
Article:
Baek, Y.-M. orcid.org/0000-0002-9796-3756, Cho, E., Shin, D. orcid.org/0000-0002-0840-6449 et al. (1 more author) (2024) An extensible modeling method supporting ontology-based scenario specification and domain-specific extension. International Journal of Software Engineering and Knowledge Engineering, 34 (1). pp. 91-162. ISSN 0218-1940

https://doi.org/10.1142/s021819402350047x

An Extensible Modeling Method Supporting Ontology-based Scenario Specification and Domain-specific Extension

Young-Min Baek*[1], Esther Cho[1], Donghwan Shin[2], and Doo-Hwan Bae[1]

[1]KAIST, Republic of Korea
[2]University of Sheffield, United Kingdom

Abstract

Scenario-based techniques, also known as scenario methods, have been actively employed to resolve intricate problems for engineering complex software systems. Scenarios are powerful tools that allow engineers to analyze the dynamics and contexts of complex systems. Despite the widespread use, there is a lack of a well-established reference framework that systematically organizes key concepts and attributes of scenarios. This has left engineers without a systematic guidance at the method level, hindering their ability to utilize the scenario methods effectively. To address the challenges associated with scenario methods, this study aims to provide a reference framework and modeling method. By conducting a literature review and suggesting a *Conceptual Scenario Framework (CSF)*, we establish a conceptual basis that systematically presents the core concepts and characteristics of scenarios. Additionally, we introduce the *Extensible Scenario Modeling Method (ESMM)* that empowers engineers to perform scenario modeling and domain-specific extensions using the framework. With the inclusion of the *Extensible Scenario Modeling Language (ESML)*, which comprises domain-general model types and classes for scenario description and ontological analysis, *ESMM* facilitates flexible design of domain-specific scenario elements through language-level extensions. This study assesses the proposed method in comparison to existing scenario development methods in the automated driving system domain. Through an analysis of their ability to represent scenario data, it was established that the language constructs of *ESML* possess semantic expressiveness suitable for serving as a reference framework. Furthermore, the findings from the case study validate the extensibility of *ESMM* for specialization in creating a scenario modeling language tailored to specific domains, while also effectively supporting the ontological analysis of particular application domains.

1 Introduction

1.1 Research Background

Scenarios are versatile. They serve as user-friendly artifacts that enhance comprehension and communication by providing intuitive stories involving plausible flows and contexts. Despite originating from the film industry, the concept of '*scenario*' has gained widespread acceptance and usage in various aspects of life, industries, and academia. In the field of software and systems engineering, scenarios have become a familiar technique employed as artifacts or tools to capture and convey specifications [15], fostering better understanding among diverse stakeholders from different backgrounds [16]. This growing significance of well-crafted scenarios is particularly evident as software systems grow increasingly complex. Leveraging the adaptability and versatility of scenarios, as depicted in Figure 1, they have become a frequently utilized approach for addressing intricate problems and specifying requirements throughout the software/system development process [6, 10], as Figure 2 shows.

*Corresponding Author: baekym@kaist.ac.kr

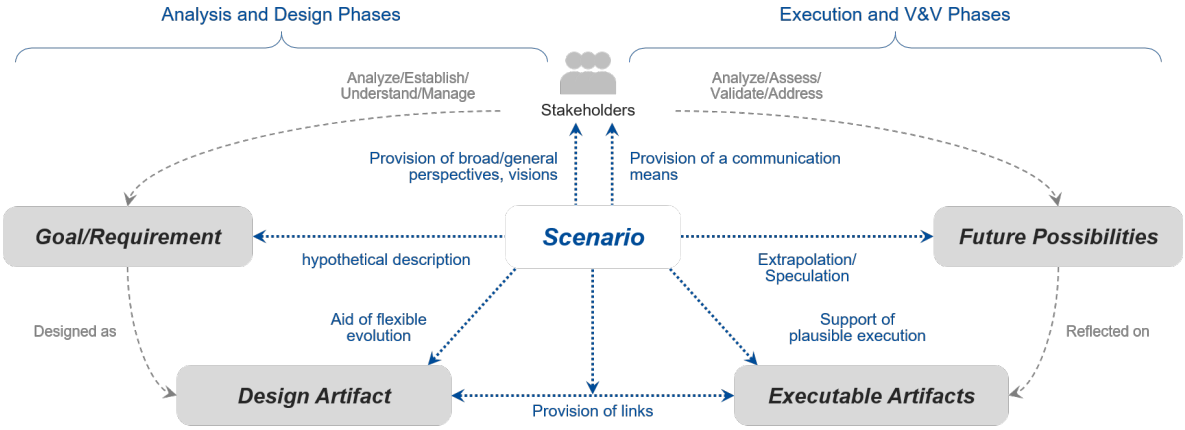

Figure 1: Roles and advantages of utilizing scenarios for engineering purposes

Scenario-based engineering approaches, known as *scenario methods*, have gained significant traction in the software industry. These methods serve as crucial tools for a range of engineering purposes, including scenario-based analysis and design [15, 16, 28], simulation [25, 32, 35], and testing [27]. Unlike traditional specifications that focus on specific system aspects, scenarios have the unique ability to incorporate diverse information, possibilities, and relevant contexts. This versatility of scenarios enables more effective communication and decision-making processes, making them valuable for engineers. Scenarios often go beyond being mere models or documents; they have a profound connection with the goals and requirements of the systems they represent. As a result, scenarios offer several advantages, such as providing clear explanations of specifications and resolving inconsistencies among development artifacts [21]. They act as a medium or proxy for enhancing the readability and comprehensibility of specifications, allowing stakeholders to grasp the essence of the system more intuitively.

Scenarios plays a vital role in analyzing and verifying the criticality of software/systems. Critical systems, such as military, aviation, or automotive systems, require accurate and reliable responses in diverse situations, making systematic scenario development essential. These scenarios not only validate functional requirements but also aid in identifying defects and their causes. For this purpose, scenarios are used to analyze system behavior under specific contexts, supporting future designs and decision making. In addition, scenarios can serve as efficient and user-friendly means of communication, enabling stakeholders and engineers to build consensus during the initial stages of development based on shared understanding of the desired features.

1.2 Problem Statement

P1: Lack of a Reference Framework that Establishes a Common Understanding of Scenarios and Scenario Methods. Scenarios have garnered increased attention as crucial artifacts due to the escalating complexity of system specifications and the challenges associated with engineering intricate contextual information. This trend is evident in Figure 2, which illustrates the growing number of studies related to scenarios or employing scenarios in the software/systems engineering domain over the past two decades. It is important to note that this figure represents the results of a search conducted on *Scopus* using the query ((ALL("software engineering" OR "system* engineering") AND ALL("scenario*")) AND PUBYEAR > 1999)[1]. Scenarios have not only been extensively employed in the domain of requirements engineering but also find application in various other engineering activities, including simulation, testing, verification, training, and decision making. Consequently, the formats and constructs of scenarios exhibit high levels of diversity, and numerous description methods

[1]It does not imply that each publication necessarily develops or utilizes an independent scenario specification method

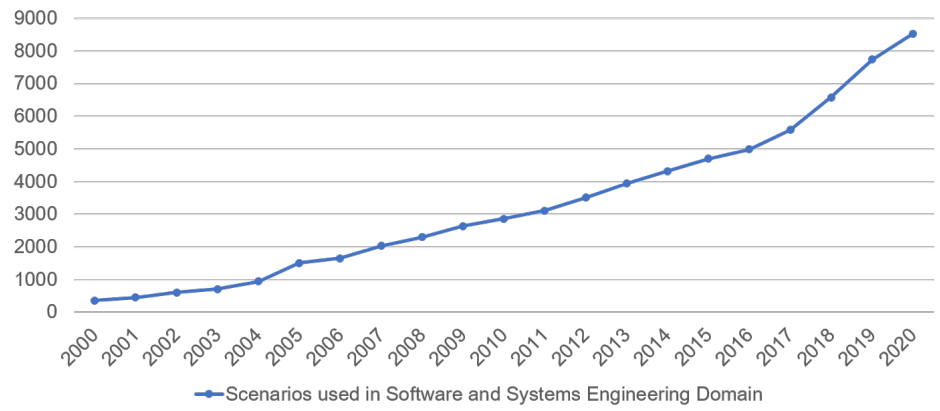

Figure 2: Increasing trend of scenario methods in software and systems engineering fields

(i.e., modeling/specification methods and languages) have been utilized. Despite efforts to formalize domain-specific scenarios, the theoretical or conceptual basis and shared understanding of scenario methods are still lacking [43, 24, 45]. In order to effectively utilize scenario methods, engineers involved in scenario development need to possess a comprehensive understanding of scenarios and scenario methods. Existing studies have presented scenarios developed in various manners, encompassing differences in purpose, semantics, formats or syntax, and the underlying definitions/rules of scenarios. This abundance of published definitions for the term "scenario" spans a wide range of domains, making it imperative to establish commonly-shared concepts and characteristics of scenarios through research on existing scenario methods [1, 38].

One major challenge is the lack of a conceptual (or theoretical) basis that can be universally agreed upon and shared among engineers and stakeholders. To overcome this challenge, it is essential to provide a well-established reference framework that guides scenario-based engineering activities in a systematic manner. This framework can include scenario vocabularies, such as a dictionary, that enhance understanding of scenario constructs and methods. Moreover, if it is presented in the form of a conceptual framework, it can serve as a more effective communication and analysis tool for establishing shared understanding and analyzing developed or to-be-developed scenarios. Even though various scenario development methods have been proposed particularly in the field of safety/mission-critical systems engineering [13, 29, 34, 43, 33, 24], they are highly specialized for specific system types and application domains. This still makes it challenging to utilize or extend them as general reference approaches or frameworks for future scenario engineering.

P2: Absence of a Scenario Development Method that Effectively Supports Ontological Analysis and Inter-Model References. Scenarios play a crucial role in the analysis, validation, and verification of software/systems as they provide detailed contextual information that significantly impacts system behaviors. Their importance becomes even more pronounced when the failure or malfunction of a system can lead to significant costs or losses. Therefore, it is essential to employ a well-designed scenario development method that not only specifies and describes scenarios but also aligns with system engineering activities such as system modeling, domain engineering, and environment engineering.

Scenarios are typically developed using scenario description/modeling languages, and it is imperative to use domain-specific languages when developing real-world domain-specific scenarios. For example, Figure 3 illustrates a simple scenario where an autonomous vehicle drives on a two-lane road. By describing the *World-Of-Interest* (WOI), including scenes, entities, behaviors/states, and contextual factors, scenarios can be developed and analyzed. A scenario goes beyond simply defining a path

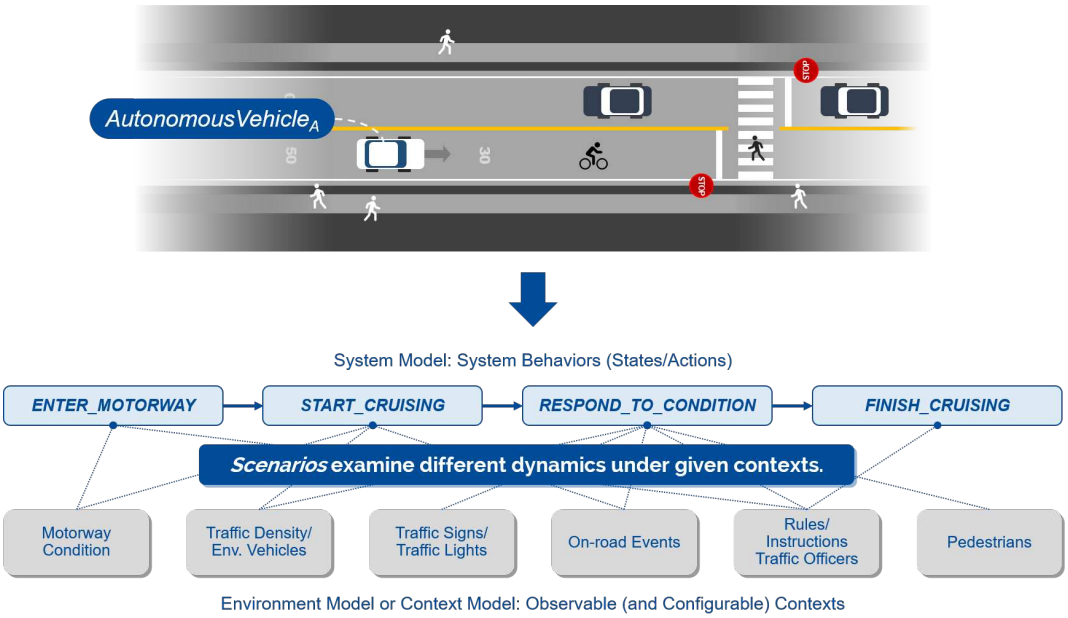

Figure 3: A simple case example that identifies ontological information from a case/scene description

or flow; it intricately weaves together concrete entities, objects, data, and variables present in the analyzed WOI, providing a comprehensive representation of the possibilities and interactions within the system.

To ensure practical effectiveness, scenarios must incorporate domain-specific objects and data. The most widely used approach for identifying domain information and establishing the boundaries of the WOI is through the use of *ontologies*, which provide an explicit specification of a conceptualization [20]. Ontologies not only conceptualize and organize domain information but also ensure consistency by unifying components within a discourse universe in a technology-independent manner. To enable a scenario description language to express domain-specific information and contextual factors as an ontology, it is crucial to provide language-level support for ontology modeling that can be flexibly extended to meet domain requirements. While some recent studies have highlighted the benefits of ontologies in scenario development [7, 4], and proposed the use of ontology models in specific domains like the *Operational Design Domain (ODD)*, there is still a lack of technical proposals and implementations on how scenario description languages can effectively utilize ontologies. One notable technical limitation is the lack of support for inter-reference, which establishes links between scenario models (or objects) and WOI ontology models. In the conventional process of describing scenarios, engineers have to manually refer to objects or data in external models of the WOI. However, if the modeling method or language provides support for inter-reference between scenario and WOI components, traceability can be enhanced, while ensuring better consistency.

P3: Lack of Extensibility of Existing Scenario Development Methods for Domain-Specific Applications. As discussed earlier, the scope and semantic space covered by scenarios can vary depending on the target application domain and its requirements. To effectively address the evolving domain requirements, a scenario development method should offer a suitable level of extensibility and flexibility at the method level. Currently, scenarios are often used as auxiliary artifacts across different domains, resulting in a lack of standardization and casual technology. On the other hand, when more formal approaches are used to develop scenarios, they tend to be developed in highly domain-specific formats and languages. While these domain-specific methods may excel within their specific domains, they may not be easily adaptable to other fields, highlighting the need for flexibility.

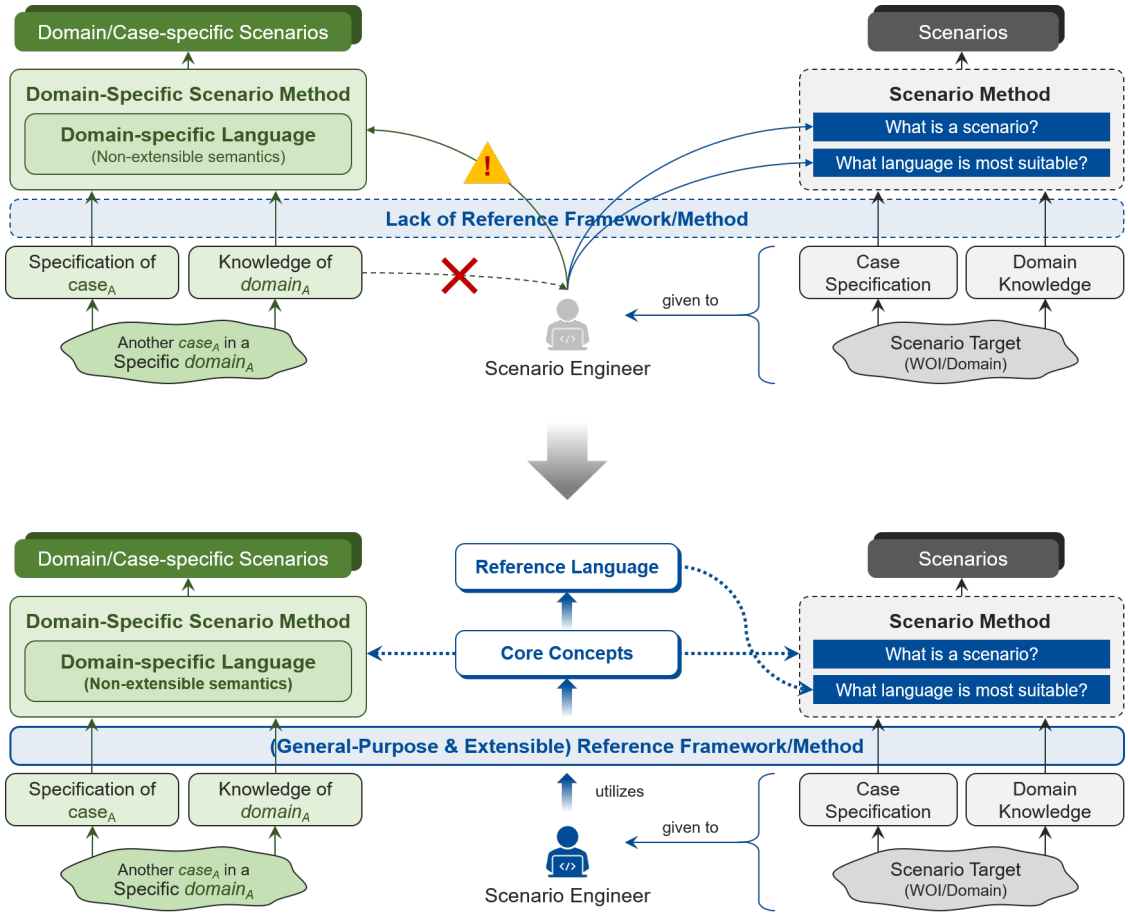

Figure 4: Illustration of the lack of a reference scenario framework/method

Figure 4 illustrates this challenge. Even if there is a domain-specific scenario method that effectively represents and analyzes $case_A$ and $domain_A$, it remains challenging to answer the question: "*How can we develop our own scenario method that supports the most appropriate scenario development language for our specific case and domain?*" Using a domain-specific method and language as a reference to assess its suitability for the engineer's target and to adapt it to a new language that aligns with their application domain can be burdensome and costly. The process of acquiring and understanding domain-specific information from other domains is time-consuming and tedious, requiring through checking of scenario description language semantics.

An effective solution to these challenges is to develop a scenario method and provide an intermediate-level reference method that supports the development of a domain-specific scenario modeling language. If the reference framework described earlier is well-established and can be provided to scenario engineers as a reference modeling method, core concepts and a reference language can serve as a starting point for extension to fit the engineer's own domain and case. By offering an extensible framework/method that focuses on scenario characteristics and provides a high-level common foundation, guidance can be provided on the components and rules for developing a domain-specific scenario method.

1.3 Research Objectives

Based on the key problems discussed, this study aims to achieve three primary objectives in developing a reference method for scenario-based software/systems engineering, with a focus on the necessity of a reference framework and method. The objectives can be summarized as follows: (A) Conducting a comprehensive literature review to investigate diverse scenario methods, (B) Suggesting a conceptual framework that conceptualizes scenario variables, data, and information, with the aim of establishing a shared understanding of the constructs of a scenario method, and (C) Proposing a scenario modeling method that can serve as a reference method and address the limitations of existing scenario description languages.

As an initial step in designing a conceptual framework, this objective involves conducting a literature review of publications from industry and academia that study scenarios or utilize scenario methods. By analyzing these sources, we can gain insights into the meanings and purposes of employing scenario methods and identify commonly used conceptual variables and data.

Building upon the conceptual variables identified from the literature review, we propose a *Conceptual Scenario Framework (CSF)* that encompasses conceptualized *Scenario Variables (SVs)* and a *Conceptual Scenario Model (CSM)*. The *CSF* aims to provide guidance to scenario engineers in establishing a shared understanding of scenarios and scenario methods. The components of *CSF* facilitate comprehensive analysis and development of a scenario method. This framework serves as a reference for scenario engineers to answer questions such as "*What is a scenario*?", "*What are the constructs of a scenario and a scenario method?*", and "*What aspects and components should be considered for the development of a scenario method?*"

Leveraging the *CSF* as an input reference framework, this study also proposes an *Extensible Scenario Modeling Method (ESMM)*, which includes an *Extensible Scenario Modeling Language (ESML)*. To tackle the challenges discussed in the previous section, our method supports ontology-based scenario development by incorporating an inter-reference mechanism between scenario models and WOI models/objects. Moreover, scenario engineers seeking to develop a domain-specific scenario language can utilize the *ESML* library as a reference scenario modeling language by extending the predefined scenario constructs.

1.4 Paper Organization

The structure of this paper is as follows. Section 2 provides an overview of related work, focusing on existing scenario methods and scenario description languages. Section 3 presents the overall approach and steps for developing the proposed framework and modeling method. Section 4 conducts a semi-systematic literature review to conceptualize the collected data into a conceptual model, leading to the development of the *CSF*. Section 5 introduces the *ESMM*, which includes a scenario modeling language and a procedure for language-level extension. The evaluation of the proposed framework and method is presented in Section 6, which includes case studies with real-world scenarios addressing the research questions. Section 6 conducts case studies with real-world scenarios according to research questions, and evaluates the proposed framework and method. Section 7 discusses potential threats to validity, and Section 8 concludes the paper.

2 Related Work

2.1 Scenario Development Methods

2.1.1 General-Purpose Scenario Development Methods

General-Purpose (GP) scenario development methods aim to establish standard practices for scenario specification and validation. These methods offer several advantages, such as utilizing well-known

modeling languages (e.g., graph-based, diagrammatic modeling languages) and providing intuitive and widely accepted semantics. One traditional and straightforward method for describing scenarios is to use a semi-formal graph or tree-based model to describe and test possible sequences of states or events/actions. This approach, known as *Scenario Tree (ST)*, defines functional behaviors or interactions of a system by analyzing alternative paths (branches) and considers executable sequences as scenarios [36]. An extension of ST called the *Scenario Search Tree (SST)* explicitly includes conceptual variables along with functionality, states, or events [9]. While these approaches effectively model system behaviors similar to a behavioral model, they have limitations in expressing contextual information and environmental conditions that can influence system behaviors.

Another method, called *Scenario Description Language Q*, was developed as an extension of *Scheme*[2]. It focuses on analyzing and constructing interactions between social agents and humans [22]. By supporting semantics for variables, cues, actions, commands (guarded or unguarded), agents, and environment, this method enables specification of state transitions and actions while considering the external environment. However, it relies on established functions of the software/system and may lack comprehensive information without the involvement of system engineers. Additionally, it has limitations in the specification and analysis of functions between software, social agents, and users from a human-computer interaction perspective.

The *ACDATE/ Scenario model* provides a more sophisticated representation of general scenario semantics. It specifies scenarios based on *actors*, *conditions*, *data*, *actions*, *timing*, and *events* (ACDATE), and optionally includes *policies* (ACDATEP) [37]. This approach supports scenario-oriented requirements engineering and planning, particularly in command and control systems, through the *Integrated ACDATE/Scenario Model (IASM)*. The *IASM* facilitates static analysis to check completeness, consistency, and analyze service properties, such as reliability. While it supports generic behavioral modeling of various system types, its low-level (code-like) and fixed set of semantics limit flexible and extensible scenario specification.

Aside from these methods, many existing approaches have utilized variants of the UML/SysML-style sequence diagram (e.g., *Action Sequence Charts (ASCs)* [21], *Modal Sequence Diagram (MSD)* [16, 17]), semi-formal diagrams (e.g., process mining for scenario discovery [39]), and formal modeling languages (e.g., *Petri Nets (PNs)* [11], *Hybrid Automata (HA)* [8], and *Extended Finite State Machine (EFSM)* [44]).

2.1.2 Domain-Specific Scenario Development Methods

In recent times, there has been an increasing adoption of domain-specific scenario development methods, particularly in domains involving critical systems like safety and mission-critical systems. Scenarios have emerged as a vital tool for engineers to capture specifications and enhance communication among stakeholders throughout the development lifecycle. These methods leverage scenarios to engineer various critical aspects, including features, dynamics and behaviors, processes, and regulations/policies, in accordance with relevant domain standards. By employing scenarios, engineers can ensure clearer and more comprehensive representation of system requirements and facilitate transparent communication among project participants.

Scenario-in-the-Loop (SCIL) is a scenario-based methodology for behavior-driven development (BDD) [42, 41]. It aims to automate testing by defining usage scenarios using a 3-layer composition: communication and documentation layer, modeling layer, and validation layer. *SCIL* leverages *Gherkin* syntax to systematically generate *Scenario Modeling Language for Kotlin (SMLK)* specifications, which are executable and testable models based on ample data from the target system under test. However, understanding and analyzing Kotlin-based implementation may be required for scenario definition, as the scenario semantic domain is not clearly defined.

[2] *Scheme* is a dialect of Lisp Programming Language.

Scenario Modeling Language (SML) suggested by Greenyer *et al.*is a variant of live sequence charts (LSCs) [16, 18, 19]. It focuses on the interaction aspect and defines situations in which objects within the system may, must, or must not react. Scenarios developed using *SML* have high executability due to their systematic mapping to entities defined in a system model. *SML* includes elements such as *domain* (Controllable, Uncontrollable), *operations* (Events, Messages), and *collaboration* (Role, Requirement, Scenario, Assumption). While *SML* can respond to various system models and environments, it is a domain-specific language and may restrict the semantics of information considered in scenario development to LSC-based syntax. However, it provides rationale for using abstracted models and allows realizability checking or property violation detection through formal models.

Gherkin[3], a line-oriented language developed by *Cucumber*[4], is designed for describing use cases to generate tests. It follows the syntax of BDD, incorporating keywords such as *And*, *Given*, *When*, *Then*, and *But*. Scenarios written in *Gherkin* outline the *preconditions*, *action steps*, and *expected outcomes* using the keywords into a textual form. As a syntax-based language, *Gherkin* provides a structure for scenarios but leaves the definition of data semantics, inputs, behaviors, and flows to the discretion of the developer/designer. Consequently, it can be considered a template-based approach for succinctly capturing scenarios rather than providing conceptual modeling.

Two notable examples of domain-specific scenario definition languages are the *Military Scenario Definition Language (MSDL)* and the *Aviation Scenario Definition Language (ASDL)*. *MSDL* is a language standardized by *Simulation Interoperability Standards Organization (SISO)* for developing scenarios in the military domain, specifically for modeling and simulation of command and control systems [34, 43]. It provides a specialized ontology tailored to the military domain, enabling detailed analysis, planning, and execution of operations. Notably, *MSDL* recognizes the heterogeneous and federated nature of the military domain, placing significant emphasis on scenario consistency and reusability as key quality attributes. Being a *SISO* standard, the *MSDL* format facilitates the online development of real-time, collaborative, and geographically-distributed scenarios through interfaces like *WebMSDE*[5].

Similarly, *ASDL* offers a methodology for capturing and specifying scenario details related to flight missions in the aviation domain, including procedures, operations, and communication [24, 23]. It supports the development of operational, conceptual, and executable scenarios using a model-driven engineering process, enabling multi-level and multi-purpose scenario development. In the aviation domain, scenarios require more comprehensive and in-depth analysis of the system and the ever-changing environment. To facilitate this, an ontology model that defines domain-specific objects (e.g., *aircraft*) and environmental elements (e.g., *weather*) is used as an input, and the analysis of scenario elements and communication is supported through the integration of the *base object model*.

However, the domain-specific nature of *MSDL* and *ASDL* can limit their extensibility to other domains. Their structured and standardized methodologies (e.g., the *MSDL* organization includes *Units* and *Equipment* as mandatory classes) lack the flexibility to adapt to newly discovered problems or application domains. Therefore, abstracting the domain-specific elements can enhance the flexibility of the scenario languages, enabling methodologists and engineers to utilize, modify, and extend the existing methods more easily. Moreover, this flexibility can support scenario-based engineering activities conducted by independent analysis and domain experts who may have limited information on the architecture and implementation details of the target systems involved in the simulation.

OpenSCENARIO[6], developed by the *Association for Standardization of Automation and Measuring Systems (ASAM)*, is a standard for traffic simulation scenario development, complemented by *Open-*

[3]Gherkin Syntax, https://cucumber.io/docs/gherkin/
[4]Cucumber, https://cucumber.io/
[5]Web Military Scenario Development Environment
[6]ASAM OpenSCENARIO: Version 2.0.0 Concepts,
https://www.asam.net/project-detail/asam-openscenario-v20-1/

DRIVE[7] and *OpenCRG*[8]. While *OpenDRIVE* and *OpenCRG* provide static content, *OpenSCENARIO* offers dynamic and vendor-independent traffic elements and maneuver libraries. This allows for the creation of layered environments in scenario development and simulation, facilitating the testing, validation, and certification of driver assistance systems and autonomous driving environment.

Similarly, X. Zhang and S. Khastgir propose the *Scenario Description Language (SDL)* for the domain of automated driving systems [45]. *SDL* stands out for its identification of various models and data sources (such as accident database and System-Theoretic Process Analysis (STPA) analysis) required for scenario development. It provides a set of specific models to describe the scenario development process. *SDL*-defined scenarios encompass scenery, dynamic elements, base scenarios/elements, and can include contextual and causal factors as needed. The focus of their research is on testing control actions specific to the driving system domain and identifying potential unsafe hazards. However, specifying scenarios using *SDL* may pose compatibility issues, as they are defined specifically at the domain-level and may lack the necessary elasticity for engineers with varying goals.

Another scenario description language is *Scenic*, proposed by Fremont *et al.* [14]. *Scenic* allows the definition of various environmental scenes, such as badly-parked car scenes, which serve as training data for machine learning to analyze and architect perception systems. The scenarios described in this work focus on generating synthetic data based on statistical behaviors of environmental elements, primarily for researchers working on data generation and sampling for the learning within the same framework and toolset. It also enables the description of elaborate scenes with detailed specifiers and expresses uncertainty through probability. However, this approach is scene-centric and relies on a probabilistic programming language to generate data, which limits its ability to specify complex processes.

In the field of ADS engineering, scenario methods have often emphasized contextual information such as non-ego vehicles, road network traffic, and weather conditions, similar to other critical systems like the military and aviation domains. ADS scenarios distinguish between normal baseline scenarios and critical scenarios based on the criticality (e.g., safety, mission, or security) of the behaviors under study. While domain-specific approaches support ontological analysis and the application of domain knowledge, the lack of commonly shared conceptual basis hampers extensibility and flexibility of these methods. The introduced studies are just a few of scenario studies in ADS domains. Other scene-focused scenarios (e.g., Ontology-based Scene Creation [4]), infrastructure/environment-focused scenarios [40], and risk/hazard-focused scenarios [5] also hold important positions in scenario-based testing and simulation (in the ADS domain).

2.2 Discussion on Related Work

Based on the preliminary investigation of related work, we can compare some major studies that support techniques, methods, or languages for development. Table 1 first categorizes existing scenario development methods into *general-purpose (GP)* and *domain-specific (DS)* methods depending on whether they are dependent on particular application domains, and compares some features provided by the methods to discover the similarities and differences between the approaches. A circle symbol (◯) indicates that the method generally supports the feature well, a triangle (△) indicates partial support, and an × symbol indicates that there is generally no explicit support to represent the semantics. In the case of triangle-marked cells, the specific method or tool used to support the feature is noted in parentheses.

In general, GP approaches include narrative or template-based scenarios (e.g., functional test description, use case scenario), model-based diagrams (e.g., sequence diagrams), business process models, and context diagrams. On the other hand, DS methods encompass code-based language, schema-based methods, and other simulation-oriented scenario description languages. During the investigation, it

[7] Open Dynamic Road Information for Vehicle Environment
[8] Open Curved Regular Grid

became evident that the most active application and standardization efforts for scenario methods were observed in critical system domains, such as automotive, aviation, and military. Given that such critical systems often involve the active engagement of multiple stakeholders and engineers in communication and decision-making processes, the significance of employing scenario methods has been widely recognized, with scenarios playing a crucial role in system development.

The first feature to be analyzed can be summarized as the question "*What aspects of a target are primarily described/analyzed through scenarios?*". DS methods have primarily focused on supporting scenario development techniques that effectively analyze the comprehensive relationships between contexts, dynamics, and ontologies. In contrast, the semantics of GP (modeling) methods/languages are more limited. Since most GP methods aim to provide a convenient and unified means of modeling, they have limitations in expressing the comprehensive and essential information required for scenario specification.

Table 1: Preliminary investigation of scenario development methods
UML Sequence Diagram (UML-SD), UML Activity Diagram (UML-AD), Live Sequence Charty (LSC), Business Process Modeling Notation (BPMN)
*Context Diagram (CD), Open Simulation Interface (OSI)
**Scenario Modeling Language for Kotlin (SMLK), Aviation Scenario Definition Language (ASDL), Military Scenario Definition Language (MSDL), Scenario Modeling Language (SML)

Feature \ ML	**Unified / General-Purpose**			**Domain-Specific*** **		
	Narrative/Template	**Model-based***	**Context-oriented****	**Code-based**	**Schema-based**	**Simulation-oriented**
	Use Case Scenario	*UML-SD, UML-AD, LSC,BPMN*	*Context/ Interface Models (CD, OSI)*	*GherkinScenario, SMLK*	*ASDL,MSDL*	*OpenSCENARIO, Scenic Program,SML*
Aspect	Facet of System's Behaviors		Contextual Entity	Event/Action Sequence	Task/Mission Description	Description of Dynamic Content
Domain	General-Purpose			PL-Specific	Aviation	Automotive
Specification Format	Template Instance	Diagram/Model	Schema	Script/Code	Schema	Schema, Script/Code
Formality	Informal	Semi-Formal | Formal	Semi-Formal	Semi-Formal	Semi-Formal	Semi-Formal | Formal
Abstraction Level	Functional	Logical | Concrete	Functional	Concrete	Logical	Logical | Concrete
Hypothesis Definition	Implicit	Input	X	X	Input	Implicit | Input
Ontological Analysis	X	X	O	Database	O	External
Inter-Reference	X	X	X	Tool-Dependent	Ontology-based	External
Semantic Extensibility	X	△ (Stereotype)	X	X	△ (Schema)	△ (Namespace)

When comparing the methods in terms of the formality and abstraction levels of developed scenarios, it is observed that semi-formal (i.e., model/diagram-based) description techniques are generally preferred in both industry and academia. Since scenarios are intended to serve as artifacts that facilitate improved communication among multiple engineers and stakeholders with diverse backgrounds, there is a tendency to avoid formal or program-like languages that are only interpretable by machines. Some methods have been identified that support the development or conversion of scenarios into executable models with concrete values, enabling immediate execution of scenarios from scenario files.

Another observation is that many methods lack support for ontological analysis at the method or language level. In most cases, engineers need to manually grasp, understand, and utilize specifications of their domains and worlds to build ontology models. While certain DS methods support the use of ontologies defined in external models, they do not explicitly provide means or mechanisms for achieving inter-referencing between scenarios and their target world models. This limitation can lead to a decrease in synchronization, traceability, and consistency of scenario data.

Furthermore, most scenario languages do not offer semantic extensibility and mechanisms for extension or specialization at the method level. The lack of extension support makes it challenging to use them as reference modeling methods and hampers their flexible expansion for specific application domains. Therefore, this study primarily focuses on language-level constructs' extensibility and proposes a scenario modeling method that can serve as a reference method/language. Our apporach can be considered a general-purpose method not tied to a specific engineering/application domain, offering inherent scalability. Scenarios can be modeled using semi-formally defined models and schemas. Our method aims to define the contexts and dynamics of a scenario target and, most importantly, provide means and mechanisms to build ontologies and support the inter-reference relationships.

3 Overall Approach

This study follows the two stages of research processes, as Figure 5 shows. In the first *Stage A*, a literature review is conducted to investigate scenario data/information from scenario methods that were already suggested/developed. The main goal of the *Stage A* is to develop a *CSF* that encompasses conceptual *SVs* and a *CSM* supporting/guiding the application of scenario methods. On the basis of the well-established conceptual framework, the second *Stage B* develops an *ESMM* that supports ontological analysis for scenario specification and domain-specific extension at the method/language-level. After all these processes, an *ESMM* library and a modeling tool (*ESMM-Tool*) will be released as a reference scenario development/modeling method.

3.1 Stage A: Literature Review & Development of Conceptual Scenario Framework

A-(i). Design Literature Review: This study first designed search queries to conduct a semi-systematic literature review on scenarios and scenario methods in both industry and academia. *A-(ii). Select Publications:* An initial set of publications was collected from search engines (*Scopus* and *Web of Science*) using the search queries, and they were then reviewed through multiple rounds of selection processes based on selection criteria. *A-(iii). Investigate Scenario Concepts & Data:* This step systematically collected information and data related to scenarios or scenario methods to establish a comprehensive conceptual background. *A-(iv). Classify and Conceptualize Scenario Variables:* The collected scenario data was conceptualized as *SVs*, which are key elements of the *CSF*. *A-(v). Develop a Conceptual Scenario Model:* Based on the defined *SVs*, a conceptual model, known as *CSM*, was developed, organizing the variables with specific relationships.

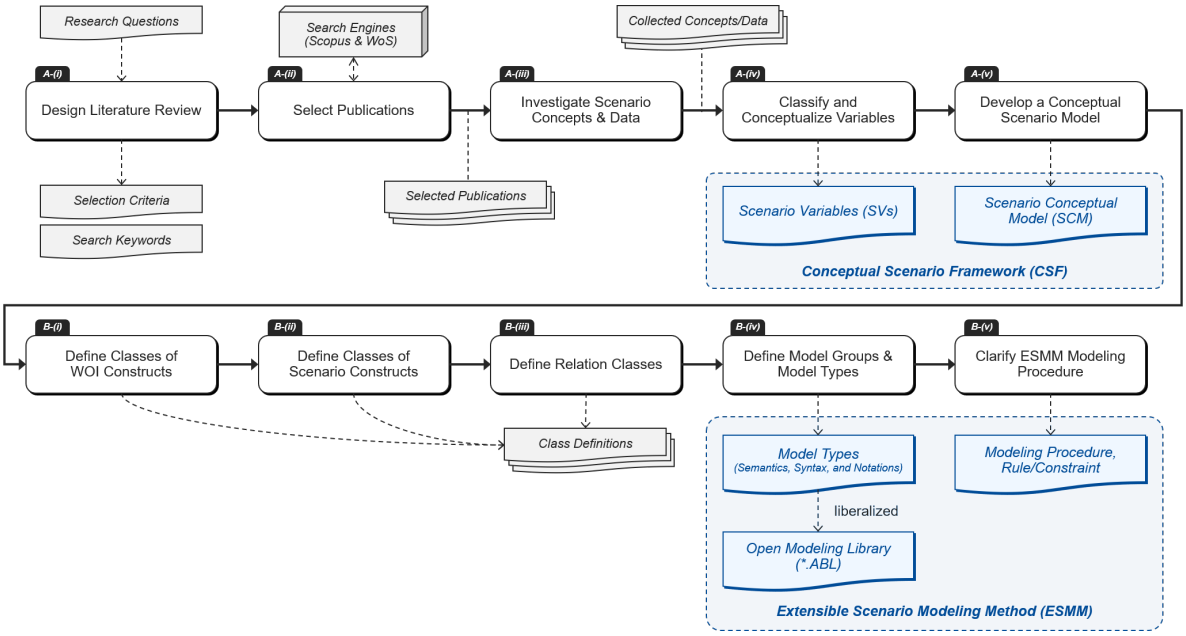

Figure 5: Overall Approach

3.2 Stage B: Development of a Scenario Modeling Method and Its Modeling Language

B-(i). Define Classes of World-Of-Interest (WOI) Constructs: WOI constructs were defined to support ontological analysis in the development of *ESML* models. Abstract and concrete classes were defined to express WOI and its components. *B-(ii). Define Classes of Scenario Constructs:* This step defines abstract and concrete classes for scenario models based on the *SVs* from the *CSF*. *B-(iii). Define Relation Classes:* Relation classes are defined to express diverse relationships between model objects and inter-references. This step focuses on defining relations and specifying constraints between classes. *B-(iv). Define Model Groups and Model Types:* By using the defined classes and relation classes, this step defines 10 model types that classify models into two groups: *WOI Model Group* and *Scenario Model Group*. These components contribute to the development of *ESML* models, which can be packaged into exportable files. *B-(v). Clarify ESMM Modeling Procedure:* The modeling procedure in *ESMM* is divided into three phases: Scenario planning with ontological analysis, scenario design, and scenario specification/modeling. *ESMM* supports both top-down and bottom-up approaches based on information ordering.

4 Conceptual Scenario Framework (CSF)

The main objective of *Stage A* is to establish a conceptual framework for scenario methods, which encompasses methods related to scenario definition and utilization for various purposes. Following the steps outlined in Figure 6, this study involves designing search queries and selection criteria, selecting relevant publications, identifying conceptual variables, and finally, developing a conceptual framework.

4.1 Literature Review of Scenario Methods

During the literature review process, the focus is on designing appropriate search queries and selection criteria to ensure a comprehensive collection of diverse scenario methods and concepts.

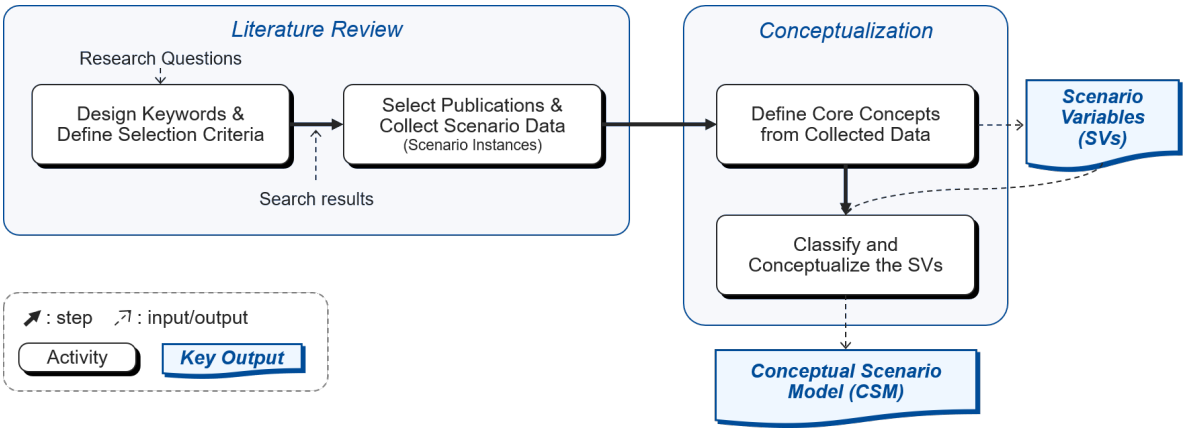

Figure 6: Overall process to develop the *CSF*

4.1.1 Design of Literature Review

Targets of this survey[9] are (a) scenarios defined or specified/modeled in selected publications, and (b) scenario-based/driven engineering approaches (i.e., methods, techniques, methodologies) suggested or utilized in the reviewed publications. The ultimate goal of this literature review is to develop a conceptual basis for scenario methods. Therefore, this study focuses on identifying as much conceptual data and defining them as conceptual *scenario variables*, shortly *SVs*.

Search Engines and Search Keywords. In the survey, initial raw publications were first collected using the most well-known search engines in the engineering and science fields, *Scopus* and *Web of Science* (WoS). The first step of our literature review process is the construction of search queries and keywords. To research a tuned set of scenarios and scenario methods used in software/systems engineering domains, we refined the search and determined three main keywords. First, like the initial search, "software/system(s) engineering" was included in the search term to limit the engineering domain of publications to software and systems engineering studies. Second, "scenario" was certainly included to research scenarios and scenario-based methods, techniques, and methodologies. To get more elaborate results, "scenario-based/driven" and "event" were also included in the actual search query to retrieve scenario methods that consider events. Finally, "requirement," "validation," "test," and "simulation" were included in the query, because they were deemed the four most representative engineering activities employing scenario methods, based on the preliminary investigation of Section 2.

Following are the search queries for *Scopus* and *WoS*, respectively[10].

> ALL("software engineering" OR "system engineering" OR "systems engineering") AND TITLE-ABS-KEY(("scenario*" OR "scenario-based" OR "scenario-driven") AND "event*") AND TITLE-ABS-KEY("requirement*" OR "validation*" OR "test*" OR "simulation*") AND PUBYEAR > 1999 AND (LIMIT-TO (SUBJAREA , "COMP")) AND (LIMIT-TO (LANGUAGE , "English"))

> "software engineering" OR "system engineering" OR "systems engineering" (All Fields) and "scenario*" OR "scenario-based" OR "scenario-driven" (All Fields) and "event*" (All Fields) and "requirement*" OR "validation*" OR "test*" OR "simulation*" (All Fields) and 2000-2021 (Year Published) and English (Language) and Engineering or Computer Science (Research Areas)

The reason why this is a *semi-systematic literature review* is that (a) we only use two major search engines (*Scopos*, *WoS)* and (b) the search keywords are not incrementally designed or refined during the

[9] Y. M. Baek, E. Cho, D. Shin, and D. H. Bae, "Literature Review to Collect Conceptual Variables of Scenario Methods for Establishing a Conceptual Scenario Framework", *arXiv:2205.08290 [cs.SE]*, 2022.

[10] Note that the research area was limited to *computer science* and *engineering* (i.e., COMP) for both search engines.

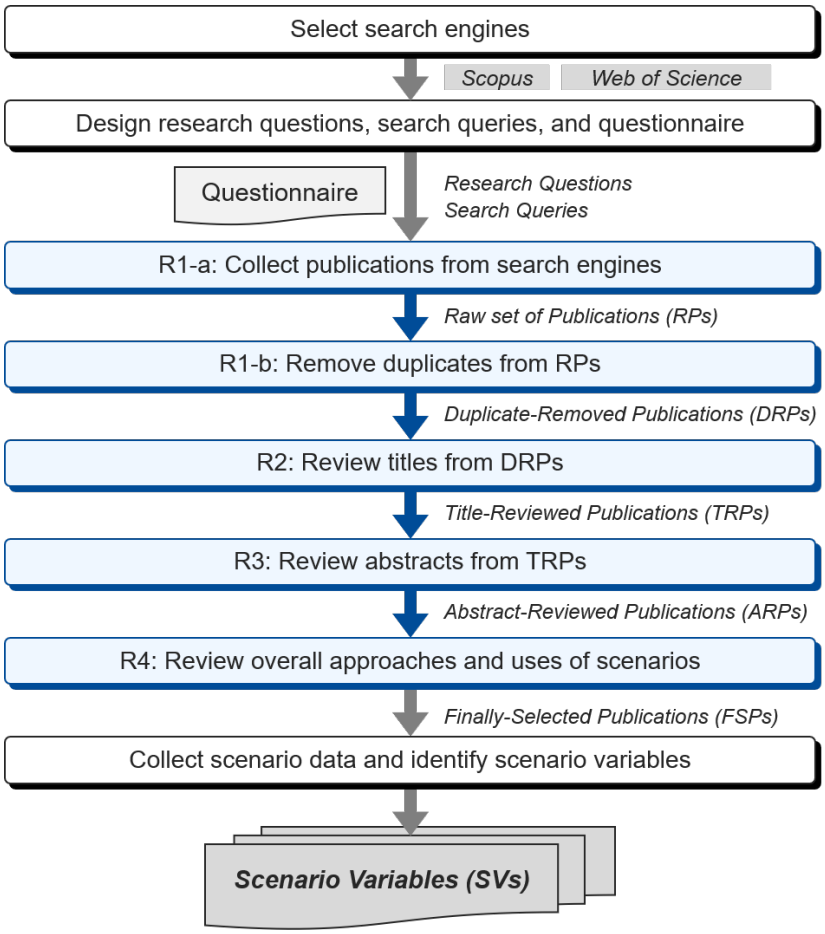

Figure 7: A publication selection process for the literature review

survey process (i.e., Our survey does not use a snowball method to find more literature from selected publications). Therefore, the selection process, introduced in Section 4.1.2, only excludes publications that are away from our interests and intentions, which are to identify constructs and variables used in scenario methods.

Selection Criteria. To ensure efficient examination of publications, specific inclusion and exclusion criteria were designed. The inclusion criteria were designed to encompass publications that contribute data, concepts, and values related to scenarios or scenario methods.

- Studies of scenario development, such as modeling and specification methods (and languages), techniques, processes, and methodologies
- Scenario-based or -driven engineering studies, which explicitly specify scenarios for specific engineering purposes, such as scenario-based requirements engineering, validation, design, testing, simulation, and verification

Conversely, the exclusion criteria were applied to filter out publications during the search process. Two levels of exclusion processes were implemented: *title-abstract-exclusion-criteria* for rounds 2 and 3, and *ARP-FSP-exclusion-criteria* for rounds 4 and subsequent rounds. These criteria ensure that the survey focuses on relevant publications and narrows down the scope of the investigation.

Table 2: Selection of publications from Round 1 to Round 4

Round	Criteria	Output	Selected	Excluded
R1-a: Collect Publications	-	*Raw Publications (RPs)*	1071	
R1-b: Remove Publications	*title-abstract-selection-criteria*	*Duplicate-removed Publications (DRPs)*	992	79
R2: Review Titles		*Title-reviewed Publications (TRPs)*	851	141
R3: Review Abstracts	*ARP-FSP-selection-criteria*	*Abstract-reviewed Publications (ARPs)*	765	86
R4-a: Review Overall Approaches & Uses of Scenarios		*Finally Selected Publications (FSPs)*	354	411
R4-b: Select 100 Most-Relevant Publications	*relevance-criteria*	*FSP-100*	100	254

- *title-abstract-exclusion-criteria* (Exclusion criteria for title and abstract review): Inapt publication type (e.g., whole proceedings, an entire book, newspaper articles, web pages, etc.); Unrelated engineering domain, which is not related to software or systems engineering (e.g., chemical, biological, medical engineering or non-engineering publications); Unrelated system or application domain (e.g., political or organizational system, international ecosystem); and Unrelated approach, which does not utilize scenarios for an engineering purpose, and less than or equal to 3 pages
- *ARP-FSP-exclusion-criteria* (Exclusion criteria for reviewing abstract-reviewed publications (ARPs) and finally-selected publications (FSPs)): Scenarios only used as a term to simply represent a system, a system type, a paradigm, or a case; Scenarios (or scenario methods) not explicitly used or mentioned (i.e., unable to retrieve in a document); Absence of scenario instances or insufficient semantic data of scenarios or event; Totally informal scenarios (i.e., narrative descriptions)

4.1.2 Selection of Publications & Data Collection

In *Round 1 (R1)*, we first collected raw data from the search engines using the search terms. The initially collected publications are called *raw publications (RPs)* and the set of publications after the duplicate removal is called *duplicate-removed publications (DRPs)*. In *Round 2 (R2)*, we reviewed and excluded publications based on the selection criteria *title-abstract-exclusion-criteria* to obtain *title-reviewed publications (TRPs)*. Since some papers out of scope or not fit the purpose of this investigation, they are simply considered as unrelated studies by reviewing their titles. In *Round 3 (R3)*, we manually reviewed abstracts of *TRPs*, along with their introduction and conclusion sections to extract more detailed insights and information beyond what could be obtained solely from the titles. Like *R2*, this round also uses the *title-abstract-exclusion-criteria* to exclude less relevant publications and generate a set of *abstract-reviewed publications (ARPs)*. In the final round (*Round 4 (R4)*), we reviewed key sections of the *ARPs* based on the *ARP-FSP-exclusion-criteria*. From this review, we obtained a set of *finally-selected publications (FSPs)* that were then prioritized for a full-read review, and then selected 100 most-relevant publications were selected (*FSP-100*). Table 2 provides an overview of the selected and excluded publications, including statistics for each selection round. The overall process for data collection is illustrated in Figure 7.

4.2 Conceptualization of Scenario Variables (SVs)

In this study, *SVs* are defined as key constructs of a conceptual framework and they are further classified into *primary variables* and *subordinate variables*, as Table 3 summarizes. Also, the collected *SVs* from the *FSPs* have been classified into four levels of a scenario method. Each level is intended to be addressed by different engineers/stakeholders who possess expertise in specific areas, enabling them to focus on more targeted issues, knowledge, and information. For instance, a system engineer would have extensive knowledge about system states and features required to specify the event-level scenario constructs that define actions or functions, which require understanding of internal structures and behaviors. The following descriptions provide summaries of the levels and the *SVs*.

- *Method-level SVs.* The highest-level *SVs* pertain to the selection or development of a scenario method for specific engineering purposes. The method-level *SVs* involve the collaboration of different stakeholders to establish the overall goals and scope of the scenario method. A suitable method is determined to analyze a universe of discourse (UoD), encompassing relevant goals/requirements, plans, decisions, technologies, and potential risks at the project level.
- *Scenario Suite-level SVs.* A scenario suite represents a coherent set of multiple scenarios that share a common viewpoint and aligned goals. To ensure coherence and alignment, scenario engineers must systematically identify and analyze the shared information among these scenarios. This process involves narrowing the gap between scenario engineers from diverse backgrounds by defining clear targets and objectives.

Table 3: *SVs* identified and collected from the literature review

Level	**Primary SV**	**Subordinate SVs**
Method	*ScenarioPurpose*	*TargetWOI, TargetProblemDomain, TargetVisionIntent (UseOfScenario), Hypothesis, StrategyTactic*
	ScenarioSpecification	*SpecType, SpecSemantics, SpecSyntax, SpecLanguage, SpecFormality*
	ScenarioExecution	*ExecType, ExecDriver, ExecDondition, ExecAutomation, ExecOutput, ExecMedia, ExecCoverage*
ScenarioSuite	*SuiteScaleMeta*	*SuiteScale, SuiteCohesion*
	SuiteViewpoint	*SuitePerspective, SuiteBaselineScenario*
	SuiteHypothesis	*HypothesisAssumption, HypothesisAssertion*
	SuiteTarget	*SystemDomainOntology, IntangibleFactorOntology*
	SuiteConstituents	*SuiteScenarioPool, SuiteEventPool, SuiteDataPool*
	SuiteInput	*SuiteInputConfiguration, SuiteInputData, SuiteInputModel*
	SuiteScenarioComposition	*InterScenarioAssociation, InterScenarioConcurrency, InterScenarioCausality*
Scenario	*ScenarioMeta*	*ScenarioType, ScenarioLifecycle*
	ScenarioParticipant	*ScenarioParticipant*
	ScenarioUnit	*ScenarioUnit, ScenarioInterUnitRelationship, ScenarioInterUnitTransition, ScenarioPattern*
	ScenarioCriticalityAnomaly	*ScenarioTargetCriticality, ScenarioTargetAnomaly*
	ScenarioInput	*ScenarioInputConfigurationMode, ScenarioInputModel, ScenarioInputDataParameter*
	ScenarioOutput	*ScenarioOutputData, ScenarioOutputModel, ScenarioIndicatorOracle*
	ScenarioCondition	*ScenarioCondition (Pre/Post), ScenarioConstraintInvariant*
	ScenarioTemporalData	*ScenarioTemporalScale, ScenarioTimeUnit, ScenarioTemporalData*
	ScenarioSpatialData	*ScenarioSpatialScale, ScenarioSpatialUnit, ScenarioSpatialData*
	ScenarioChange	*ScenarioChangeDriverAttribute*
	ScenarioInteraction	*ScenarioUserInteraction, ScenarioEnvInteraction, ScenarioInterSysInteraction*
	ScenarioUncertainty	*EventTransitionUncertainty, ScenarioEnvUncertainty*
ScenarioUnit	*UnitMeta*	*UnitTypeGranularity, UnitSource, UnitFrequency*
	UnitInput	*UnitInputParameter, UnitInputModel*
	UnitOutput	*UnitOutputData*
	UnitCondition	*UnitConstraintInvariant, UnitCondition (Pre/Post)*
	UnitBehaviorOccurrence	*UnitActor, UnitAction, UnitOccurrenceMeasure*
	UnitInteraction	*UnitUserInteraction, UnitInteractionMessage*
	UnitTemporalData	*UnitTime, UnitSynchronization*
	UnitSpatialData	*UnitLocation*
	UnitUncertainty	*UncertaintySource, UncertaintyRepresentation*

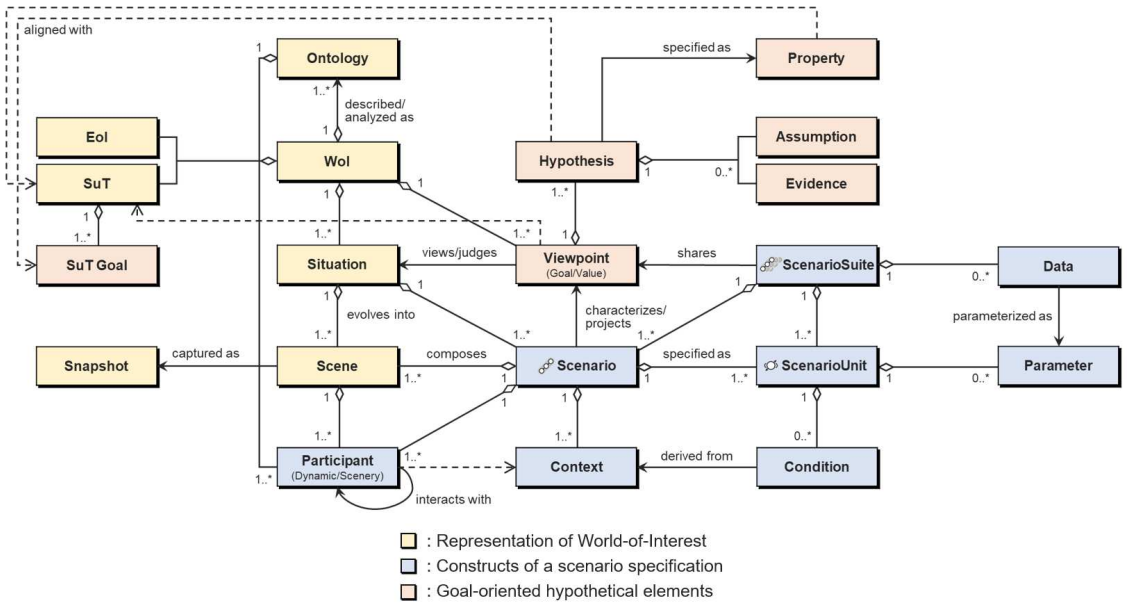

Figure 8: A *CSM* and three aspects of scenario methods

- *Scenario-level SVs.* Building upon the higher-level SVs, scenario-level *SVs* focus on the specification of individual scenarios. Individual scenarios may have different abstraction levels, semantics (configurations, properties), contexts and data. To sufficiently express the information, scenario-level *SVs* include participants, inputs/outputs, conditions, configuration, and scenario-level uncertainty.
- *Scenario Unit/Event-level SVs.* The lowest and most detailed level of information is captured by event-level *SVs*, which represent the behavioral aspects and dynamics of a scenario. Events, as defined in conventional modeling approaches, refer to occurrences that elicit a response from the system or any interactions. They are characterized by action(s) that can be triggered/performed either internally or externally, when specific preconditions are met. Various studies distinguish between events, acts, actions, activities, and stimuli based on their causes and types. However, we adopt a broad definition of an event, considering all behavioral information and data can be abstracted as events and their occurrences.

4.3 Conceptual Scenario Model (CSM)

Based on the collected *SVs*, we construct *CSM*, as depicted in Figure 8. The *CSM* serves as a meta-model that conceptualizes essential information about a scenario method and its specification. It comprises three types of meta-classes represented by different colored boxes in the figure. Furthermore, the *CSM* comes up with relationships between the classes, leveraging the four-level hierarchy established for the *SVs*, which serve as constructs for scenario development. The role of the *CSM* is crucial in bridging and organizing the meta-classes extracted from three different sources (WOI (System), hypotheses, and scenario elements).

In applying a scenario method, it is important to begin with analysis of the WOI, which represents a scenario target that scenario methods aim to describe and analyze. In essence, the WOI encompasses all the entities (e.g., system, environment, situations, and scenes) that the scenario engineer intends to examine. Therefore, the WOI is often highly domain-specific, reflecting engineers' particular focus

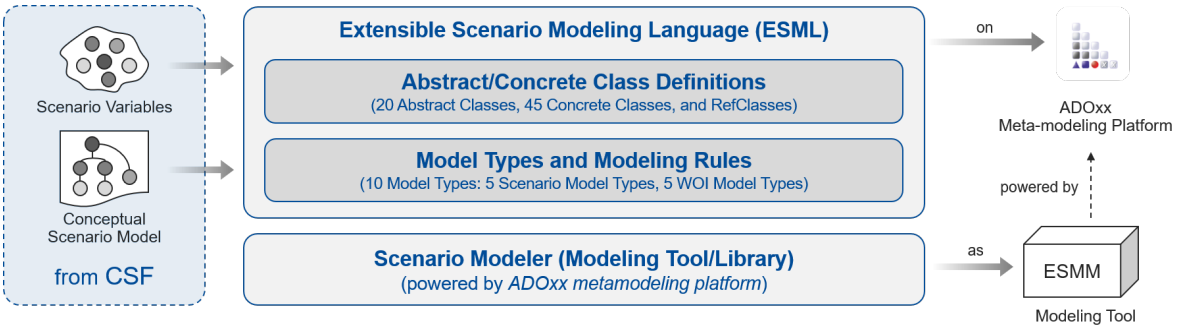

Figure 9: Development of *ESML* and *ESMM-Tool (Scenario Modeler)* of *ESMM*

on the scenario analysis. *Ontology*, which refers to an explicit specification of a conceptualization [20], is the most widely used method for defining the boundaries of the WOI. An ontology model serves multiple purposes: it conceptualizes and organizes entities, ensures consistency of information across different engineering activities and phases, and enhances interoperability. For instance, when analyzing the WOI for highway autonomous driving, one can construct ontologies such as the *highway ontology* (e.g., road network, traffic, regulatory elements), *vehicle ontology* (e.g., driving functions, maneuvers), and *weather ontology* (e.g., temperature, humidity, precipitation).

Next, the scenario method extracts the main contents and objectives for scenario specification, aligning them with the goals and objectives of engineering WOI. Traditional software engineering processes often rely on use cases and scenarios based on system goals or requirements. However, different viewpoints and hypotheses may result in varying definitions of scenarios even within the same WOI. By establishing a viewpoint, specific situations, behaviors, and scenes within the WOI and its system can be more specifically identified and analyzed. Once the viewpoint is established, the dynamics of scenarios need to be specified, analyzed, and observed. These dynamics are typically described as paths or flows, consisting of a sequence of scenario units that align with use cases, tasks, or missions. Scenario-level context is incorporated to account for situations that interact with or are affected by scenario execution.

The lowest level of scenario specification is the event or scenario unit, which specifies behavioral information or occurrences. Event-level *SVs*, such as input/output, behavior, interaction, temporal/geospatial information, and uncertainty, are utilized as data for event execution and configured as parameters. The event-level context also includes specific triggering conditions for execution and event transitions. For example, an ADS scenario may specify scenario units, such as driving functions, maneuvers (e.g., full-brake), external events (e.g., pedestrian crossing), and environmental event (e.g., weather change). Each event can contain temporal and spatial information and requires logical or concrete parameters (e.g., vehicle performance, acceleration rate, initial traffic scene).

5 Extensible Scenario Modeling Method

5.1 Overview of Extensible Scenario Modeling Method

Based on the component definitions of a modeling method proposed in [26], the *ESMM* comprises two fundamental elements: (a) *ESML* and (b) the *ESMM-Tool*, which encompasses the *ESML* and a *modeling procedure*.

5.1.1 Extensible Scenario Modeling Language

ESML is a modeling (or specification/description) language actually used by scenario engineers to develop scenario-oriented models. It provides *semantics*, *syntax*, and *notations* necessary for scenario modeling and WOI analysis to help engineers specify scenarios based on *SV*s. Major building blocks of the *ESML* are *abstract classes*, *concrete classes*, *relation classes*, and *model types*. An *ESML abstract class* is a non-instantiable class that contains shared attributes in common at a higher level, in scenario constructs and WOI constructs, respectively. On the other hand, *ESML concrete classes* are instantiable subclasses that inherits the attributes of their parent abstract/concrete class, and is used to define specific scenario/WOI elements. *Relation classes* of *ESML* are defined as a directed edge from a source class (from) to a destination class (to). Each *model type* of *ESML* is a sub-collection of *classes* and *relation classes* to represent/demonstrate a particular aspect of scenario engineering. Therefore, a model type are instantiated as an independent diagram, which includes a set of classes and additional constraints or rules (e.g., class cardinality).

5.1.2 *ESMM-Tool* powered by ADOxx Metamodeling Framework

ESMM and its components are developed based on *ADOxx Platform (Version 1.5)*, which is the metamodeling development and configuration platform for developing modeling methods[11]. The *ADOxx* supports the effective development of a modeling method/tool to provide users with a graphical modeling language, and core elements (e.g., pre-defined abstract classes with their attributes) for metamodeling a language to be developed in advance. As shown in Figure 9, a modeling method developed with *ADOxx* can be released as an *application library*, and it includes definitions of a set of model types and set of classes (and their attributes).

5.2 Key Features of *ESMM*

Definition of Reference Scenario Classes. The *ESMM* serves as a reference method to support scenario planning, design, and development, providing scenario engineers with tailored constructs for scenario specification. To achieve this, the abstract/concrete classes, relation classes, and their attributes are derived from the *SVs* and *CSM* developed in *CSF*. Based on our investigation in Section 4.1, the *SVs* became fundamental concepts that represent the essential elements of scenario constructs. As a result, the classes defined within the framework can be utilized as foundational materials for scenario development in various application domains, allowing for domain-specific extensions.

Support of Ontological Analysis. In Section 1, we highlighted the problem of acquiring data for scenario specification, specifically for the scenario target WOI, as existing scenario description techniques lack support for ontology building. This limitation prevents engineers from effectively utilizing information about scenario components, such as entities, data/variables, and events, as they are not integrated with the available and accessible models of the WOI elements. Incorporating ontological analysis becomes crucial in order to define an accessible data set, particularly when dealing with highly domain-specific objects and data. Additionally, for scenarios that need to be executable, parameterizing WOI components based on the developed ontologies becomes essential. The parameterization of objects and behaviors can be achieved through the ontological analysis process supported by *ESMM*.

Inter-Reference between Scenario Constructs and WOI Objects. Even if a scenario development technique does not directly support ontology definition, ontological analysis on a WOI can still be conducted using an external model separate from the scenario models. However, this approach has a limitation: the elements of the scenario model cannot directly access the domain object/data to express the information required to adequately specify scenarios. As mentioned earlier, to effectively reference a scenario target, a mechanism that establishes links (inter-references) between the WOI models and scenario models is necessary. In the case of *ESMM*, scenario elements can have inter-reference-typed

[11] Introduction to *ADOxx*: https://www.adoxx.org/live/introduction-to-adoxx

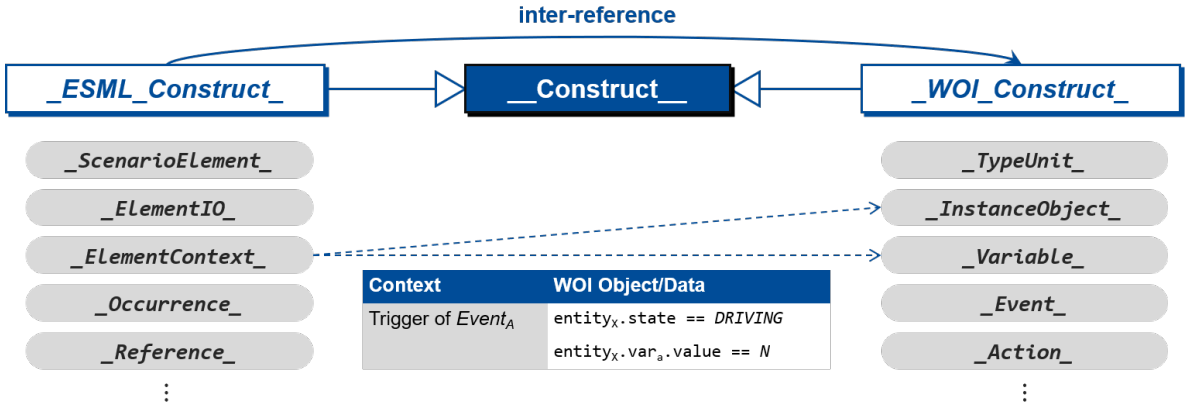

Figure 10: Classification of abstract classes into `_ESML_Construct_` and `_WOI_Construct_`

attributes that enable referencing model instances or objects in the WOI, thereby facilitating direct access to the WOI model or its objects. Furthermore, *ESMM* supports the separation of concerns for engineers involved in scenario development activities by enabling independent and parallel engineering of the WOI domain/system and scenario development tasks.

Flexibility & Extensibility. As discussed earlier, the classes in *ESMM* are derived from the *SVs* of *CSF*, making them domain-general classes specifically designed for scenario development. However, when applied to a specific application domain, there may be limitations. To address this, the open-source form of *ESML* allows for additional customized extensions to create a domain-specific scenario modeling language (DS-SML) that aligns with the preferences of domain engineers. Methodologists can specialize *ESML* into a DS-SML by importing the *ESML library* and defining additional domain-specific classes based on the predefined *ESML* classes. This flexibility enables the adapation of *ESMM* to various application domains.

Tool Support. The aforementioned features of *ESMM* are accompanied by a dedicated modeling tool called *ESMM-Tool*, which allows scenario engineers to effectively engage in practical scenario development. With this tool, scenario engineers can create 10 different types of diagrams tailored to the specific scenario target. Moreover, methodologists who are involved in customizing or developing new scenario languages to meet the requirements of their respective engineering domains can utilize the *ESML library* to assess and modify the internally-defined classes and model types.

5.3 ESML Constructs

5.3.1 Class Definitions

The constructs defined in *ESML* can be classified into two main element types, one is `_ESML_Construct_` and the other is `_WOI_Construct_`. *ESML* supports the definition of sub levels of abstract/concrete classes by inheriting the top-level abstract classes. Classes with two underscores before and after the class name (e.g., `__ClassName__`) are *ADOxx-predefined abstract-classes*, and *ESMM-predefined-abstract-classes* follow the naming convention with one underscore on the front and back (e.g., `_ClassName_`). Figure 10 shows the classification of highest-level of abstract classes and inter-reference relationships, and Figure 11 introduces key abstract classes defined in *ESML*.

Definition of `_ESML_Construct_` Classes. The `_ESML_Construct_` is the highest level abstract class to define the elements to be derived in the process of scenario analysis and specification centering on the scenario semantics defined in our conceptual framework. Figure 12 defines abstract and representative concrete classes inheriting `_ESML_Construct_`.

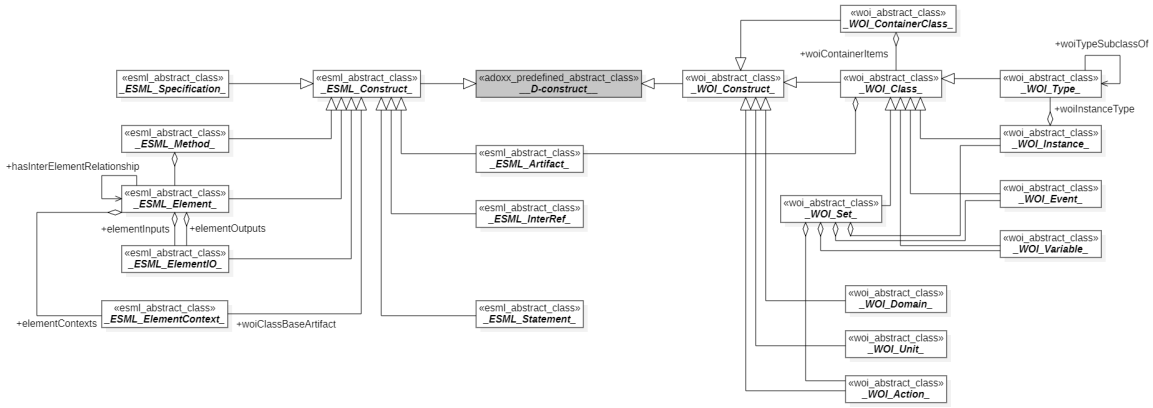

Figure 11: Highest-level of abstract classes defined in *ESML*

- `_ESML_Method_`: Almost all classes under `_ESML_Construct_` are used in a scenario method (SM).
- `_ESML_Element_`: This class is an abstract class to commonly define attributes of *scenario elements* that play the most pivotal role in *ESML*, such as *ScenarioSuite*, *Scenario*, and *ScenarioUnit*. All concrete classes derived from the different levels of `_ESML_Element_` are commonly characterized by their own participants, contexts, and inputs/outputs included at the different levels of the elements.
- `_ESML_ElementIO_`: This abstract class represents the information that is either required for scenario execution (i.e., input) or can be obtained when the element is executed or observed during the execution time (i.e., output).
- `_ESML_ElementContext_`: This abstract class derives all possible contexts, conditions, and constraints, since the ultimate purpose of applying an SM is to identify and analyze the contextual information.
- `_ESML_Artifact_`: `_ESML_Artifact_` is utilized to define subclasses for referencing external models, data, and documents with independent URIs to address the needs of scenario engineers.
- `_ESML_Specification_`: This abstract class is used to define specification classes/objects, which are closely related to hypotheses and viewpoints.
- `_ESML_Statement_`: The lowest level statements that need to be modularized and reused are defined/declared through this `_ESML_Statement_` class[12].
- `_ESML_InterRef_`: This abstract class is a parent class of the reference type classes used to refer to other model(s) or instance elements under `_WOI_Construct_`. It is mainly used to refer to WOI objects and instances, but it can also be used to refer to the *ESML* construct defined in one model type from another model type.

Definition of `_WOI_Construct_` Classes. This is an abstract class for defining WOI elements (i.e., ontological analysis) accessed from (i.e., (inter-)referenced by) scenario elements (i.e., concrete classes

[12]There are several subclasses to describe the occurrence of observable events (`StmtOccurrence`), to describe unit actions (`StmtBehavior`), to describe time (`StmtTime`) and location (`StmtLocation`), to describe probability (or probability distribution) (`StmtProbability`), to declare assertion statements (`StmtAssertion`), to describe assumptions (`StmtAssumption`), to express constraint information (`StmtConstraint`), to specify propositions requiring a true/false judgement (`StmtProposition`), and to specify arguments having a specific domain (`StmtParameter`).

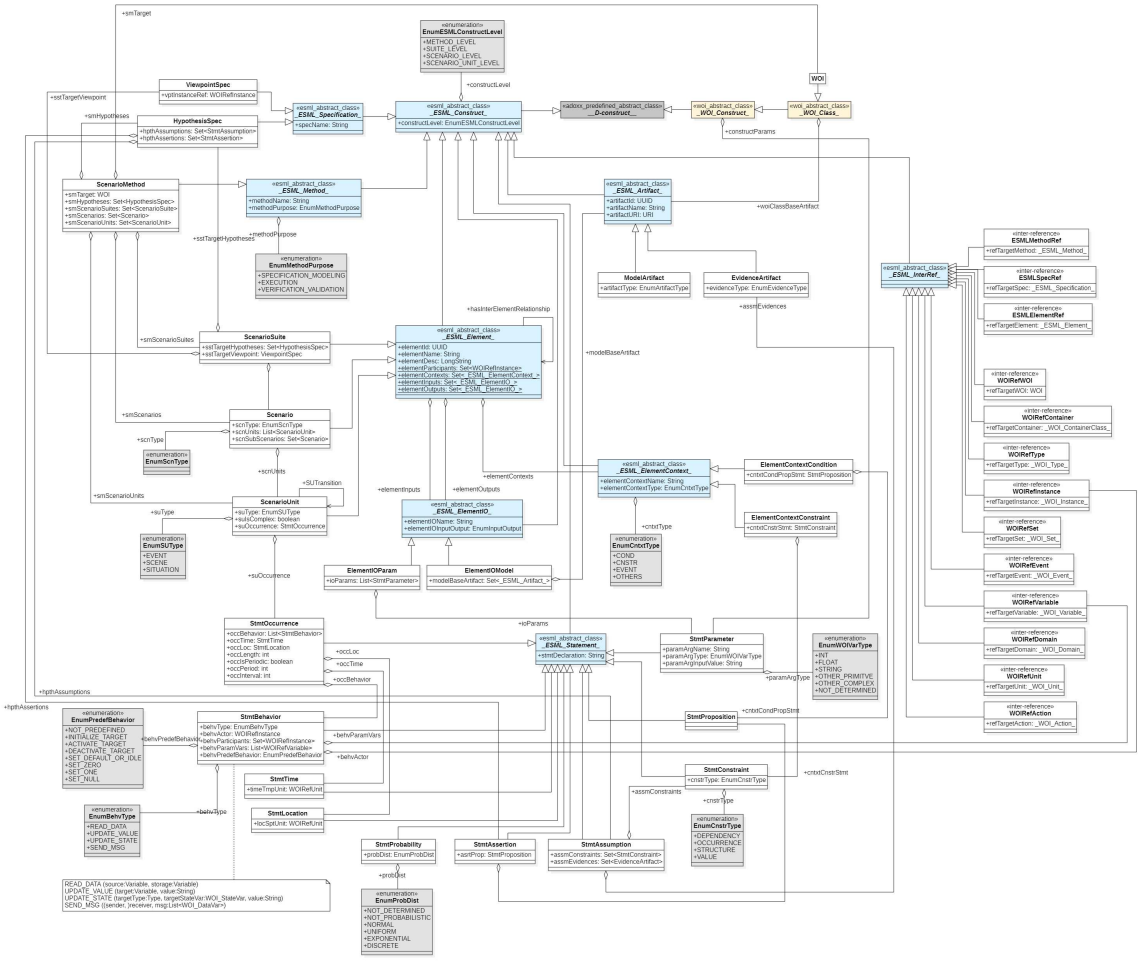

Figure 12: Scenario constructs (classes under `_ESML_Construct_`) defined in *ESML*

from `_ESML_Construct_`). The `_WOI_Construct_` class is defined based on the *Meta-Model for Systems-of-Systems (M2SoS)*—a meta-model that conceptualizes structural components of an SoS—proposed in the author's previous study [3]. Figure 13 defines abstract and representative concrete classes inheriting `_WOI_Construct_`.

- `_WOI_Class_`: To define class types, instances, events, variables, sets, which are major components of WOI constructs, it is used to define elemental and structural aspects of a WOI model. The main reason for using the term *class* is that the WOI objects are modeled through the instantiation of class types, according to the object-oriented paradigm (OOP).
  - `_WOI_Type_`: This class abstracts participant types, and two representative types of `_WOI_Type_` in *ESML* are `EntityType`and `DataType`.
  - `_WOI_Instance_`: A concrete class inheriting `_WOI_Instance_` (e.g., `TypedEntityInstance`, `EnvFactorObjectInstance`) can be defined by instantiating `_WOI_Type_` -based classes predefined in the WOI, and supports fields for initializing instances of WOI object accessed by scenarios.
  - `_WOI_Event_`: Classes inheriting the `_WOI_Event_` are used to specify occurrence elements that

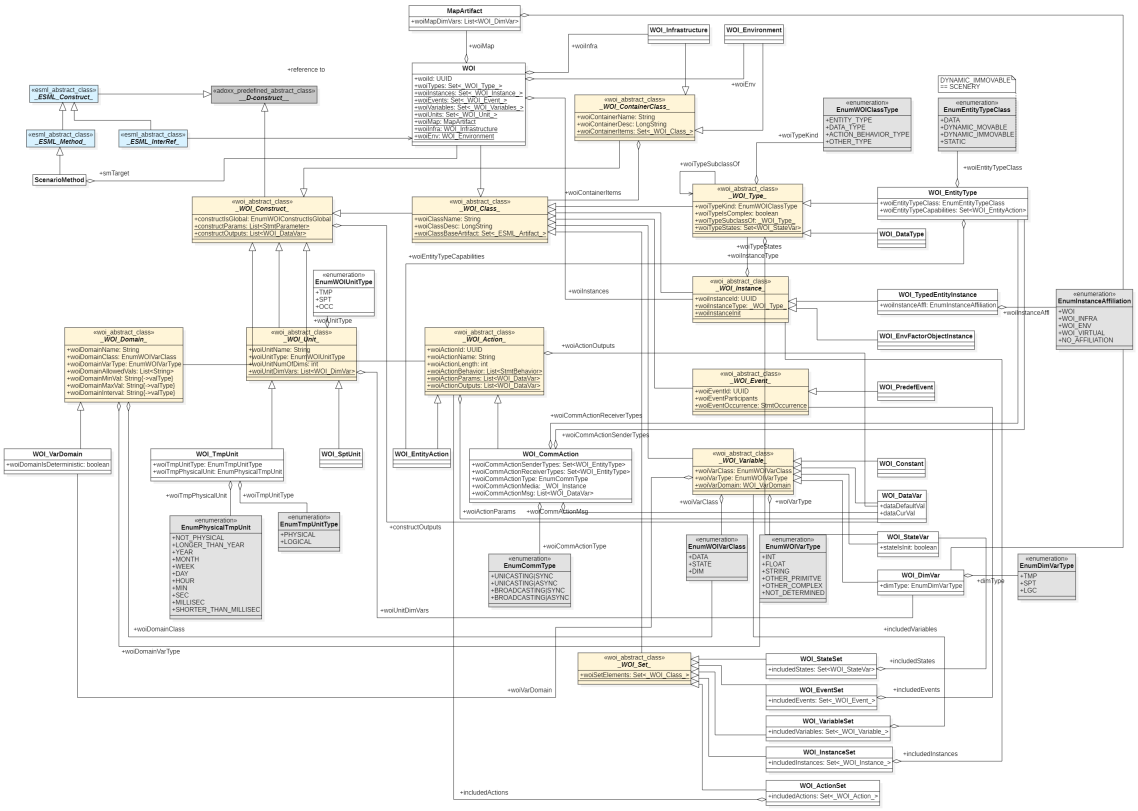

Figure 13: WOI constructs (classes under `_WOI_Construct_`) defined in *ESML*

are expected to occur during the execution of scenarios.

  - `_WOI_Variable_`: Specific data objects should be accessible by a scenario so that scenario elements perform CRUD (create, read, update, delete) operations on WOI elements. Variables are classified into 3 specialized types: `WOI_DataVar` stores typed data according to its data type, `WOI_StateVar` stores state information, and `WOI_DimVar` expresses a dimension of any unit.

  - `_WOI_Set_`: The last abstract class among the subclasses of `_WOI_Class_` is a `_WOI_Set_` that expresses sets of multiple class objects by tying them together.

- `_WOI_ContainerClass_`: `_WOI_Container_` classes need to be defined so that multiple participants of a scenario (or a scenario suite) can be selectively included as a bundle.

- `_WOI_Domain_`: While engineering scenarios, each variable serves to provide a place to be assigned a value of an appropriate type. All WOI variables in *ESML* must be modeled to include their `_WOI_Domain_`, which limits the values that the variables can contain, by defining *varType*, *allowedVals*, *minVal*, *maxVal*, etc.

- `_WOI_Unit_`: `_WOI_Unit_` refers to an abstract class that defines domain-specific multi-dimensional units to describe items that need to be defined through a measurable unit based on a particular metric in a WOI.

- `_WOI_Action_`: To describe capability element of a WOI entity expressed in scenarios, *ESML* statically describes the executable function through this `_WOI_Action_` class. *ESML* simply ab-

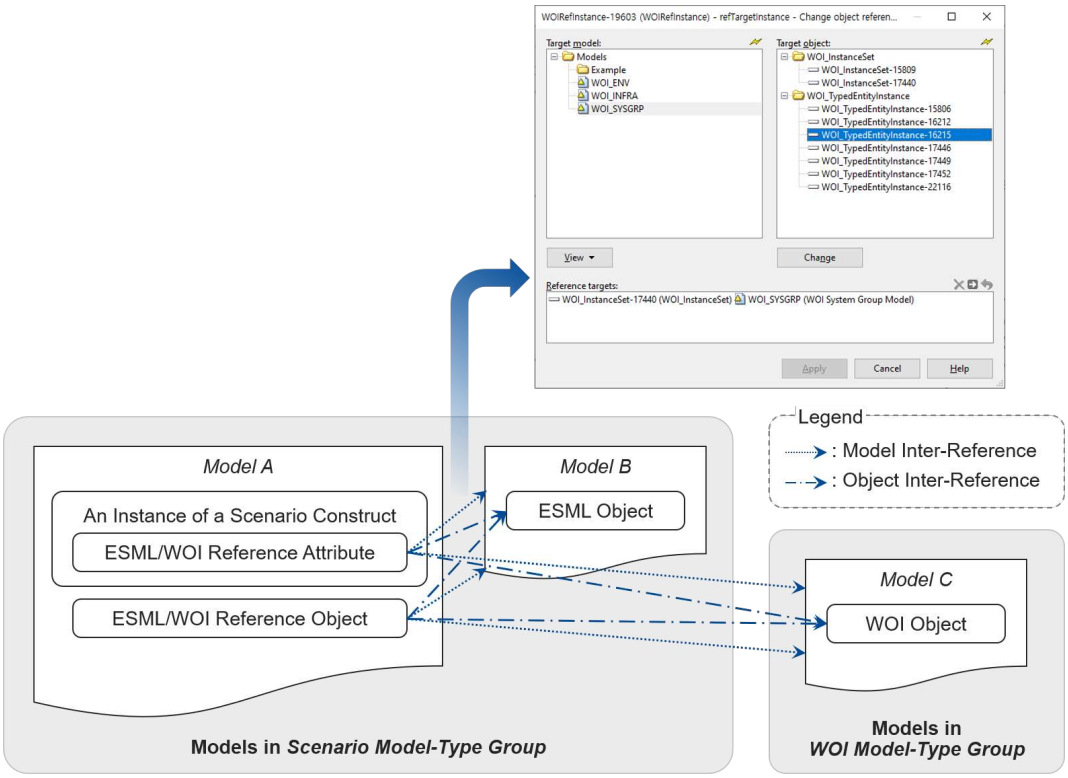

Figure 14: Support of inter-reference relationships between scenario models and WOI models

stracts these elements into *Actions* and supports engineers to further refine and specialize the action classes.

5.3.2 Model Type Definitions

The model types of *ESML* can be classified into two groups: one is *WOI Model Group* that models `_WOI_Construct_` to represent different aspects of the target WOI and its components; the other is the *Scenario Model Group* that models `_ESML_Construct_` to specify hypotheses and scenario elements, which references the WOI objects. As explained in Chapter 1, inter-reference relationships between scenario models and WOI models need to be supported by *ESML*. As Figure 14 illustrates, the proposed *ESML* supports both *object-to-model* and *object-to-object* inter-references. In summary, *ESML* provides 10 model types in total, and Table 4 summarizes the components (included classes and relation classes) of each model type.

5.4 ESMM Modeling Tool (*ESMM-Tool*)

5.4.1 ADOxx Meta-modeling Platform.

The *ADOxx Metamodeling Platform* was initially developed by the *OMiLAB* at the *University of Vienna* and is recognized as one of the most innovative tools for developing modeling methods, alongside frameworks like the *Eclipse Modeling Framework (EMF)*. This platform offers extensible features, libraries, and comprehensive support for the entire development process of a modeling method and creation of GUI-based modeling tools [12]. The framework supports two types of language formats for outputs: the *ADOxx Library Language (ABL)* and the *ADOxx Model Language (ADL)*. The *ADL* is used to describe and store modeled results such as diagrams, objects, and values, making it the pre-

Table 4: Concrete classes and relation classes defined in the *ESML* model types

Model	Model Type	Classes	Relation-Classes
WOI Model Group	**Target WOI Model**	{WOI, WOI_TypedEntityInstance, WOI_InstanceSet, MapModelArtifact, WOI_Constant, WOI_DataVar, WOI_VarDomain, WOI_TmpUnit, WOI_SptUnit, WOI_DimVar, WOIRefWOI, WOIRefContainer, WOIRefInstance, WOIRefSet, WOIRefVariable}	{WoiClassHasConstant, WoiHasDataVar, WoiHasInstance, WoiHasInstanceSet, WoiContainsElement, WoiHasMap, WoiHasUnit, MapHasDimVar, UnitHasDimVar, VarHasDomain}
	WOI Type Definition Model	{WOI_EntityType, WOI_DataType, WOI_EntityAction, WOI_StateVar, WOI_DataVar, WOI_Constant, WOI_VarDomain, ModelArtifact}	{TypeSubclassOf, TypeHasState, TypeHasData, WoiClassHasConstant, TypeCapableOf, ActionHasParam, StateTransition, VarHasDomain}
	WOI System Group Model	{WOI_SystemGroup, WOI_TypedEntityInstance, WOI_InstanceSet, WOI_Constant, WOI_DataVar, WOI_VarDomain, WOI_CommAction}	{ContainerHasInstance, ContainerHasInstanceSet, SetHasInstance, WoiClassHasConstant, ActionHasParam, VarHasDomain}
	WOI Infrastructure Model	{WOI_Infrastructure, WOI_TypedEntityInstance, WOI_InstanceSet, WOI_Constant, WOI_DataVar, WOI_VarDomain, WOI_CommAction}	{ContainerHasInstance, ContainerHasInstanceSet, SetHasInstance, WoiClassHasConstant, ActionHasParam, VarHasDomain}
	WOI Environment Model	{WOI_Environment, WOI_TypedEntityInstance, WOI_EnvFactorObjectInstance, WOI_InstanceSet, WOI_Constant, WOI_DataVar, WOI_VarDomain, WOI_CommAction}	{ContainerHasInstance, ContainerHasInstanceSet, SetHasInstance, WoiClassHasConstant, ActionHasParam, VarHasDomain}
Scenario Model Group	**Hypothesis Model**	{HypothesisSpec, ViewpointSpec, StmtAssumption, StmtAssertion, EvidenceArtifact, StmtProposition}	{ViewpointHasHypothesis, HypothesisHasAssumption, HypothesisHasAssertion, AssumptionHasConstraint, StmtHasEvidence}
	Scenario Method Model	{ScenarioMethod, WOIRefWOI, ESMLRefScenarioSuite, ESMLRefScenario, ESMLRefScenarioUnit, SpecRefViewpoint, SpecRefHypothesis}	{MethodToTargetWOI, MethodToRefElement, MethodToRefSpec, ScnElementToRefElement, RefElementToRefElement}
	Scenario Suite Model	{ScenarioSuite, ElementIOModel, ElementContextCondition, ElementContextConstraint, WOIRefInstance, WOIRefType, ESMLRefScenarioSuite, ESMLRefScenario, ESMLRefScenarioUnit, StmtInitialization}	{ParticipateIn, ScnElementHasPrecondition, ScnElementHasPostcondition, ScnElementHasConstraint, ScnElementToRefElement, RefElementToRefElement, ScnElementInitializedAs, ScnElementHasInput, ScnElementHasOutput}
	Scenario Model	{Scenario, ElementIOParam, ElementIOModel, ElementContextCondition, ElementContextConstraint, WOIRefInstance, WOIRefType, ESMLRefScenario, ESMLRefScenarioUnit, StmtProbability, StmtInitialization}	{ScnElementHasPrecondition, ScnElementHasPostcondition, ScnElementHasConstraint, ScnElementToRefElement, RefElementToRefElement, ScnUnitRefTransition, ParticipateIn, ScnElementHasInput, ScnElementHasOutput, ScnElementInitializedAs}
	Scenario Unit Model	{ScenarioUnit, ElementIOParam, ElementIOModel, ElementContextCondition, ElementContextConstraint, WOIRefInstance, WOIRefType, ESMLRefScenarioUnit, RandomGenerator, StmtOccurrence, StmtProbability, StmtInitialization}	{ScnElementHasPrecondition, ScnElementHasPostcondition, ScnElementHasConstraint, ScnElementToRefElement, RefElementToRefElement, ScnElementHasInput, ScnElementHasOutput, ParticipateIn, ScnUnitOccursAs, ScnElementInitializedAs}

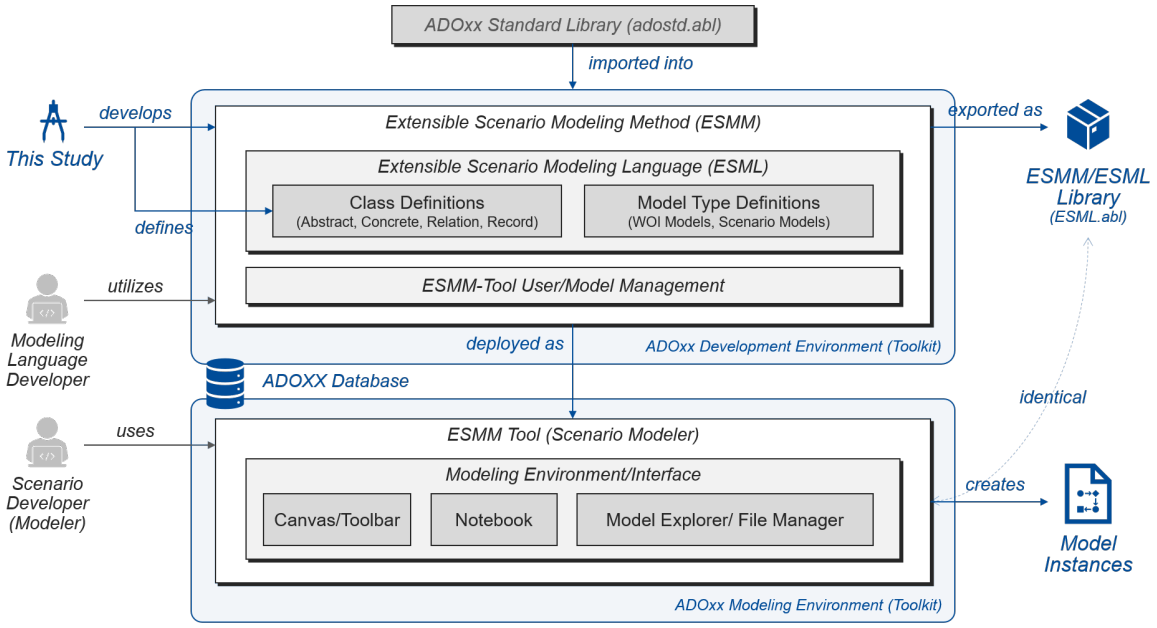

Figure 15: Overview of *ESMM-Tool* and *ADOxx* environment

ferred format for scenario designers and modelers to create and manage model instances. On the other hand, the *ABL* serves as a method-level library that stores all meta-level information of a modeling method, including class definitions, model type definitions, and optionally mechanisms and algorithms. By deploying and releasing an *ABL*-formatted library or tool, other method engineers can engage in refinement or extension activities, as it contains all the necessary data to serve as a reference modeling method.

5.4.2 Modeling Environment and Two Roles Who Use *ESMM/ESML*

Figure 15 briefly shows how *ESMM/ESML* was developed as a library and modeling tool on the *ADOxx* framework environment described above. This figure is largely divided into the *ESMM part*, which is implemented and operated on the *development environment*, and the *ESMM-Tool part*, which is made in the form of an actual modeling tool and executed on the *modeling environment*. In order for us to develop *ESMM*, the *ADOxx framework* must be installed, and a database must be running for storing a modeling method, models being created, and library and user information[13]. *ESMM* was distributed as a library or tool (i.e., serve as a reference scenario modeling method), and the latest version of the *ESMM Library* can be downloaded from *GitHub*[14].

In the figure, two arrows go out from the *ESMM* box. One is simply exporting and packaging the *ESMM* in *ABL*-formatted library (*ESML.abl*), and the other is deploying it as a tool through the database so that modeling activities can be practically performed. Both internally contain exactly the same modeling method information (i.e., *ESMM* component definitions), but the difference is that the *ESMM-Tool* can support actual modeling activities by providing a graphical user interface, realized by the middleware of the *ADOxx modeling environment*. The development environment is a space for method developers who want to newly develop modeling methods/languages or modify/extend existing libraries. If a *modeling language developer* wants to develop a domain-specific scenario modeling language (DS-SML) by extending our *ESML*, the process of importing the *ESML* library and adapting

[13]The database of *ADOxx Framework* is executed based on *MS-SQL*.
[14]*ESMM* Library Repository: https://github.com/AnthonyBaek/esmm

it to suit his or her own domain can be performed on the development environment in the same way as the above process. In this respect, *ESML* can be the starting point of the specialization process required to develop the DS-SML suitable for their domains or particular cases.

As a second role, *scenario developers* can perform actual modeling activities for scenarios/WOI specifications using the *ESMM-Tool*. The screenshots of utilizing *ESMM-Tool* from scenario developers' point of view are shown in Figure 16, and scenarios can be developed by developing the model types defined in *ESML*. When a modeler develops an actual model instance, he/she performs the instantiation of class objects on the canvas, modifies attributes, and creates relationships between the objects defined. Finally, the models created by the scenario developer on the tool can be exported as *ADL*-formatted models, XML files, or images[15]. Some of the results of actual modeling using the tool are introduced in Section 6.

6 Case Study

In this section, case studies will be conducted to evaluate whether *CSF* and *ESMM/ESML* proposed in the previous chapters can provide essential information and features as a reference method/methodology.

6.1 Research Questions

As discussed in Section 1, a crucial criterion for a method to be considered a reference method is its ability to capture the general semantic information present in real-world scenarios. This is accomplished through the use of constructs, which correspond to the *SVs* and are represented by the class definitions in *ESML*. Furthermore, in order for *ESMM* to serve as an effective guideline for scenario engineers as a reference method, it is important to assess its applicability to practical scenario specification and modeling activities. To address these considerations, the following research questions (RQs) have been formulated.

RQ1: To what extent do the *SVs* defined in *CSF* cover the semantic spaces of existing scenario development methods?

RQ2: Does *ESML* encompass scenario constructs and WOI constructs (i.e., class definitions) that effectively facilitate the development and analysis of real-world scenario models?

RQ3: How well can the constructs (i.e., model types and classes) defined using *ESMM/ESML* be extended to develop a domain-specific scenario modeling language?

The analysis of *RQ1* involves mapping the information from five existing methods to the primary *SVs* using a semantic mapping table, which shows how the semantics of existing methods are covered by our *CSF*. To answer *RQ2*, semantic mapping is also performed between 10 scenario instances from existing methods and the *ESML* classes. By answering *RQ3*, it examines the potential for extending *ESML* into a domain-specific scenario modeling language (DS-SML) for driving systems' scenarios. It investigates whether *ESML* actually supports the necessary specialization mechanism to define domain-specific objects, such as data, instances, actions, and behaviors.

6.2 Analyzed Scenario Development Methods and Scenario Instances

Scenario Development Methods. To evaluate the semantic space expressed by the *SVs* and *CSM* developed in this study, we use 6 scenario development/specification techniques suggested in the automated driving system (ADS) domain. Because the ADS domain, along with several critical system domains (e.g., military, aviation), is one of the most active field of research and standardization of scenario-based engineering (See Chapter 2), scenario methods employed in the ADS domain were selected as a representative application domain.

[15]Only XML- and *ADL*-formatted models are modifiable on the tool.

Table 5: Scenario instances used for the case study

Instance Category	Scenario Method	Scenario Instance(s)/Input(s)	Abstraction Level
Category A	*OpenSCENARIO-CARLA*	***SET_A:*** {LaneChange, FollowLeadingVehicle, PedestrianCrossing}	Logical/Concrete
	OpenSCENARIO-NHTSA	***SET_B:*** {PedestrianCrossing AtIntersection, TurningLeftAtRoundabout}	Functional/Logical
	OpenSCENARIO-SOTIF	***SET_C:*** {SuddenLaneChange}	Functional/Logical
	W.Chen, et al., 2018	***SET_D:*** {InsertionOfVehicles OfHighway}	Logical
	X. Zhang, et al., 2020	***SET_E:*** {StraightMotorwayYJunc, STATS19_Accident} ***SET_F:*** {EuroNCAP_CCRs}	Logical
Category B	*Euro-NCAP AEB C2C Test Test Procedure Description*	***NCAP_CCR_TEST_SET:*** {NCAP_Test_CCRs, NCAP_Test_CCRm, NCAP_Test_CCRb}	Logical/Concrete
Category C	*SESAR Engage KTN Drone Flight Mission*	***BVLOS_DRONE_TEST_SET:*** {UCS_1_StateSurveillance, UCS_2_MedicalSupplyMission, UCS_3_OffshoreInspection, UCS_4_UrbainAirMobility, UCS_5_CoastguardSearchRescue, UCS_6_WindfarmInspection, UCS_7_PackageDelivery, UCS_8_FireandRescueService}	Functional/Logical

The first and second methods are *ASAM OpenSCENARIO* (v2.0), a standard for testing and validation of ADSs, and *PEGASUS*, a methodology of scenario-based safety analysis and testing. These methods define engineer-friendly domain-specific scenario description language for expressing driving scenarios at multiple abstraction levels. Both methods also support the development of scenario execution models (i.e., test/simulation scenarios) for industrial application. Additionally, other methods proposed by B. Schutt *et al.* [31], J. Bach *et al.* [2], C. M. Richard *et al.* [30], and D. J. Fremont*et al.* [14] were analyzed in terms of scenario constructs from a modeling perspective. The first four methods develop scenarios for the test development, while the fifth method aims for scenario-based task analysis and workload estimation, and the sixth method aims to propose a probabilistic programming language for scene generation.

Scenario Instances. In order to analyze the overall feasibility and applicability of whether the proposed *ESMM/ESML* can be used (i.e., useful) for actual scenario development, inputs of real-world scenario instances are essential. Because standardization efforts are being most actively conducted in the ADS domain, many scenario(-relevant) instances are publicly available and can be obtained by this study. Instances introduced in Table 5 are categorized into three groups (*Category A*∼*C*), according to the collection method of scenario instances. Instances in *Category A* are based on driving scenarios collected from the web and literature, while *Category B* are scenario instances based on test procedure descriptions defined by the *Euro NCAP* test protocols. Finally, as part of *Category C*, we include a

group of instances to assess the universal extensibility of *ESML* across different application domains. Specifically, we present eight instances of drone flight mission scenarios in the unmanned aerial vehicle (UAV) domain.

6.3 RQ1: Expression of Scenario Semantics using *SVs*

This RQ is answered by Table 6, and semantics of *ESMM* are mapped and compared with the languages of existing scenario development methods introduced in Section 6.2. For better readability, similar *SVs* centered around primary *SVs* were merged to identify data corresponding to 3 method-level *SVs*, 3 suite-level *SVs*, 8 scenario-level *SVs*, and 7 event-level *SVs*. Similar to the literature review, data was manually extracted and analyzed based on keywords from exemplary scenario instances and explanation introduced in each method's representative publication(s). Analysis of the *SVs* does not directly show whether a scenario development method is superior or more effective than other methods. Instead, it provides information on which semantic domain each method primarily focuses on. Because the *SVs* were defined by investigating the actual scenario instances, we expect conceptual data used in the existing methods can be adequately analyzed at meta-class levels. The information contained in each cell of the table can be viewed as a class or instantiated data for scenario specification.

The investigated methods contain different types of data depending on the engineering activity, phase, and purpose even when applied in the same application domain. As described above, a set of conceptual data of a method is considered as a semantic domain for scenario specification. Therefore, it is helpful in establishing the general requirements needed for appropriately employing a scenario method. For example, C. M. Richard *et al.*'s approach provides semantics (e.g., workload demand, bottlenecks) focused on engineering activity of scenario-wide task analysis, but does not provide an actual test development. On the other hand, J. Bach *et al.*'s approach supports visualization tools (scenario editors) to effectively support the development of test scenarios, while other methods support the specification by providing formal/textual scenario description languages.

Based on the comparison, we could carry out comparative analysis of scenario semantics of the investigated scenario development methods, with respect to four levels (method, suite, scenario, and unit/event).

- *Method-level.* Although the method-level data shared similarities, the level of abstraction supported by *SpecMethod* and the observation/evaluation/validation criteria of *ExecMethod* differed. Most methods in the ADS domain utilized means of generating logical or concrete scenarios from abstract/functional scenarios, which shows the necessity of supporting multiple abstraction levels for both specification and execution. Also, the *SVs* that capture the goal-oriented nature can be leveraged to analyze and define data at the method level. This includes the explicit definition of hypotheses for scenario development and testing, as well as the ability to define assumptions and evidence for method selection and utilization.
- *Suite-level.* Suite-level data could be classified according to the viewpoints, which showed significant difference in inputs of each method. The widely used viewpoints in the automotive domains were typically an ADS (i.e., ego vehicle) and traffic infrastructure. The ontologies of target vehicle, infrastructure, and environment were defined differently depending on the viewpoint. In addition, the composition method to coherently formulate multiple scenarios decided scenario classification criteria (e.g., ordinary-critical, baseline-alternative) within a scenario suite according to *MethodPurpose* at method-level. Additionally, unlike simple programs defined at a functional or logical levels, the suite-level inputs on physical setting, environment, and regulatory artifacts were usually required. When data-driven approach is supported (*PEGASUS Method*, C. M. Richard *et al.*'s), historical data related to driving function and ADS users are also often required to develop a test suite. Compared to the first five methods, *Scenic*, proposed by D. J. Fremont *et al.*, is more specialized to develop realistic scenario suites based on prescribed scenes, thus it requires more specific requirements and data to create the suite-level scenario data.

- *Scenario-level.* Based on the scenario-level inputs of ADS scenarios, the detailed specification of environmental context each scenario faces (or interacts with) were crucial. All the listed methods utilized the environmental and operational context such as *weatherCondition* and *stationaryCondition* to process the runtime context of each scenario. These data are used to define scenario-level condition (e.g., *initialCondition*, *terminalCondition*) and constraints (e.g., *hardware/software Constraints*, *regulatoryConstraints*).

  The most central data at the scenario level is provided by parameterization. Logical scenario utilizes parameter range to specify the scenario dynamics, and concrete scenario provides a set of concrete parameter values within the range. Additionally, due to the characteristic of ADS scenarios which defines the dynamics of participants according to the flow of time within a region, temporal and geospatial abstraction are considered as important scenario-level data. Based on the target timeframe/timeline of each scenario, temporal abstraction is detailed through phases, stages, or milestones. Geospatial abstraction mainly details a road network (e.g., layout, geometry) and geospatial changes to effectively represent scenario-level participants' movements.

- *ScenarioUnit/Event-level.* Event-level data mostly contained similar semantic domain, including behavioral, temporal, spatial, and uncertainty properties. According to the each specification method, the unit behavior of an event and its granularity may differ, which are often act, action (e.g., maneuver), communication (e.g., interrupt, stimuli), and activity. At the scenario unit level, behavioral information is commonly represented by specifying event occurrences and their associated properties. The proposed *SVs* encompass a wide range of elements at this level, including input/output, conditions, temporal/spatial data, and uncertainty. Because an event is the most generally used unit accessing actual states or data of scenario participants (i.e., entities), the semantic domain of each information on unit event can be varied depending on each method.

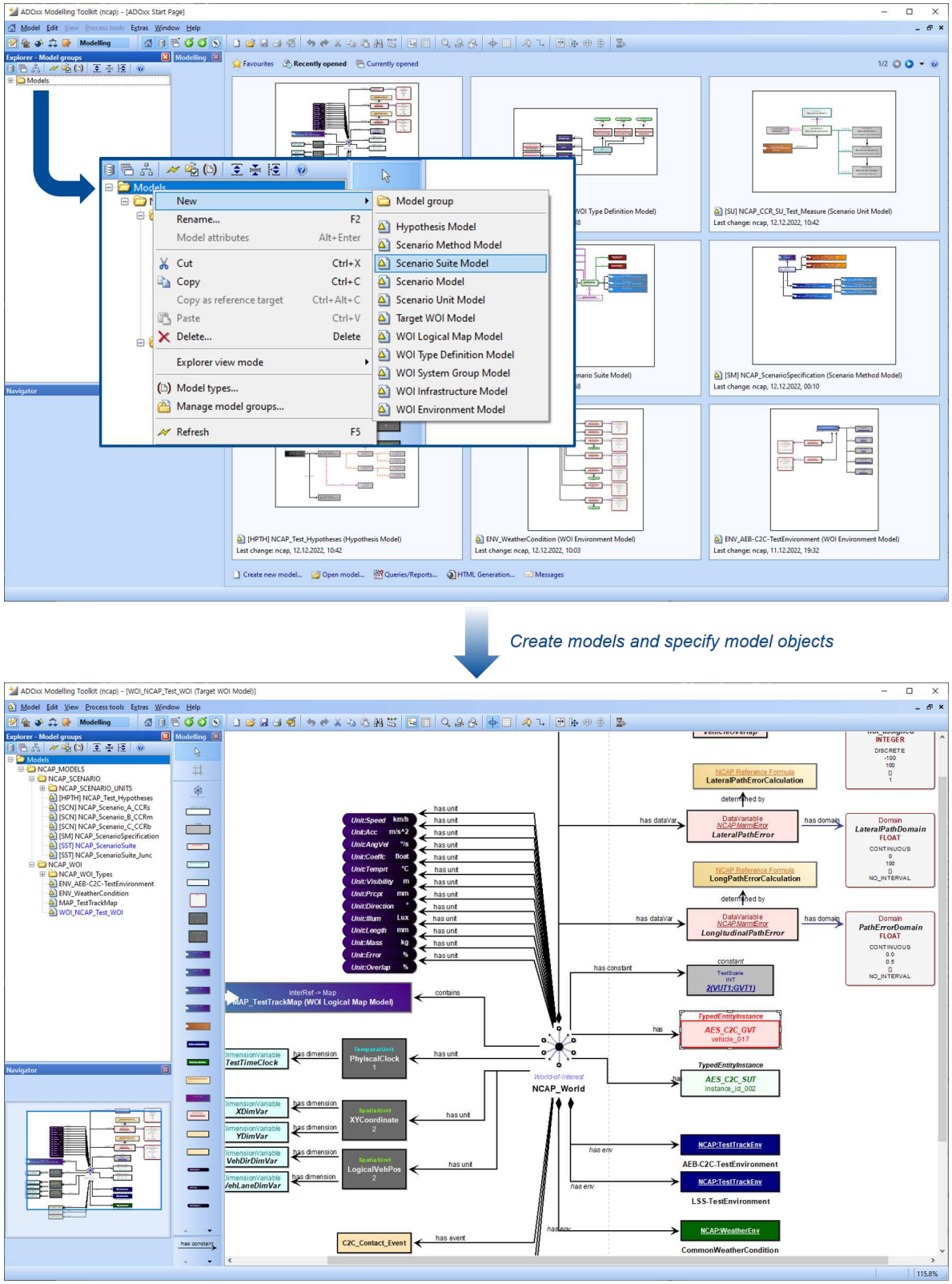

Figure 16: Execution and use of *ESMM-Tool* on *ADOxx Modeling Environment*

Table 6: Analysis of scenario specification methods in terms of primary scenario variables

	Source	*ASAM OpenSCENARIO V2.0*	*PEGASUS Method*	*[Barbara Schütt, 2020]*	*[J. Bach, 2016]*	*[C. M. Richard, 2006]*	*[D. J. Fremont, 2019]*
	Instances	Cut-in, SlowPrecedingVehicle, EndOfTrafficJam, TrafficJam, DoubleLaneChanger, Overtaking, SynchronizedArrival, CloseVehicleCrossing, FastOvertakeWithReinitialization,	PEGASUS Exemplary Scenarios (Tesla Model S AccidentScenario) Cut-in, IncidentEnvironment	UrbanIntersection	Single-lane Approach-and -Follow	Left Turn onGreen Light Straight on Yellow Light Right Turn on Red Light Stop on Red Light	GTAV Scene Description
M	MethodPurpose	Test-basedSimulation, TestCompleteness	Test-basedSafety Analysis, Verification & Validation	Test Development	TestDevelopment	Scenario-WideTask/Segment Analysis (& Resource Estimation), Planning, Safety Analysis	TestDevelopment (Generation of Synthetic Data Set)
	SpecMethod	AbstractionLevel (Abstract/Concrete), DecompositionOfFlows	AbstractionLevel (Functional/Logical/Concrete)	AbstractionLevel, Modularity (Database, Library)	Model-basedSpatial and Temporal Abstraction, Visualization	SegmentDiagram, ScenarioDetails	Scene,Distribution (Configuration)
	ExecMethod	Simulation-basedTesting, Measurement (Channel), Metric, AcceptanceCriteria, SafetyChecker, Evaluators, Coverage	Testing,Prototyping, Pass/Fail Criteria	Simulation-based Testing, Pass/Fail Criteria	(Simulation-based) Testing		SimulationGame, Training, Testing, Debugging
St	Viewpoint& Ontology	ADS/Traffic-focused	ADS-focused, Safety-focused	ADS-focused Actor (Vehicle, Pedestrian)	ADS/Traffic-focused Vehicle, Participant	Infrastructure/TrafficScene-focused, ADS-focused	Scene-focused
	Inputs	Map/Template,Scene (Graph), Configuration, InitAction (Global/User-Defined/Private), FunctionalGoals, DomainModel	Scene, SafetyArgument, FailureTree, RegulatoryElement, AcceptanceModel, RegionOfInterest, Virtual/RealDataSource, RoadUserData, AccidentModel	TrafficSequence, Setting (using Functional Scenario or Misc)	Scene, RegulatoryElement, Domain Model, Rule Sets, ManeuverType, PerceptionLayers	RegulatoryElement, Historical/Operational Data	Scenario(Default) Requirements, Property,ImagesOfCars
	Composition	Scene-based,Causality, Baseline-Alternative	Ordinary-Critical	Situation-based	Main-Alternative, Causality	ScenarioSegments	
	Constituent-Events	Serial/Sequential,Parallel, Synchronized, Event-driven, XOR	BaseEvent, Faulty/FailureEvent	EventConnection, Join, Sync	Act-based,Event-driven	Sequential	
	Target	TestDescription	Description ofAutomated Driving Functions, Criticality	Multi-levelBehavior Description	BehaviorDescription	Task	
Scn	Inputs	ScenarioMeta-Data, Story,Route, Procedure, Entities/ScenarioObjects (Vehicles (Movable, Stationary), Driver), Parameters (Distribution), File, Weather, StationaryCondition, ManeuverGroup, CatalogReference	EnvironmentCondition (Weather), SAELevel, Risk, Actor (Vehicle), Sensors, Parameters (FrequencyDistribution)	Weather, Actor(Wheeler, Pedestrian), Parameter (Range)	Scenery (RoadNetwork), Participant, Route, Situation, Weather	Vehicle,Scenario Segments Assumption, Workload Demand (with Bottlenecks), Configuration, Milestones, Parameters	Objects, ScenarioFile, Statement, GlobalParameters
	Outputs	ScenarioLibrary/Element, ParameterMeasurementResults	TestSpecification, Probability and Security of occurrence of (harmful) outcomes	Scenario File	Test Cases, Recorded Data	WorkloadProfile	
	Condition	EnvironmentCondition, EntityConstraints Software/Hardware Constraints, TerminationCondition,	EnvironmentCondition, Traffic		TransitionCondition (Situation)	TrafficCondition, DriverCondition, TimeConstraints, ResourceConstraints	Hard/SoftConstraints, Traffic/RoadCondition
	Temporal	ScenarioTime, Timeline	Time-Continuity, Start/End Time			Timeline/TimeFrame, Temporal Milestones, Interval	
	Geospatial	SpatialAbstraction of RoadNetwork (Intersection, Lane, Highway, Trajectory)	SpatialAbstraction of Road (Lane, Highway, TrafficDensity, Geometry)	SpatialAbstraction of RoadNetwork	SpatialAbstraction of RoadNetwork	Region,Layout,Infrastructure	EgoObjectAbstraction Geometry, Distribution
	Changes & Uncertainty	ArbitraryPosition/Direction, StochasticDistribution	OccurrenceLikelihood			WorkloadRatings	Mutation,Noise, ProbabilityDistribution
	Input	EventParameter, ControlStrategy	Severity,Controllability, Criticality, EventParameter (with Unit)	ActionParameter	EventSource	Equations,Variables, AssumedValues	
	Output	ExecutionCount					
Evn	Condition	EventCondition(StartTrigger, StopTrigger), ActionCondition, Parameter/ActionConstraints		ActionCondition (ConditionNode)	ManeuverCondition (Distance, Time, Speed), ParticipantState	ManeuverCondition, EnvironmentCondition	EventFrequency
	Behavioral& Interaction	Act, Actor,Event, Maneuver, EnvironmentAction, Calculation	Event, Action,Maneuver	Actor, Action,Maneuver	Participant,Act, Maneuver, Perception, Interaction, ADAS Function	Actor,Activity, Event Perceptual/Cognitive/Psychomotor Task, V2V Interaction (Adjacent)	Actor (Ego),Vision
	Temporal	Duration,(Relative) Time, TimeReference	Time		Time	(Relative)Timing,Duration	
	Geospatial	(Cartesian)Coordinate, Speed (Relative/Absolute), Position, Direction/Angle, Distance	Coordinate,Distance, Velocity/Speed, Orientation	Radius,Direction, Velocity	Lane,Position, Distance	Distance,Direction, Speed	LocalCoordinate, Offset, Direction, Position, Spot, Distance
	Uncertainty	StochasticEvent	ProbabilityOfOccurrence, Controllability, Fluctuation, Variability	ProbabilityDistribution (of Params)		Distribution of Information Perceived	ProbabilityDistribution

Table 7: Expression of scenario data of real-world scenarios using *ESML* constructs

ScnSet	WOI Semantics			Scenario Semantics			Uncovered
	World/Scene	EnvInfra	SysOntology	Abstraction	Participant	Context	
SET_A	RoadNetwork	RoadEnvironment (Time,Weather, RoadCondition), EnvVehicle, RoadNetwork	VehicleCatalog, ControllerCatalog, ManeuverCatalog	Storyboard-Story-Act-Maneuver-Event-Action	Actor/Vehicle, Pedestrian	RoadCondition, ElementState, EntityCondition, ValueCondition, Dist/PosCondition	Animation, BoundingBox, Road Geometry
SET_B	Intersection/ Roundabout	Weather Condition	ODDElementRef	Scenario-Maneuver-Event/Interaction-Maneuver	RoadwayUser, Non-RoadwayUser	TrafficCondition, RoadCondition, BoundaryFailure, Condition	Detection Algorithm, Road Geometry
SET_C	Motorway	Weather Condition (Sight)		SceneGraph-Scene-Event-Action	EgoVehicle, EnvVehicle	Event/Action, Condition	
SET_D	Highway (HighwayOntology)	Weather Ontology	VehicleOntology, ControlActions	Scenario-Situation-Action	ADS, MobileElement, StaticElement	WeatherCondition, ActionCondition	Road Geometry
SET_E	Motorway/ Junction	Weather Condition	AgentObject	Scenario-Phase-Action	EgoVehicle, EnvVehicle	EntityCondition (WHEN)	Road Geometry
SET_F	TestEnvironment		System Specification		VehicleUnderTest, VehicleTarget		

6.4 RQ2: Expressiveness of *ESML* Constructs

If the *RQ1* analyzed the semantic space defined in scenario development methods to validate the expressiveness of *SVs*, this *RQ2* evaluates the expressive power of *ESML* constructs based on scenario instances actually developed with several existing description langauges. The *RQ2* performed analysis on the scenario development methods and their instances in *Category A* of Table 5. The above three sets are XML-based *xosc files*[16] written based on *OpenSCENARIO* (*SET_A*) and scenarios defined as use cases in official documents provided by the *OpenSCENARIO* user guide (*SET_B*, *SET_C*). The other three sets below are scenario instances developed in the studies by W. Chen, *et al.* [7] and X. Zhang, *et al.* [45]. Since each instance had different levels of abstraction, the analysis through them can be regarded as a case study of the different types of scenario instances used in practice. Table 7 summarizes WOI and *ESML* classes at the abstract class level, and A[17] shows all semantic mapping contents at the concrete class level through 6 tables.

First, the world modeling method such as ontology must be supported so as to effectively express and interweave the world model's instances and scenario elements/data. Focusing on these properties, the first comparison criterion can be how the scenario description languages express the WOI semantics (See *WOI Semantics* in Table 7). In *SET_A*, *SET_E*, and *SET_F*, the model for the world was not explicitly supported at the level of the scenario language, but most of them received external catalogs or system models as inputs to use system/environment data. However, based on the importance of ontology in scenario modeling, it was confirmed that some scenario development methods are supporting the development of ontology-based models at the language level. Like the *NHTSA scenario*, the automotive domain often uses *Operational Design Domain (ODD)* that defines system and environmental entities used for the automotive engineering. More specifically, the *ODD* is a taxonomy-based standard that provides essential concepts necessary to describe conditions for safety driving, and supports ontological analysis on scenery, environmental conditions, and dynamic elements. In the fourth group, W. Chen, *et al.*'s study, defines domain-specific ontologies, such as highway, weather, and vehicle ontologies, to effectively describe a target world and to secure high reusability of the ontologies within the domain. From this point of view, it can be analyzed that the performance of ontological analysis through the development of the world model group of *ESML* can guide the ontology-based scenario development.

The second most notable part was that the dynamics represented by the scenarios was different for each

[16] **.xosc*: The file format used in *OpenSCENARIO*

[17] Each *Case* of the tables refers to the scenario instances of from *SET_A* to *SET_F* in order.

scenario description languages (See *Scenario Semantics - Abstraction* in Table 7). For example, the *OpenSCENARIO* defines their scenarios based on the '*Storyboard-...-Maneuver-...-Action*' hierarchy, and a *SceneGraph* is defined as a core unit in the *SOTIF* scenarios. Based on this, it was confirmed that different scenario languages have defined their own hierarchical structures to express scenario semantics, and the lowest level units are defined domain-specifically. Since all the analyzed instances are developed in the ADS domains, the lowest level units are often defined as *action* or *maneuver* of dynamic entities. From the *ESML* construct point of view, engineers can flexibly select the granularity and behavior abstraction of the scenario units without limiting what information these units and suites contain. Therefore, *ESML*'s scenario hierarchy divided into '*Method-ScenarioSuite-Scenario-ScenarioUnit*' can be expected to effectively respond to various abstraction levels of real-world scenarios.

Lastly, although most of semantics expressed by the scenario instances can be mapped to (i.e., covered by) classes defined in *ESMM*, some data could not be fully covered by the *ESML*. The semantic space that is not covered by *ESML* is organized on the far right of Table 7, and there are three main reasons why these types of data could not be expressed with *ESML* constructs. First, data such as *animation*, *bounding box*, and *object group detail* are elements defined for the purpose of providing visualization of scenario objects in simulation engines. Second, because *ESMM/ESML* treats *Actions* as a black-box, it was unable to express function-level internal data or logic. In the case of the *NHTSA* scenario's *detection algorithm*, it is inevitable to express it as an abstraction as an external reference for the algorithm or just to abstract it as '*detectAction*'. Third, it was not possible to directly cover the *road geometry* models/data (e.g., traffic map) used in various scenario languages. In fact, for the scenario instances under investigation, the geometry-data/models were developed as external files and were imported by the scenarios, and the data was using a method to express a very specific road scenery data for each scenario. By taking this approach, we can conclude that *ESMM/ESML*, being a generally scalable framework, does not take into account inputs and content that need to adhere to specific requirements of a particular domain.

Through this semantic mapping of the *RQ2*, we can first gain insight into what variables can be mapped to the data of real-world scenario instances, which were developed by existing methods. The other implications can be checked through specific semantic mapping data for each concrete class shown in A. From this result of the semantic mapping, we were able to confirm that *ESML* itself can sufficiently express (and have possibility to be extended to express) the semantic space of real-world scenarios with only the modeling language constructs before it has extensibility. For some uncovered semantics, the '*essence of a scenario*' can be the ground to explain that the exclusion of such special data is natural (and a practice) while developing scenarios. According to the scenario definition, the common information that should be included in a scenario can be summarized as: (a) temporal development of dynamics/possibilities (i.e., path), (b) context(s), and (c) a participant(s). Here, the temporal development can be expressed as a path of one or more scenario units by a scenario model, (b) can be expressed as a (parameter) range through conditions, constraints or domains of elements in the model, and (c) can be expressed using the ontology through the WOI models.

6.5 RQ3: Extensibility of ESML Constructs for Developing a Domain-specific Scenario Modeling Language

The final RQ aims to assess the extensibility of *ESML* and examine its potential as a domain-specific scenario modeling language. Figure 17 provides an overview of the process employed to address *RQ3*. The evaluation utilizes the *NCR_CCR_TEST_SET* and *BVLOS_DRONE_TEST_SET* introduced earlier and extends *ESML* by analyzing accessible documents (i.e., *Euro NCAP test description*, *BVLOS drone use case scenarios*). Through the extension and specialization of *ESML* constructs (classes and model types), an attempt is made to develop domain specific scenario modeling languages known as *Scenario Modeling Language for EuRo NCAP (SML-ER-NCAP)* and *BVLOS Drone Missions (SML-BVLOS-FLIGHT-MISSION)*, respectively.

Table 8: Identification of domain-specific classes and instances from *Euro NCAP Test Procedures*

Construct	**ESML Class**	**Domain Classes**	**Class Instances (Objects)**
_ESML_Construct_ (Scenario Element)	ScenarioSuite	*VehicleTestScenarioSuite*	{*AEB_C2C_TestSuite*}
	Scenario	*Car2CarTestScenario*	{*C2C_RearStationary_TestScenario, C2C_RearMoving_TestScenario, C2C_RearBraking_TestScenario*}
	ScenarioUnit	*OnRoadTestEvent*	{*PedestrianPassing, OnRoadCollision*}
	ElementContext	*TestCondition*	{*TestPrecond, DrivingCond, TestTerminationCond*}
		DrivingConstraint	{*MaxSpeed, ProhibitedAction*}
	Specification	*FormulaSpecification*	{*LateralPathErrorSpec, LateralOverlapSpec*}
	ExternalArtifact	*ReferenceArtifact*	{*MeasurementReference, VehicleTargetReference, VehicleDataMeasurement, ReferenceSystem*}
_WOI_Construct_ (WOI Element)	WOI	*VehicleTestWOI*	{*AEBC2CTestWOI*}
	WOI_Env	*VehicleTestEnvironment*	
		WeatherEnvironment	{*Temperature, TrackTemperature, Visibility, Wind, Precipitation, Illumination*}
		PredefEnvEvent	{*WrongPathEvent, IndividualCarAccident, CarToCarCollisionContact*}
	WOI_Map	*TestTrackMap*	{*AEBC2CTestTrackMap*}
	WOI_EntityType	*VehicleEntityType*	{*VUT, GVT, SOV*}
		PedestrianEntityType	{*PedestrianEntity, BicycleRiderEntity*}
	WOI_EntityAction	*VehicleAction*	{*StartMoving, Steer, Accelerate, Decelerate, StopMoving*}
		VehicleDrivingAssistAction	{*AutonomousEmergencyBraking, ForwardCollisionWarning, DynamicBrakeSupport, AutonomousEmergencySteering, EmergencySteeringSupport*}
	WOI_Unit	*SpeedUnit*	{*VehSpeedUnit, WindSpeedUnit*}
		AccelerationUnit	{*VehAccUnit*}
		AngularVelocityUnit	{*VehAngVelocityUnit*}
		CoefficientUnit	{*PeakBreakingCoeffUnit*}
		TemperatureUnit	{*AmbientTempUnit, TrackTempUnit*}
		VisibilityUnit	{*HorizontalVisibilityUnit*}
		PrecipitationUnit	{*TrackPrecipitationUnit*}
		DirectionUnit	{*VehDrivingDirUnit, WindDirUnit*}
		IlluminationUnit	{*AmbientIlluminationUnit*}
		LengthUnit	{*VehWidthUnit, VehLengthUnit*}
		MassUnit	{*VehMassUnit*}
		ErrorUnit	{*LongPathErrorUnit, LatPathErrorUnit*}
		OverlapUnit	{*C2CLateralOverlapUnit*}
	WOI_DataVar	*VehiclePropertyVar*	{*VehWidth, VehLength, VehMass*}
		VehicleActionVar	{*SpeedVar, AccVar, AngVelocityVar*}
		TestEnvCondVar	{*AmbTemperature, TrackTemperature, WindSpeed, WindDirection, AmbIllumination*}
		MeasurementErrorVar	{*AllowedErrorVar*}
	WOI_Constant	*MapConditionConst*	{*TrackDryness, TrackPavement, TrackUniformity, TrackSlopeLevel, TrackPBC*}
	WOI_StateVar	*VehicleTestStateVar*	{*Idle, OnDriving, OnCollision*}

Table 9: Identification of domain-specific classes and instances from *Category C*

Construct	**ESML Class**	**Domain Classes**	**Class Instances (Objects)**
_ESML_Construct_ (Scenario Element)	ElementContext	*VolumeContext*	{*X, Y, Zu, Za*}
		PhaseContext	{*U2, U3*}
	Scenario	*FlightProcedure*	{*TakeOffProcedure, FlightBackProcedure*}
		SafetyProcedure	{*BuiltinSafetyProcedure*}
	ScenarioUnit	*AbnormalFlightBehaviorSU*	{*AbnormalExcursion*}
		ExternalAttackSU	{*ExtDroneInfringement, SignalSpoofing, MaliciousCode*}
		DataSignalErrorSU	{*LossDataIntegrity, SignalInterference, HighElectroMagnetic, HighSignalDensity*}
		CommLinkEventSU	{*DataTransmissionFailure, LinkBroken, LinkRecovered*}
		InternalMalfunctionSU	{*OnboardMalfunction*}
		HazardSU	{*LossOfControl, MidAirCollision, LALT, SEC, LVLOS*}
_WOI_Construct_ (WOI Element)	WOI	*DroneOperatingEnvironment*	{*CitySurveillanceWOI, MedicalSupplyWOI, OffshoreInspectionWOI, UrbanAerospaceWOI, CoastguardWOI, WindfarmWOI, PackageDeliveryWOI, FireRescueWOI*}
	WOI_Infra	*CommuniationInfra*	{*GroundCommChannel, C2CCommChannel, RadioCommChannel*}
		FlightInfra	{*TakeoffLandingInfra, ExclusionZone*}
	WOI_Env	*AirTrafficEnvironment*	{*AirspaceTrafficEnv*}
		FlightAreaEnvironment	{*HighRiseBuildingEnv*}
		MeteorologicalEnvironment	{*Fog, Wind, Visibility*}
		HumanEnvironment	{*Crowd*}
		PredefEnvEvent	{*BirdStrike, Smoke, Fire*}
	WOI_Map	*USpaceMap*	{*UrbanFlightSpaceMap, OffshoreFlightSpaceMap, CoastguardFlightSpaceMap, WindfarmAreaMap*}
		OnMapData	{*PrearrangedWaypoint*}
	WOI_SptUnit	*AirborneLocUnit*	{*AirXYZCoordinateUnit*}
		GroundLocUnit	{*GroundXYCoordinateUnit*}
	WOI_EntityType	*PilotEntity*	{*RemotePilot(, AttackerPilot)*}
		DroneEntity	{*BVLOSDrone(, ExternalDrone, AttackerDrone)*}
		DeviceEquipEntity	{*Multi-Rotor, RemotelyPiloted, DAAEquipment, FlightAssistSensor*}
	WOI_DataType	*FlightActivityData*	{*SurveillanceData*}
		CommunicationData	{*PilotRequestData*}
		WeatherData	{*WeatherInfoData*}
	WOI_Constant	*FlightConstant*	{*SpeedLimit, AltitudeRestriction*}
	WOI_EntityAction	*DroneFlightManeuver*	{*StationaryFlightManeuver, RouteFlightManeuver, AreaFlightManeuver, AutonomousFlightManeuver, DitchingManeuver*}
		DroneUnitFlightAction	{*TakingOffAction, DescendingAction, LandingAction, MissionPauseAction, MissionResumeAction, DAAAction, EmergencyLandingAction*}
		DroneMissionAction	{*TelemeteringAction*}
		DroneCommAction	{*TelemetryTransmissionAction*}
		PilotAction	{*SeizeEquipmentAction, EmergencyNotiAction, MonitoringAction*}
	WOI_Unit	*DroneStatusUnit*	{*DroneFuelUnit, DroneSpeedUnit, DronAltUnit, DroneVisualSightUnit*}
		DataSignalUnit	{*DataIntegrityUnit, ElectromagneticIntensityUnit, InterferenceUnit, SignalIntensityUnit*}
		FlightEnvUnit	{*WindSpeedUnit, FogDensityUnit, VisibilityUnit*}
	WOI_DataVar	*DroneStatusVar*	{*VisualSight, Altitude, Speed*}
		CommunicationVar	{*DataSignalIntensity*}
		FlightActivityVar	{*RouteDeviationRate, PowerConsumptionRate*}
		FlightAreaVar	{*AreaVolume, AreaType, AreaDensity/Population*}

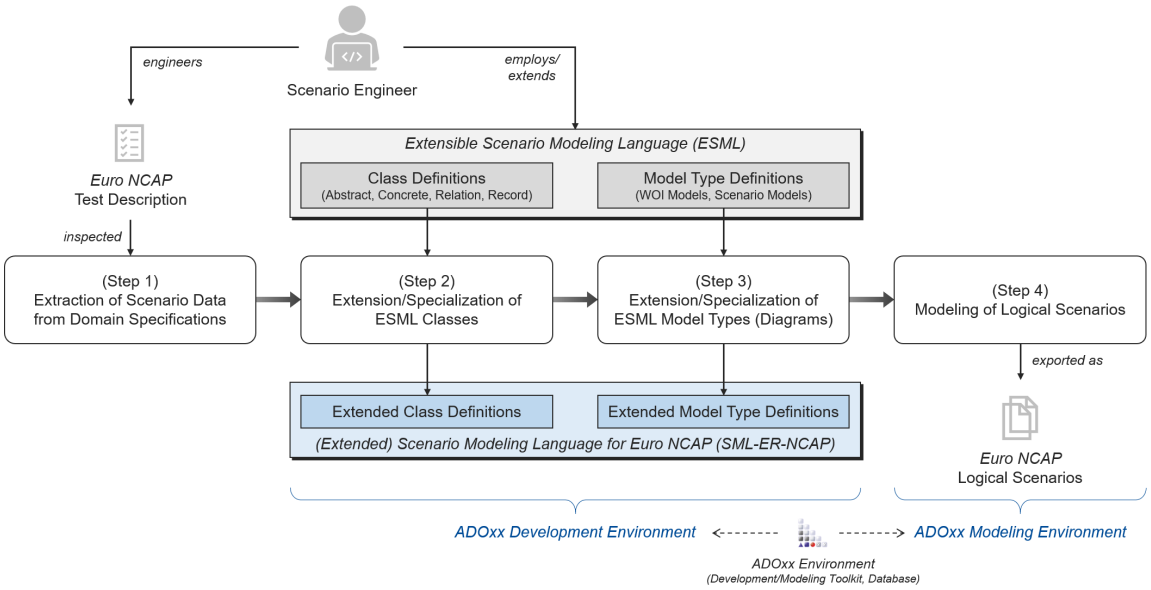

Figure 17: Extension of *ESML* into a scenario modeling language for *Euro-NCAP* test description

Step 1. Following each step, the extension process is explained as follows. In *Step 1*, the *Euro NCAP* test description, which describes tests of *autonomous emergency braking (AEB) function* and *forward collision warning (FCW) function* in a car-to-car situation, is received as an input specification. This step extracts domain-specific data from the given specification to understand the semantic space of the target domain and identify constructs that need to be extended/specialized to cover the semantic space. This extraction step takes two sub tasks, which are (i) extraction of scenario data and (ii) extraction of WOI/domain data (i.e., ontology). The scenario data may require the specialization of classes under `_ESML_Construct_` to describe domain-specific scenario elements, contexts, i/o, and statements. The WOI and domain data may require the specialization of classes under `_WOI_Construct_` to build a domain-specific ontology, which consists of domain types, objects/entities, units, variables, and events. The Table 8 provides a summary of the domain classes and instances extracted in *Step 1*. It allows us to identify the necessary class definitions under *ESML* abstract classes to express domain instances.

Step 2. In this step, a *modeling language developer* (See Figure 15) needs to extend the classes defined in *ESML*. First, the engineer imports the *ESML* library and its internal definitions on the *ADOxx Development Environment*. To do this, he/she should access to the database with an authenticated account and import the published *ESML Library*[18] into the development toolkit and name it as *SML-ER-NCAP-Library* for further manipulation and extension. The engineer can enter the *library management* mode and add domain-specific classes or class attributes into the existing *ESML* class definitions. Based on the extracted data from *Step 1*, specialized classes (i.e., domain-specific classes) can be newly defined considering the overall class hierarchy. Not only concrete classes can be added into the existing library, but relation classes (e.g., *has:* $A \rightarrow B$) also can be built on the library to represent class-to-class relationships. If a class is completely newly defined, internal attributes should be specified, including graphical notations.

Step 3. In *Step 3*, the developer can extend the existing *ESML* model types by including domain-specific classes, which were defined/extended in the previous step. Only instantiable classes and relation-classes can be included in the model types, and it is recommended to specify graphical nota-

[18]Current version of *ESML Library*: 1.15

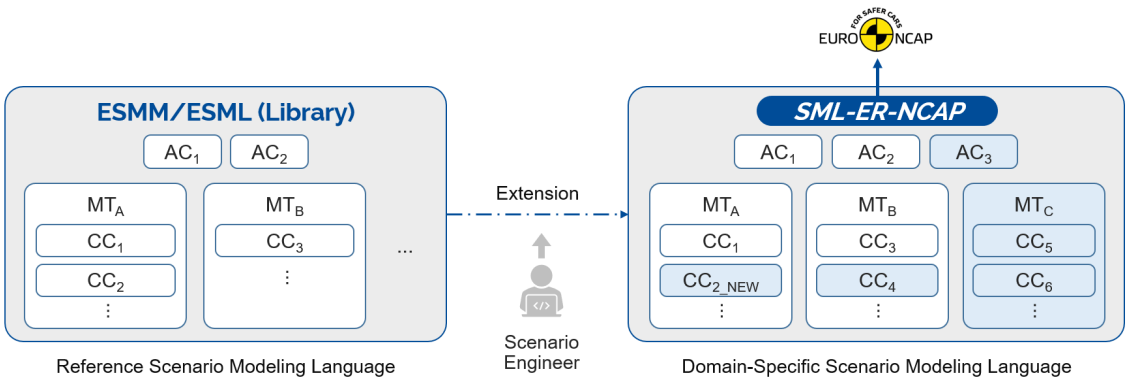

Figure 18: Extension (*Step 2* and *Step 3*) of *ESML* constructs to develop *SML-ER-NCAP*

tions before this model type extension process. After this step, the developer can export this extended library into an *ABL*-formatted file (e.g., *ESML_Euro_NCAP_AEB_C2C.abl*, and another scenario engineer can use the released library as a modeling tool or a library (i.e., a modeling method/language) iteself. The processes of *Step 2* and *Step 3* are summarized in Figure 18.

Step 4. At the last step, a scenario developer (i.e., modeling engineer, modeler) can perform actual modeling activities to specify *Euro-NCAP* world ontology and test scenarios. The modeler can create his/her own models using a graphical user interface, supported by the *ADOxx Framework*. Some actual modeling results are attached as figures: Figure 19 is a WOI model to describe the *Euro NCAP* domain ontology, by specifying domain-specific objects, such as a map, time/space units, domain-specific units (e.g., *acceleration rate unit*), predefined events, environment elements, instances, constants, and variables; Figure 20 shows one of type definition models that specify a type of a *vehicle under test (VUT)*; Figure 21 is a scenario suite model that composes participant objects, constituent scenarios, and relevant conditions and constraints; Figure 22 shows an example scenario model that describes contexts and a path to test a *Car-to-Car Rear Stationary (CCRs)* case.

By following the four steps outlined above, it becomes evident that *ESML* not only supports fundamental scenario modeling activities but can also be flexibly extended to serve as a DS-SML. Traditional scenario description languages focus solely on accurately representing scenario semantics, which limits their ability to handle domain-specific requirements related to additional domain entities and data. *ESML*, on the other hand, overcomes this limitation by distinguishing between *ESML* constructs and WOI constructs at the abstract class level. The *ESML* constructs provide domain-independent support for scenario semantics, while WOI constructs enable domain-specific class definitions for domain-focused descriptions. This distinction allows scenario models to incorporate domain knowledge as an ontology base, making them more intuitive and understandable.

In addition to the *Euro NCAP* test procedure discussed earlier, an additional language extension was performed to validate the applicability of *ESML* in different domains. This extension was based on drone flight mission scenarios from the UAV domain, included in *Category C* of Table 5. The extended language (*SML-BVLOS-FLIGHT-MISSION*) developed through this process encompasses eight major scenarios related to the flight missions of *Beyond Visual Line of Sight (BVLOS)* drones, which operate beyond the pilot's field of view. The extension process for *ESML* followed the same steps as the previous *SML-ER-NCAP*, and the domain terms and data were collected based on the technical reports of *Single European Sky ATM Research (SESAR)* published in 2021[19]. This data was then used to define subclasses and attributes of abstract classes of *ESML* and update the corresponding model

[19]Jacob Blamey et al., "(C12) Safe Drone Flight - Assuring telemetry data integrity in U-Space scenarios," 2021.

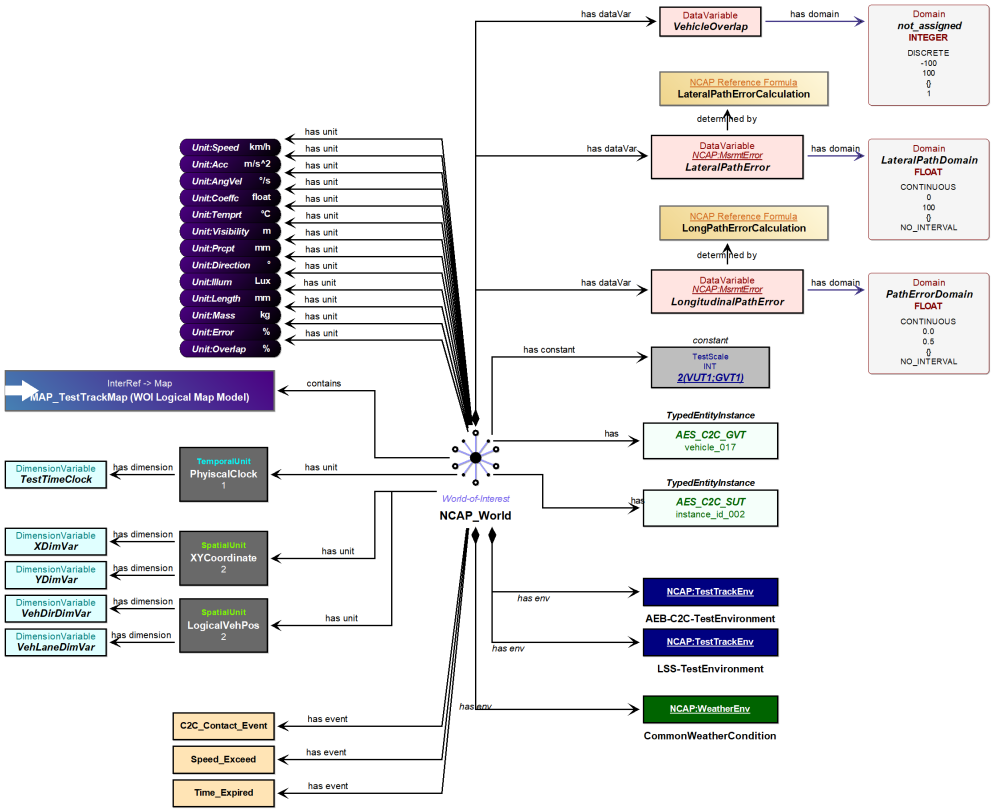

Figure 19: A WOI model developed using *SML-ER-NCAP*

type. Due to space limitations, detailed descriptions of each step and modeling results were omitted, and instead, the derived domain classes were summarized in Table 9. The modeling language library of *SML-BVLOS-FLIGHT-MISSION* was also uploaded to the *ESMM* Github repository[20].

The DS-SMLs, which are extensions of *ESML*, proved to be effective for both domain-specific class definitions and ontology-based scenario development. However, they have certain limitations because *ESML* is originally designed for conceptual modeling that describes black-boxed scenario data. For instance, they primarily allow for the specification of observable actions/events, thus it is challenging to express the detailed technical data/objects (e.g., blockchain, encryption/cryptography, authentication, mobile apps) or domain-specific algorithms (e.g., detection and avoidance algorithms). Another observation is that extending the language for a specific domain often necessitates well-defined domain-specific variables (`WOI_DataVar`) and well-defined units (`WOI_Unit`) to be used in the variables. A clear definition of the values, domains, and units of the variables in the model is particularly crucial to transform them into concrete scenarios that can be utilized as inputs in real-world tests and simulations based on functional/logical scenarios.

[20] *ESMM* Library Repository: https://github.com/AnthonyBaek/esmm

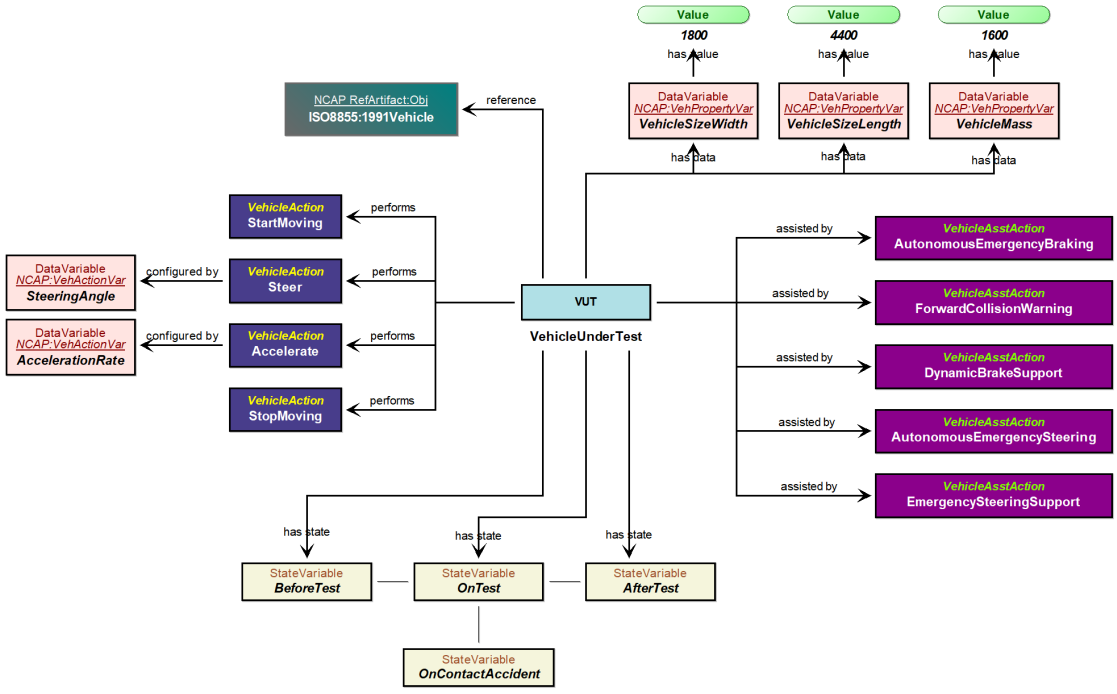

Figure 20: A type definition model (VUT) developed using *SML-ER-NCAP*

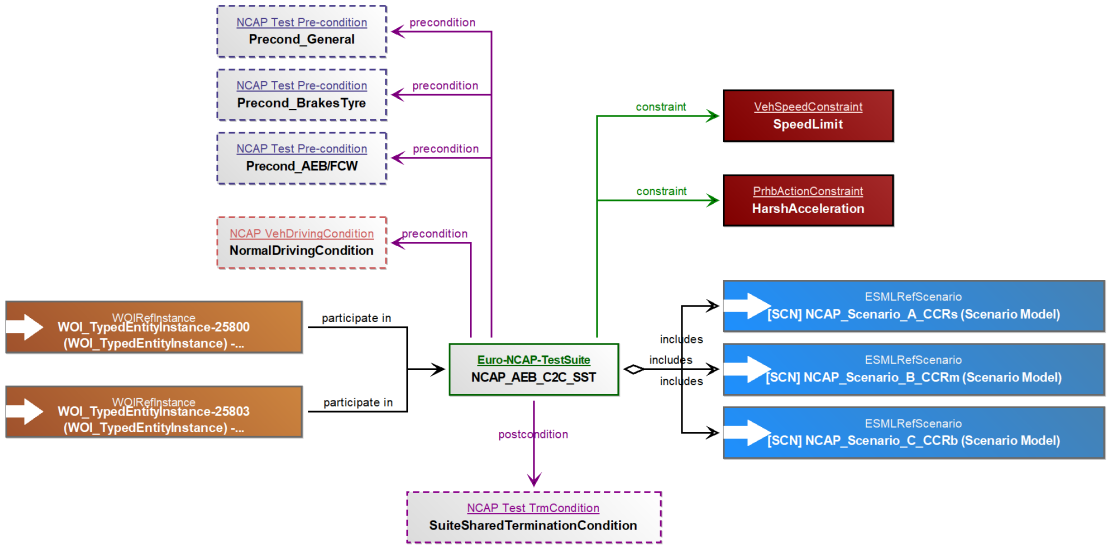

Figure 21: A scenario suite model developed using *SML-ER-NCAP*

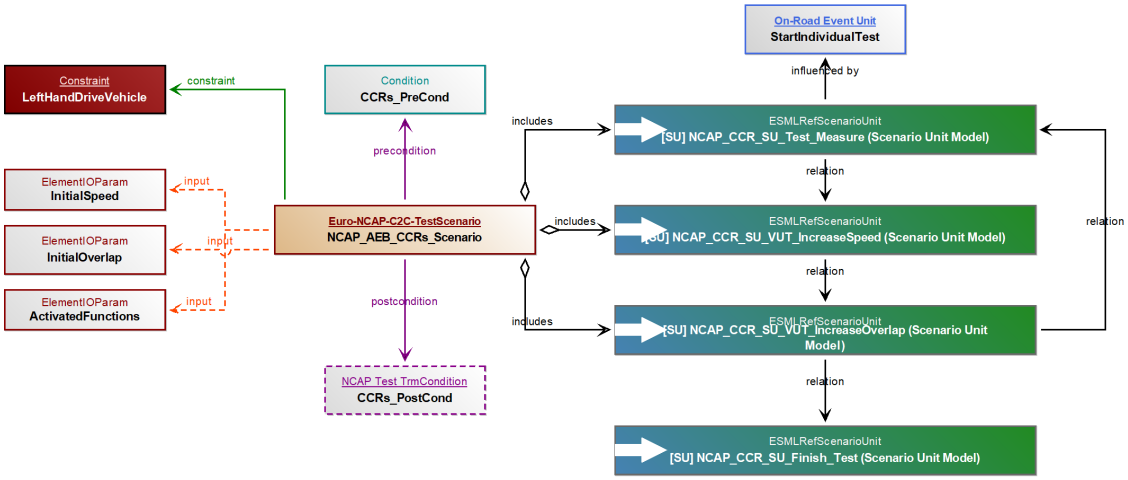

Figure 22: A scenario model developed using *SML-ER-NCAP*

6.6 Summary of Case Study using *ESML*

The *ESMM* modeling tool (*ESMM-Tool*) basically supports scenario modeling activities, which are to be performed by scenario engineers (i.e., modelers). By providing user interfaces, controllers, and graphical notations/visualizations, the engineers can receive practical help from the tool. In addition, the manipulation and import/export of the library enables a method engineer to easily develop a new domain-specific scenario modeling method, aided by the *ADOxx* development environment. These supports, as a result, facilitate the customization of the general-purpose modeling language or method on the open modeling environment, such as *OMiLAB*[21].

As designed in Section 5.3, *ESML* provides basic scenario constructs, which represent essential information to describe scenarios and a WOI. As this case study showed, the *ESML* constructs could play a role as base constructs that scenario engineers can easily understand and extend. In other words, a method engineer can efficiently include domain-specific knowledge by extending the pre-defined classes, and the metamodeling of domain-specific classes can expand overall semantic space of the language. If these processes are not supported at the method-level, the engineer might need to manually dismantle other domain-specific languages and implement a new domain-specific tool from scratch. In summary, *ESML* is a domain-independent language specifically designed to be extensible and adaptable, making it suitable as a reference method for scenario development.

7 Discussion

Content and Construct Validity. *Content validity* refers to the extent to which a survey covers a representative sample of the target domain it aims to measure, while *construct validity* pertains to the legitimacy and accuracy of the survey in measuring what it claims to measure. In this study, content validity was achieved by designing search keywords and queries specifically tailored to the software and systems engineering domain that is highly relevant to scenario usage. However, due to the impracticality of analyzing over 50,000 research results, the search was narrowed down to event-based methods and target engineering activities such as analysis, design, simulation, and testing.

[21] OMiLab Community - Projects - Modeling Tools:
https://www.omilab.org/activities/projects/details/

These activities were selected based on a preliminary investigation that identified the primary scenario-utilizing practices, ensuring the collected data was representative of the overall contents.

To address construct validity, this study designed the research questions and survey process meticulously. However, there are three elements that may impact this validity. Firstly, conducting a comprehensive manual review and in-depth inspection of conceptual data for all 354 all FSPs was impractical, resulting in random selection of 100 FSPs for questionnaire-based manual investigation. Secondly, keyword-based searches were conducted on the textual descriptions of methodologies and scenario development methods, which means data collection was limited to explicitly mentioned information, potentially excluding implicitly included data. Thirdly, a manual inspection was performed during the full-read step, although the use of a well-structured questionnaire and lookup tables based on high-level variables helped minimize missing data and ensure consistency in data collection.

Internal and External Validity. *Internal validity* pertains to the extent to which causal relationships can be inferred based on the measures used, research settings, and overall research design. *External validity*, on the other hand, refers to the generalizability of internally valid results to other cases. In this study, internal validity was addressed by defining characteristics and concepts based on the consolidation and classification of survey data (i.e., *SVs*), avoiding subjective definitions by the authors. The selected papers were reviewed not only from a technical perspective but also considering actual scenario instances. The *SCF*, which abstracts the entire collected dataset, is expected to encompass equivalent or similar results from independent scenario investigations. Therefore, the concepts defined in this study are grounded in the survey data, ensuring the internal validity.

Furthermore, the investigated scenario instances were not solely developed for academic purposes but also for practical industrial use, including actual tests and simulations. As a result, the scenarios developed based on the *CSF* and *ESMM* can be seen as incorporating real-world scenario data. Domain-specific data can be mapped to higher-level abstract concepts in the *SCF* and *ESMM* classes through a generalization step, and if necessary, domain-specific objects can be created. Thus the proposed methods have external validity as they can be extended and applied to scenario development in external real-world cases.

8 Conclusion

This study addresses the lack of a conceptual basis and a reference method for scenario development, which has posed challenges for scenario engineers who lack well-established understanding and need to develop their own scenario methods. To tackle this issue, this sutdy introduces *CSF* and *ESMM* as a solution. Firstly, to establish a comprehensive understanding of scenarios and scenario methods, a semi-systematic literature review was conducted to conceptualize *SVs*, which form the core concepts of the *CSF*. Secondly, building upon the *CSF*, this study developed *ESMM*, which offers a graphical modeling language called the *ESML*. By designing the constructs of *ESMM/ESML* in a domain-independent manner, the proposed method provides scenario engineers with extensible semantics that can be customized for domain-specific requirements. In the future, the proposed *CSF* and *ESMM* will serve as a reference method for (a) method developers seeking to create domain-specific scenario modeling languages suitable for their respective domain, and (b) modeling engineers aiming to specify scenarios that accurately capture the characteristics of their scenarios using the provided scenario language.

References

[1] Alexander, I. On abstraction in scenarios. *Requir. Eng. 6* (01 2002), 252–255.

[2] Bach, J., Otten, S., and Sax, E. Model based scenario specification for development and test of automated driving functions. In *2016 IEEE Intelligent Vehicles Symposium (IV)* (2016), pp. 1149–1155.

[3] Baek, Y.-M., Song, J., Shin, Y.-J., Park, S., and Bae, D.-H. A meta-model for representing system-of-systems ontologies. In *2018 IEEE/ACM 6th International Workshop on Software Engineering for Systems-of-Systems (SESoS)* (2018), pp. 1–7.

[4] Bagschik, G., Menzel, T., and Maurer, M. Ontology based scene creation for the development of automated vehicles.

[5] Bagschik, G., Reschka, A., Stolte, T., and Maurer, M. Identification of potential hazardous events for an unmanned protective vehicle. In *2016 IEEE Intelligent Vehicles Symposium (IV)* (06 2016), pp. 691–697.

[6] Börjeson, L., Höjer, M., Dreborg, K.-H., Ekvall, T., and Finnveden, G. Scenario types and techniques: Towards a user's guide. *Futures 38*, 7 (2006), 723–739.

[7] Chen, W., and Kloul, L. An ontology-based approach to generate the advanced driver assistance use cases of highway traffic. pp. 75–83.

[8] Cimatti, A., Mover, S., and Tonetta, S. Efficient scenario verification for hybrid automata. pp. 317–332.

[9] Cunning, S. J., and Rozenblit, J. Automating test generation for discrete event oriented embedded systems. *Journal of Intelligent and Robotic Systems 41* (2005), 87–112.

[10] Di Leva, A., Sulis, E., Lellis, A., and Amantea, I. *Business Process Analysis and Change Management: The Role of Material Resource Planning and Discrete-Event Simulation.* 01 2020, pp. 211–221.

[11] Faria, J. P., and Paiva, A. C. R. A toolset for conformance testing against uml sequence diagrams based on event-driven colored petri nets. *International Journal on Software Tools for Technology Transfer 18* (2014), 285–304.

[12] Fill, H., and Karagiannis, D. On the conceptualisation of modelling methods using the adoxx meta modelling platform. *Enterp. Model. Inf. Syst. Archit. Int. J. Concept. Model. 8*, 1 (2013), 4–25.

[13] for Standardization of Automation, A., and (ASAM), M. S. Asam openscenario: Version 2.0.0 concepts, 2020.

[14] Fremont, D. J., Dreossi, T., Ghosh, S., Yue, X., Sangiovanni-Vincentelli, A. L., and Seshia, S. A. Scenic: A language for scenario specification and scene generation. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation* (New York, NY, USA, 2019), PLDI 2019, Association for Computing Machinery, p. 63–78.

[15] Godet, M. The art of scenarios and strategic planning: Tools and pitfalls. *Technological Forecasting and Social Change 65*, 1 (2000), 3–22.

[16] Greenyer, J., Gritzner, D., Gutjahr, T., König, F., Glade, N., Marron, A., and Katz, G. Scenariotools–a tool suite for the scenario-based modeling and analysis of reactive systems. *Science of Computer Programming 149* (2017), 15–27.

[17] Greenyer, J., Haase, M., Marhenke, J., and Bellmer, R. Evaluating a formal scenario-based method for the requirements analysis in automotive software engineering. pp. 1002–1005.

[18] Greenyer, J., Haase, M., Marhenke, J., and Bellmer, R. Evaluating a formal scenario-based method for the requirements analysis in automotive software engineering. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (2015), pp. 1002–1005.

[19] Gritzner, D., Greenyer, J., Katz, G., and Marron, A. Scenario-based modeling and synthesis for reactive systems with dynamic system structure in scenariotools.

[20] Gruber, T. R. Toward principles for the design of ontologies used for knowledge sharing? *International Journal of Human-Computer Studies 43*, 5 (1995), 907–928.

[21] Heymans, P., and Dubois, E. Scenario-based techniques for supporting the elaboration and the validation of formal requirements. *Requirements Engineering 3* (1998), 202–218.

[22] Ishida, T., and Yamane, S. Introduction to scenario description language q. pp. 137 – 144.

[23] Jafer, S., Chhaya, B., and Durak, U. Graphical specification of flight scenarios with aviation scenario defintion language (asdl).

[24] Jafer, S., Chhaya, B., Durak, U., and Gerlach, T. Formal scenario definition language for aviation: Aircraft landing case study.

[25] Jafer, S., and Durak, U. Tackling the complexity of simulation scenario development in aviation. In *Proceedings of the Symposium on Modeling and Simulation of Complexity in Intelligent, Adaptive and Autonomous Systems* (San Diego, CA, USA, 2017), MSCIAAS '17, Society for Computer Simulation International.

[26] Karagiannis, D., and Kühn, H. Metamodelling platforms. vol. 2455, p. 182.

[27] Khastgir, S., Brewerton, S., Thomas, J., and Jennings, P. Systems approach to creating test scenarios for automated driving systems. *Reliability Engineering & System Safety 215* (2021), 107610.

[28] Kurakawa, K. A scenario-driven conceptual design information model and its formation. *Research in Engineering Design 15* (09 2004), 122–137.

[29] Mazzega, J., Lipinski, D., Eberle, U., Schittenhelm, H., and Wachenfeld, W. Pegasus method. Tech. rep., 05 2019.

[30] Richard, C. M., Campbell, J. L., and Brown, J. L. Task analysis of intersection driving scenarios: Information processing bottlenecks.

[31] Schütt, B., Braun, T., Otten, S., and Sax, E. Sceml: A graphical modeling framework for scenario-based testing of autonomous vehicles. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems* (New York, NY, USA, 2020), MODELS '20, Association for Computing Machinery, p. 114–120.

[32] Siegfried, R., Laux, A., Rother, M., Steinkamp, D., Herrmann, G., Lüthi, J., and Hahn, M. Scenarios in military (distributed) simulation environments.

[33] Guideline on scenario development for (distributed) simulation environments. Sto technical report, Brussels: NATO, 2014.

[34] Standard for military scenario definition language (msdl). Standard, Simulation Interoperability Standards Organization, Inc., 2008.

[35] Tomizawa, H. Automated scenario generation system in a simulation.

[36] Tsai, W.-T., Bai, X., Paul, R., and Yu, L. Scenario-based functional regression testing. In *25th Annual International Computer Software and Applications Conference. COMPSAC 2001* (2001), IEEE, pp. 496–501.

[37] Tsai, W.-T., Fan, C., Cao, Z., Xiao, B., Huang, H., Song, W., Liu, X., Wei, X., Paul, R., Chen, Y., and Zhang, D. A scenario-based service-oriented rapid multi-agent distributed modeling and simulation framework for sos/soa and its applications.

[38] Updegrove, J., and Durak, U. Schema-based ontological representations of a domain-specific scenario modeling language. *Journal of Simulation Engineering 1* (01 2018).

[39] Wang, J., Jiang, L., and Cai, H. Scenario-based method for business process analysis and improvement in soa. In *2014 IEEE 11th International Conference on e-Business Engineering* (2014), pp. 19–25.

[40] Wen, M., Park, J., Sung, Y., Park, Y. W., and Cho, K. Virtual scenario simulation and modeling framework in autonomous driving simulators. *Electronics 10*, 6 (2021).

[41] Wiecher, C., Fischbach, J., Greenyer, J., Vogelsang, A., Wolff, C., and Dumitrescu, R. Integrated and iterative requirements analysis and test specification: A case study at kostal.

[42] Wiecher, C., Japs, S., Kaiser, L., Greenyer, J., Dumitrescu, R., and Wolff, C. Scenarios in the loop: integrated requirements analysis and automotive system validation. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings* (2020), pp. 1–10.

[43] Wittman, R. L. Defining a standard: The military scenario definition language version 1.0 standard. In *Proceedings of the 2009 Spring Simulation Multiconference* (San Diego, CA, USA, 2009), SpringSim '09, Society for Computer Simulation International.

[44] Zhang, T., Wiegley, J., Giannakopoulos, T., Eakman, G., Pit-Claudel, C., Lee, I., and Sokolsky, O. Correct-by-construction implementation of runtime monitors using stepwise refinement. In *Dependable Software Engineering. Theories, Tools, and Applications* (Cham, 2018), X. Feng, M. Müller-Olm, and Z. Yang, Eds., Springer International Publishing, pp. 31–49.

[45] Zhang, X., Khastgir, S., and Jennings, P. Scenario description language for automated driving systems: A two level abstraction approach. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)* (2020), pp. 973–980.

A Semantic Mapping of ESML Constructs with Real-world Scenario Instances

Table 10: Concept Mapping - Scenario Constructs: Cases 1-3

Category	Abstract Classes	Concrete Classes	*Case 01*	*Case 02*	*Case 03*
Scenario Construct	_ESML _Specification_	ViewpointSpec	*ScenarioObject*		
		HypothesisSpec	*TargetSpeed Comparison (TestAssertionCriteria: EqualTo, LessThan)*		
		Execution Method	*MaxExecutionCount, SimulationTimeCondition*		
	_ESML_Element_	ScenarioSuite	*StoryboardElement*		
		Scenario	*(Concurrent) Story*		
		ScenarioUnit	*(Parallel) Act (Behavior), ManeuverGroup (ManeuverSequence), Maneuver*		
	_ESML _ElementIO_	ElementParam (Scenario-level)	*ScenarioObjectParam*		
		ElementParam (ScenarioUnit (Event)-level)	*(ParameterDeclaration) Value, Rule, Ref*		
		ElementInput Model	*RoadNetwork (LogicFile(Town), SceneGraphFile), Car Model (vehicle.tesla.model3, vehicle.lincoln.mkz_2017), UserDefinedAction*		*RoadNetwork (LogicFile(Town), SceneGraphFile), CarModel (vehicle.volkswagen.t2), Pedestrian Model (walker.pedestrian.0001), UserDefinedAction*
		ElementOutput			
	_ESML _ElementContext_	ElementContext Condition (Scenario Level)	*RoadCondition, StartTrigger (OverallStartCondition), StopTrigger (EndCondition)*		
		ElementContext Condition (ScenarioUnit (Event) Level)	*StoryboardElement StateCondition, ConditionGroup, EntityCondition, ValueCondition, ParameterCondition, TriggeringEntity (Rule), TraveledDistanceCondition, RelativeDistanceCondition*		*StoryboardElement StateCondition, ConditionGroup, EntityCondition, ValueCondition, ParameterCondition, TriggeringEntity (Rule), TraveledDistanceCondition, RelativeDistanceCondition, ReachPositionCondition*
		ElementContext Constraint	*EventConstraint (KeepsVelocity)*		
	_ESML _Statement_	StmtOccurrence	*Event (GlobalAction, EnvironmentAction, PrivateAction)*		
		StmtBehavior	*Action and ActionDynamics, ElementState, VehicleAction/Maneuver (Same as _WOI_Action_)*		
		StmtTime	*StartTime, Delay*		
		StmtLocation	*Position, Distance*		
		StmtProbability	*ParameterProbabilityDistributions (e.g., Uniform, Normal)*		
		StmtAssertion	*Rule (e.g., LessThen)*		
		StmtAssumption			
		StmtParameter	*ParameterDeclaration, ActionParameter*		
	_ESML _ExtArtifact_	ModelData Artifact	*VehiclesCatalog, ControllersCatalog, ManeuversCatalog, PedestriansCatalog, RoutesCatalog*		
	_ESML_InterRef_		*ScenarioObject (Actors), ElementRef (EntityRef), ParameterRef*		

Table 11: Concept Mapping - WOI Constructs: Cases 1-3

Category	Abstract Classes	Concrete Classes	*Case 01*	*Case 02*	*Case 03*
WOI Construct	_WOI _ContainerClass_	WOI	*RoadNetwork (LogicFile)*		
		WOI Geographical (Map)	*RoadScene (External)*		
		SystemGroup	*Catalogs (VehiclesCatalog,ControllersCatalog) from ModelDataArtifact*		
		Infrastructure	*RoadNetwork*		
		Environment	*TimeOfDay, Weather, RoadCondition*		
	_WOI_Type_	EntityType	*Actor, VehicleCategory, Vehicle (with VehicleCapability (Acceleration/Deceleration/Speed))*		*VehicleCategory Vehicle (with VehicleCapability (Acceleration/ Deceleration/ Speed)), Pedestrian*
		DataType	*SunType (Elevation, Azimuth, Intensity), Fog (VisualRange), PrecipitationType (Intensity, PrecipitationType)*		
	_WOI_Instance_	TypedEntity Instance	*EgoVehicle, SimulationVehicle*		*EgoVehicle, Pedestrian*
		EnvFactor ObjectInstance	*Weather, RoadCondition*		
	_WOI_Event_	PredefEvent	*Event (Adversary Accelerates, Adversary ChangesLane)*	*Event (LeadingKeeps Velocity, LeadingWaits)*	*Event (PedestrianStops AndWait, AfterPedestrian Walks)*
	_WOI_Variable_	DataVar	*VehicleCategory, VehicleLocalProperty, PerformanceVariable, (Deceleration/Acceleration)*		
		StateVar	*ControlValueStateActive (Throttle,Brake,Clutch,ParkingBrake,SteeringWheel,Gear)*		
		DimVar	*World Position (X, Y, Z, H), DynamicsDimension*		
	_WOI_Set_	StateSet			
		EventSet			
		VariableSet	*MiscObjectCatalog*		
		InstanceSet	*Actors (ActorSet)*		
		ActionSet	*EnvironmentAction, ManeuverCatalog, ManeuverGroup*		
	_WOI_Domain_	VarDomain	*DynamicsShape*		
	_WOI_Unit_	TmpUnit	*DateTime*		
		SptUnit	*WorldPosition, RoadPosition, Lane*		
		OtherUnit	*Speed, Acceleration, Mass, Angle, LightIntensity*		
	_WOI_Action_	EntityAction	*ControllerAction, Maneuver (LongitudinalAction, LateralAction), TeleportAction*	*ControllerAction, Maneuver (LongitudinalAction), TeleportAction*	
		CommAction/ Interaction	*RelativePosition*		*Reach*

Table 12: Concept Mapping - Scenario Constructs: Cases 4-5

Category	Abstract Classes	Concrete Classes	*Case 04*	*Case 05*
Scenario Construct	_ESML _Specification_	ViewpointSpec	*ADS-Viewpoint*	
		HypothesisSpec	*AutomatedVehiclePolicy (ExpectedHazard, UnexpectedEvents, CriticalDrivingManeuvers, EvaluationCriteria), FailureMode*	
	_ESML_Method_	Specification Method		
		Execution Method	*TestEquipment, TestFacility, ExecutionOfProcedure, Metrics*	
	_ESML_Element_	ScenarioSuite	*ScenarioTotal*	
		Scenario	*BaselineScenario*	*Scenario*
		ScenarioUnit	*Maneuver (with param, e.g., SV speed, location, POVs)*	
	_ESML _ElementIO_	ElementParam (Scenario-level)	*GPS Location, StartingPosition, EndPosition, ExpectedSDVParams, OperationalParams (Visibility, Sensing, Delay)*	*RoadwayTypes, RoadwaySurfaces, RoadwayConfiguration (LaneWidth)*
		ElementParam (ScenarioUnit (Event)-level)	*Latency, ErrorRates*	*ActorParam (Speed)*
		ElementInput Model	*Operational Design Domain (ODD), Map/Template*	
		ElementOutput	*BehaviorCompetencyComparison, MeasureOfObservation(Test)*	
	_ESML _ElementContext_	ElementContext Condition (Scenario Level)	*InitialCondition (Position,Speed,etc.), BoundaryCondition/FailureCondition, AmbientConditions, TrafficCondition, Weather-induced RoadwayCondition*	
		ElementContext Condition (ScenarioUnit (Event) Level)	*InstantTrafficCondition, DrivingCondition*	
		ElementContext Constraint	*OperationalConstraints (SpeedLimit, TrafficCondition), LimitOfConditionBehaviorDomain, ConstrainedResource (+DomainConstraints, provided by ODD)*	
	_ESML _Statement_	StmtOccurrence	*Event, Interaction*	*Failure (Expected)Event (Crash, Collision), Interaction, Detection*
		StmtBehavior	*Maneuver (with ScenarioParams), DynamicResponse*	
		StmtTime	*Timeout*	*Timezone, Period (StartToEndTime)*
		StmtLocation	*Distance, Position*	
		StmtProbability	*Probability*	
		StmtAssertion		
		StmtAssumption		
		StmtParameter	*TestParameters (TestingVariables,TestingCondition), OperationalParams*	
	_ESML _ExtArtifact_	ModelData Artifact	*Map/Template*	*Digital Infrastructure (GPS, Maps, WeatherData, InfrastructureData)*
	_ESML_InterRef_		*ODDElementRef*	

Table 13: Concept Mapping - WOI Constructs: Cases 4-5

Category	Abstract Classes	Concrete Classes	*Case 04*	*Case 05*
WOI Construct	_WOI _ContainerClass_	WOI	*Intersection (N Scottsdale Rd E Thomas Rd)*	*Defined by ODD Elements - Physical Infrastructure - Operational Constraints - Environmental Conditions - Connectivity - Zones*
		WOI Geographical (Map)	*RoadMap*	
		SystemGroup	*RoadUserSystemGroup*	
		Infrastructure	*RoadInfrastructure*	*Physical Infrastructure, Signage (Signs, TrafficSignals, Crosswalks, etc.),InfrastructureSensors*
		Environment	*Weather*	
	_WOI_Type_	EntityType	*RoadwayUsers (VehicleTypes, StoppedVehicles, MovingVehicles, Pedestrians, Cyclists), Non-roadwayUser (Obstacles/Objects)*	
		DataType	*WeatherData, InfrastructureData, MapData*	
	_WOI_Instance_	TypedEntity Instance	*All ODD Instances*	
		EnvFactor ObjectInstance	*ODD Environmental Conditions*	
	_WOI_Event_	PredefEvent		
	_WOI_Variable_	DataVar	*For all ODD Element Properties*	
		StateVar	*TrafficSignState*	
		DimVar	*Map Dimensions (3D)*	
	_WOI_Set_	StateSet	*For all ODD Element States*	
		EventSet	*VehicleEventInteractions, PedestrianEventInteractions, PedalcyclistEventInteractions, AnimalEventInteractions, DebrisEventInteractions, DynamicObjectEvents, TrafficSignsEvents, TrafficSignalsEvents, VehicleSignalEvents, EmergencyVehicleEvents, SchoolBusesEvents*	
		VariableSet	*ODD Predefined Variables*	
		InstanceSet	*RoadwayUserSet*	
		ActionSet	*ManeuverActionSet*	
	_WOI_Domain_	VarDomain	*RangeOfOperationalParams*	
	_WOI_Unit_	TmpUnit	*Seconds*	
		SptUnit	*Meters, Miles, Zone, Region/State*	
		OtherUnit	*ODD-provided Speed, Temperature, Illumination*	
	_WOI_Action_	EntityAction	*VehicleAction (BehaviorCapability), TrafficSignAction*	
		CommAction/ Interaction	*ODD-Defined Connectivity (V2V, Infra)*	

Table 14: Concept Mapping - Scenario Constructs: Cases 6-8

Category	Abstract Classes	Concrete Classes	*Case 06*	*Case 07*	*Case 08*
Scenario Construct	_ESML _Specification_	ViewpointSpec	*Birdeye-Viewpoint*		
		HypothesisSpec	*RelativeSpeed (Equal-to), SafetyHazard*		
	_ESML_Method_	Specification Method			*Abstraction Level*
		Execution Method			
	_ESML_Element_	ScenarioSuite		*Case*	
		Scenario	*Scene Graph*	*Scenario*	*ScenarioDescription (Lv1,2)*
		ScenarioUnit	*Scene*	*Situation (models Scene)*	*DO (Phases)*
	_ESML _ElementIO_	ElementParam (Scenario-level)	*RoadType, SightCondition, Weather/ SightCondition*	*Number-of-Vehicles, InitialStateVariables*	*WorldType, RoadType, CarriagewayHazard, Ego (with EgoPosition) Vehicle, VehiclePosition*
		ElementParam (ScenarioUnit (Event)-level)	*ActionParam (Speed, Distance)*	*Duration, ActivityParams*	*ActionParam, RelativeActionParam, LocationParam*
		ElementInput Model	*RoadLayout*	*VehicleModel (PEPA), Infrastructure (PEPA), Visiblity (PEPA)*	*RoadNetworkModel*
		ElementOutput			
	_ESML _ElementContext_	ElementContext Condition (Scenario Level)	*TriggeringCondition*	*InitialState, WeatherCondition*	*RoadSurface Condition, WeatherCondition, LightingCondition, TerminationCondition*
		ElementContext Condition (ScenarioUnit (Event) Level)		*ActionCondition (Vehicle/Driver)*	*INITIAL (EgoState), WHEN*
		ElementContext Constraint		*SpeedLimit*	*SpeedLimit*
	_ESML _Statement_	StmtOccurrence	*Event*	*Situation*	*DO-statements*
		StmtBehavior	*VehicleAction (Same as _WOI_Action_)*	*VehicleAction (Same as _WOI_Action_), Activity*	*Drive_Towards, Drive_Away, LaneChgLeft_CutIn, Stop_Away, Going_Ahead*
		StmtTime	*ActionLength*	*Length, ScenarioLength, InstanceTime (e.g., Daytime)*	
		StmtLocation	*Distance*	*Direction, Position*	*AbsolutePosition*
		StmtProbability		*Probability, Steady-state Probability, Distribution*	
		StmtAssertion			
		StmtAssumption			
		StmtParameter	*SpeedParameter (min,max), DistanceParameter*	*EntityFunction Param, ActivityParam, WeatherParameter*	*DynamicElement Param, SceneryElement Param, EnvironmentElement Param*
	_ESML _ExtArtifact_	ModelData Artifact	*SOTIF (SafetyIntended Functionality)*		*RoadNetworkModel, Enums*
	_ESML_InterRef_		*ODDElementRef*	*(Scene)Ontology Reference*	*ODDElementRef*

Table 15: Concept Mapping - WOI Constructs: Cases 6-8

Category	Abstract Classes	Concrete Classes	*Case 06*	*Case 07*	*Case 08*
WOI Construct	_WOI _ContainerClass_	WOI	*Road*	*Highway (RoadWay, RoadPart, Symbol)*	*Junction, Road*
		WOI Geographical (Map)	*RoadLayout (RoadStructure)*	*Carriageway (ThroughLane), EntranceLane with Shoulder (Soft/Hard)*	*RoadLaneConnection, Junction, Zone, Length, Lane (Number, Width)*
		SystemGroup		*Traffic Jam Chauffeur (TJC)*	
		Infrastructure		*RoadPart (Toll, Tunnel, Bridge), Symbol, Line, RoadwaySymbol*	*JunctionNetwork, Map-Roads, Connection Control, RoadSign, TrafficLight, RoadsideFeature*
		Environment	*Weather, SightCondition*	*Weather (Lighting (DayLight), Temperature, Humidity, Pressure, Wind, WindDirection, Fog, Haze)*	*EnvironmentElement (Environment with Wind, Clouds, Precipitation, Visibility, Time, Illumination, Connectivity)*
	_WOI_Type_	EntityType	*EgoVehicle, EnvVehicle*	*Autonomous Vehicle (Ego), MobileElement, StaticElement*	*Agent/Object, VehicleUnderTest (VUT), VehicleTarget (GVT)*
		DataType		*EntityPropertyType (e.g., Lane, Symbol)*	
	_WOI_Instance_	TypedEntity Instance	*EgoVehicle, EnvVehicle*	*EgoVehicle*	*Ego (Vehicle), On-Road Vehicle (EnvVehicle), TrafficLight*
		EnvFactor ObjectInstance		*Instances of Weather*	*Instances of EnvironmentElement*
	_WOI_Event_	PredefEvent	*VehicleMerging Event, VehicleStrong DecelerationEvent*	*RoadWorkEvent*	*CarriagewayHazard*
	_WOI_Variable_	DataVar		*VehicleLocalProperty, RoadVariable, JunctionVariable*	
		StateVar	*State*	*TrafficLight: TrafficSign*	*ManeuverAngle, Car-to-car rear stationary (CCRs)*
		DimVar		*Length, Width*	*Dimensions (Width, Depth)*
	_WOI_Set_	StateSet		*VehicleActionState, VehicleLightState*	
		EventSet			
		VariableSet			
		InstanceSet		*Highway ComponentSet (e.g., RoadPartSet, HighwaySet, LaneSet, etc.)*	
		ActionSet		*ActionSet*	*JunctionControl*
	_WOI_Domain_	VarDomain	*MinMax*		*Enumeration*
	_WOI_Unit_	TmpUnit	*Milli-Seconds, Seconds*		
		SptUnit	*Meters*	*Lane, Section (AccelerationSection, Taper)*	*Zone, AbsolutePosition*
		OtherUnit	*Speed*	*Speed*	*AbsoluteSpeed*
	_WOI_Action_	EntityAction	*VehicleAction (Drive, Reach, Continue, Merge, Decelerate)*	*VehicleManeuver/ Action (Run, Acceleration, Deceleration, GoLeftLane, GoRightLane, Wait)*	*VehicleManeuver (AbsoluteManeuvers (Types) (Drive, LaneChangeRight,etc.), RelativeManeuvers (Cut-in)), Emergency Braking*
		CommAction/ Interaction		*TurnOn DirectionLight, RadarDetection*	*Collide*