task demands. Furthermore, the necessity to manage numerous intricate data interactions and support various file formats adversely affects their performance and efficiency. In contrast, our proposed task description language is designed with dynamic expressiveness, empowering UAV swarms to respond effectively to emergencies and navigate diverse operational contexts.

7. Conclusion

In order to accurately describe UAV swarm application scenarios, we propose a modeling method based on meta-level theory. We characterize UAV application scenarios with mission, resource, and constraint models. Furthermore, to formally represent the proposed modeling approach, we construct and implement an Application Scenario Modeling Language (ASML). ASML refers to the description method of extensible markup language XML and defines the corresponding lexical and BNF syntax rules. ASML can be further converted into timed automata to verify its correctness. We verify the reliability of the proposed modeling method in a logistics handling scenario.

CRediT authorship contribution statement

Manqing Zhang: Writing – original draft, Conceptualization. **Yunwei Dong:** Supervision, Methodology, Funding acquisition. **Tao Zhang:** Writing – review & editing. **Kang Su:** Software, Formal analysis. **Zeshan Li:** Resources, Project administration.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was supported by the National Key R&D Program of China under the grant number 2022YFB4501800.

Data availability

No data was used for the research described in the article.

References

Arnold, R., Jablonski, J., Abruzzo, B., Mezzacappa, E., 2020. Heterogeneous UAV multi-role swarming behaviors for search and rescue. In: 2020 IEEE Conference on Cognitive and Computational Aspects of Situation Management. CogSIMA, IEEE, pp. 122–128.

Besada, J.A., Bergesio, L., Campaña, I., Vaquero-Melchor, D., López-Araquistain, J., Bernardos, A.M., Casar, J.R., 2018. Drone mission definition and implementation for automated infrastructure inspection using airborne sensors. Sensors 18 (4), 1170.

Bozhinoski, D., Di Ruscio, D., Malavolta, I., Pelliccione, P., Tivoli, M., 2015. Flyaq: Enabling non-expert users to specify and generate missions of autonomous multicopters. In: 2015 30th IEEE/ACM International Conference on Automated Software Engineering. ASE, IEEE, pp. 801–806.

Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., Yergeau, F., 1997. Extensible markup language (XML). World Wide Web J. 2 (4), 27–66.

Brumitt, B.L., Stentz, A., 1998. GRAMMPS: A generalized mission planner for multiple mobile robots in unstructured environments. In: Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No. 98CH36146), Vol. 2. IEEE, pp. 1564–1571.

Castro Silva, D., Henriques Abreu, P., Reis, L.P., Oliveira, E., 2017. Development of flexible languages for scenario and team description in multirobot missions. AI EDAM 31 (1), 69–86.

Dantu, K., Kate, B., Waterman, J., Bailis, P., Welsh, M., 2011. Programming micro-aerial vehicle swarms with karma. In: Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems. pp. 121–134.

Dedousis, D., Kalogeraki, V., 2018. A framework for programming a swarm of UAVs. In: Proceedings of the 11th Pervasive Technologies Related To Assistive Environments Conference. pp. 5–12.

Doherty, P., Heintz, F., Landén, D., 2010. A distributed task specification language for mixed-initiative delegation. In: International Conference on Principles and Practice of Multi-Agent Systems. Springer, pp. 42–57.

Fowler, M., 2010. Domain-Specific Languages. Pearson Education.

Glossner, J., Murphy, S., Iancu, D., 2021. An overview of the drone open-source ecosystem. arXiv preprint arXiv:2110.02260.

Javaid, S., Saeed, N., Qadir, Z., Fahim, H., He, B., Song, H., Bilal, M., 2023. Communication and control in collaborative UAVs: Recent advances and future trends. IEEE Trans. Intell. Transp. Syst..

Jia, W., Ni, J., Yang, G., Wang, R., Yao, Y., Wu, W., 2022. Design and implementation of task description language for UAV swarms. In: 2022 IEEE International Conference on Unmanned Systems. ICUS, IEEE, pp. 158–164.

Khan, N.A., Jhanjhi, N., Brohi, S.N., Usmani, R.S.A., Nayyar, A., 2020. Smart traffic monitoring system using unmanned aerial vehicles (UAVs). Comput. Commun. 157, 434–443.

Kölbl, M., Leue, S., Wies, T., 2020. Tartar: A timed automata repair tool. In: Computer Aided Verification: 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21–24, 2020, Proceedings, Part I 32. Springer, pp. 529–540.

Konolige, K., 1997. Colbert: A language for reactive control in sapphira. In: KI-97: Advances in Artificial Intelligence: 21st Annual German Conference on Artificial Intelligence Freiburg, Germany, September 9–12, 1997 Proceedings 21. Springer, pp. 31–52.

Kühne, T., 2006. Matters of (meta-) modeling. Softw. Syst. Model. 5, 369–385.

Lamping, A.P., Ouwerkerk, J.N., Cohen, K., 2018. Multi-UAV control and supervision with ROS. In: 2018 Aviation Technology, Integration, and Operations Conference. p. 4245.

Lan, M., Xu, Y., Lai, S., Chen, B.M., 2018. A modular mission management system for micro aerial vehicles. In: 2018 IEEE 14th International Conference on Control and Automation. ICCA, IEEE, pp. 293–299.

Li, G.-Y., Soong, R.-T., Liu, J.-S., Huang, Y.-T., 2019. UAV system integration of real-time sensing and flight task control for autonomous building inspection task. In: 2019 International Conference on Technologies and Applications of Artificial Intelligence. TAAI, IEEE, pp. 1–6.

Lomonaco, V., Trotta, A., Ziosi, M., Avila, J.D.D.Y., Díaz-Rodríguez, N., 2018. Intelligent drone swarm for search and rescue operations at sea. arXiv preprint arXiv:1811.05291.

Merino, L., Caballero, F., Martinez-de Dios, J., Ollero, A., 2005. Cooperative fire detection using unmanned aerial vehicles. In: Proceedings of the 2005 IEEE International Conference on Robotics and Automation. IEEE, pp. 1884–1889.

Molina, M., Suarez-Fernandez, R.A., Sampedro, C., Sanchez-Lopez, J.L., Campoy, P., 2017. TML: a language to specify aerial robotic missions for the framework Aerostack. Int. J. Intell. Comput. Cybern. 10 (4), 491–512.

Mottola, L., Moretta, M., Whitehouse, K., Ghezzi, C., 2014. Team-level programming of drone sensor networks. In: Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems. pp. 177–190.

Parton, K., McKeown, K., Coyne, R.E., Diab, M.T., Grishman, R., Hakkani-Tür, D., Harper, M., Ji, H., Ma, W.Y., Meyers, A., et al., 2009. Who, what, when, where, why? comparing multiple approaches to the cross-lingual 5W task.

Pinciroli, C., Beltrame, G., 2016. Buzz: An extensible programming language for heterogeneous swarm robotics. In: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS, IEEE, pp. 3794–3800.

Renliang, W., 2024. Application scenario modeling language. https://github.com/PiedPiper911/ASML.

Ruscio, D.D., Malavolta, I., Pelliccione, P., Tivoli, M., 2016. Automatic generation of detailed flight plans from high-level mission descriptions. In: Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems. pp. 45–55.

SESAR Joint Undertaking, et al., 2019. SESAR joint undertaking: single programming document 2020–2022. EU: European Union.

Silva, D.C., Abreu, P.H., Reis, L.P., Oliveira, E., 2014. Development of a flexible language for mission description for multi-robot missions. Inform. Sci. 288, 27–44.

Silva, D.C., Abreu, P.H., Reis, L.P., Oliveira, E., 2016. Development of a flexible language for disturbance description for multi-robot missions. J. Simul. 10 (3), 166–181.

Tousignant, S., Van Wyk, E., Gini, M., 2012. Xrobots: A flexible language for programming mobile robots based on hierarchical state machines. In: 2012 IEEE International Conference on Robotics and Automation. IEEE, pp. 1773–1778.

Wen, J., He, L., Zhu, F., 2018. Swarm robotics control and communications: Imminent challenges for next generation smart logistics. IEEE Commun. Mag. 56 (7), 102–107.

Wenxiu, P., 2015. Analysis of new media communication based on Lasswell's "5w" model. J. Educ. Soc. Res. 5 (3), 245–250.

Wu, X., Liu, Y., Xie, S., Guo, Y., 2020. Collaborative defense with multiple USVs and UAVs based on swarm intelligence. J. Shanghai Jiaotong Univ. (Science) 25, 51–56.

Yang, L., Hu, Z., Long, J., Guo, T., 2011. 5W1H-based conceptual modeling framework for domain ontology and its application on STPO. In: 2011 Seventh International Conference on Semantics, Knowledge and Grids. IEEE, pp. 203–206.

Yu, Y., Bi, Y., 2010. A study on "5w1h" user analysis on interaction design of interface. In: 2010 IEEE 11th International Conference on Computer-Aided Industrial Design & Conceptual Design 1, Vol. 1. IEEE, pp. 329–332.

Zhao, Y., Yao, Y., He, T., Zhou, X., Shen, B., 2024. Sl4u: a scenario description language for unmanned swarm. J. Supercomput. 80 (4), 5363–5389.

Zhong, Y., Ye, S., Liu, Y., Li, J., 2023. A route planning method for UAV swarm inspection of roads fusing distributed droneport site selection. Sensors 23 (20), 8479.

Zhou, Y., Rao, B., Wang, W., 2020. UAV swarm intelligence: Recent advances and future trends. Ieee Access 8, 183856–183878.

Manqing Zhang received the master's degree from Northwestern Polytechnical University, China. He is currently a Ph.D. student with the School of Software, Northwestern Polytechnical University. His research interests include formal verification, program synthesis.

Yunwei Dong received the doctor's degree from Northwestern University, China. His research interests include embedded systems, information-physical convergence systems, trusted software design and verification, and intelligent software engineering.

Tao Zhang received the B.S. degree in automation, the M.Eng. degree in software engineering from Northeastern University, China, and the Ph.D. degree in computer science from the University of Seoul, South Korea. After that, he spent one year with the Hong Kong Polytechnic University as a postdoctoral research fellow. Currently, he is an associate professor with the School of Computer Science and Engineering, Macau University of Science and Technology (MUST). Before joining MUST, he was the faculty member of Harbin Engineering University and Nanjing University of Posts and Telecommunications, China. He is a senior member of IEEE and ACM.

Kang Su received the bachelor's degree from Northwestern Polytechnical University, China. He is currently a master student with the School of Software, Northwestern Polytechnical University. His research interests include formal verification and software engineering.

Zeshan Li received the bachelor's degree from Northwestern University, China. He is currently a master student with the School of Software, Northwestern Polytechnical University. His research interests include reinforcement learning and software engineering.