Internet of Things 36 (2026) 101851

Contents lists available at ScienceDirect

Internet of Things

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

Correct by design, complete by iteration: A graph-based framework for automated security assessment

Felice Moretta [a,*], Umberto Barbato [a], Massimiliano Rak [b], Daniele Granata [c]

[a] Department of Engineering, University of Campania Luigi Vanvitelli, via Roma 29, Aversa, CE, 81031, Italy
[b] Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione (DIETI), University of Naples Federico II, via Claudio 21, Naples, NA, 80125, Italy
[c] Dipartimento di Scienze Economiche, Giuridiche, Informatiche e Motorie (DiSEGIM), University of Naples Parthenope, Isola C4 Centro Direzionale, Naples, NA, 80143, Italy

ARTICLE INFO

Keywords:
System Modeling
Threat Modeling
Penetration Testing
Security
IoT
Internet of Things
CAPEC
Security Assessment

ABSTRACT

Modern IT infrastructures, especially IoT and cyber-physical systems, require systematic and repeatable security assessment methods. A persistent challenge concerns the correctness and completeness of the system model underlying such analyses, which directly affect the quality of threat modeling and penetration test planning. Existing model-based approaches support these activities, but rarely ensure that the adopted model is both structurally valid and representative of the real system. This paper addresses this gap by introducing a formal modeling framework and extending a security assessment methodology (ESSecA) with a cyclical refinement process. At its core lies the Multi-purpose Application Composition Model (MACM), a property-graph formalism equipped with a schema and a set of syntactic and semantic constraints that define the space of admissible models. These constraints are automatically verified through formal checks (e.g., Cypher queries and Neo4j triggers), enabling automated verification of model correctness throughout the assessment lifecycle. As a result, any model accepted by the framework is guaranteed to comply with the rules of the modeling system. The cyclical refinement process complements this by addressing model completeness. Penetration testing results are iteratively reintegrated into the model, enriching it with newly discovered elements and interactions. This produces progressively more accurate system representations, which in turn yield more comprehensive threat models and increasingly precise penetration test plans, effectively mitigating grey-box limitations. The contribution is demonstrated through two case studies: the eWeLink IoT ecosystem, illustrating MACM's modeling and validation capabilities, and the JetRacer autonomous vehicle platform, showcasing the full iterative methodology. Overall, the proposed approach combines a formal modeling system with a cyclic refinement process that exploits such formal guarantees to progressively enhance model completeness, ultimately strengthening threat modeling and penetration test planning.

1. Introduction

The growing implementation of the Internet of Things (IoT) has intensified the need for efficient security assessment mechanisms. Despite its widespread promise, the Internet of Things often lacks foundational security considerations [17]: many devices are de-

* Corresponding author.
E-mail addresses: felice.moretta@unicampania.it (F. Moretta), umberto.barbato@unicampania.it (U. Barbato), massimiliano.rak@unina.it (M. Rak), daniele.granata@uniparthenope.it (D. Granata).

https://doi.org/10.1016/j.iot.2025.101851
Received 13 September 2025; Received in revised form 27 November 2025; Accepted 10 December 2025
Available online 14 December 2025

ployed without a clear security model or embedded security-by-design principles, leaving them vulnerable to a wide range of cyber threats. A comprehensive survey of IoT security requirements, vulnerabilities, and solutions highlights persistent issues such as weak authentication, insufficient encryption, and poor privacy safeguards across device layers [18]. Moreover, as IoT systems increasingly integrate into critical infrastructures, the consequences of security breaches become more severe, emphasizing the need for systematic security assessment methodologies. Given the vast heterogeneity of devices and operational contexts, IoT systems require many different security assessment techniques, in order to assess their security posture, evaluate risks, and support automated penetration testing workflows. Moreover, such techniques should be applied as soon as possible in the system development workflow (the *shift left paradigm of security*), applying as many approaches as possible that address security in the design of the system (*security-by-design*).

Due to such requirements, existing solutions (explored in detail in Section 2), adopt a model-based approach: systems are described through a higher-level model (commonly a graph) that enables threat modeling and risk analysis processes; in a few cases, such techniques can support security and penetration testing [19]. Current approaches tackle a range of challenges, such as automating processes, enabling quantitative risk evaluation, and ensuring methodological simplicity. However, based on our experience, few existing techniques and tools in the security domain explicitly address the issue of underlying model quality. Ensuring the correctness of the adopted model and, consequently, of the resulting security assessment remains largely the responsibility of the experts involved.

In this paper, we address this open challenge by introducing two complementary and mutually reinforcing contributions. First, we define a formally grounded modeling system, called *Multi-purpose Application Composition Model* (*MACM*), capable of representing complex systems through precise syntactic and semantic rules. This formalization enables automated structural checks, enforces compliance with modeling constraints, and provides guarantees on correctness within clearly specified boundaries. The resulting framework allows practitioners to build models that are verifiable by construction, reducing ambiguities and limiting the reliance on expert-driven, ad-hoc interpretations.

Second, we leverage this formal modeling foundation to extend an existing model-based security assessment methodology, *Expert System for Security Assessment* (*ESSecA*), a framework that supports threat modeling, risk analysis, and penetration test planning, by integrating a cyclic refinement workflow. This extension systematically exploits feedback from test execution results to iteratively enrich the system model. By incorporating this iterative mechanism[1], the methodology becomes capable of progressively increasing the model's adherence to the System under Analysis (SuA), thereby improving the accuracy and reliability of the resulting threat models and penetration test plans.

These two contributions jointly provide both a rigorous modeling substrate and a methodology that actively exploits it. On one side, the formalization ensures that the models adhere to the rules of the modeling system; on the other, the iterative refinement process makes it possible to uncover and correct incomplete or missing elements throughout the assessment lifecycle. Together, they enable a security assessment workflow that is both verifiable and progressively more comprehensive.

As a summary, this paper aims at improving the state of the art according to the following results:

1. A formalization of the graph-based model through the Multi-purpose Application Composition Model (MACM), supporting semantic and syntactic validation in order to grant model correctness, under the defined assumptions;
2. Adoption of an iterative approach to grant progressively model completeness and validity, under the defined assumptions;
3. Implementation of tools able to support the model correctness verification and the iterative approach into an existing framework (ESSecA);
4. Demonstration and evaluation through two IoT case studies.

The results of penetration testing are used to update the system model with each round, allowing its progressive refinement. This enrichment improves Threat Modeling and Penetration Test Planning in later iterations.

The remainder of the paper is structured as follows: Section 2 summarizes the state of the art; Section 3 clearly identifies the gaps in literature and presents the contribution of this work; Section 4 details the MACM formalization; Section 5 describes how a MACM can be stored, manipulated, and explains how constraints can be verified using Neo4j; Section 6 reports the ESSecA methodology and its extension to the cyclical approach; Section 7 presents the application of the cyclical approach to the JetRacer case study; and Section 8 concludes with a discussion on the presented approach and its limitations, proposing future improvements and research directions.

2. State of the art and motivations

In the field of security assessment, a wide variety of modeling approaches have been proposed, ranging from lightweight methods aimed at qualitative threat identification to highly formalized frameworks supporting automated analysis. Over the years, methodologies such as *UMLsec*, *SysML-Sec*, *AADL Security Annex*, *MAL/securiCAD*, *SecureUML*, and *ADVISE* have emerged as representative examples of how models can serve as the backbone for reasoning about system security. Although they differ in scope, abstraction level, and degree of automation, they share the principle of embedding security considerations directly into the architecture or design models of the system.

[1] The cyclical approach was first introduced in [20], and this work expands upon that contribution.

Table 1
Comparative analysis of system modeling approaches for security assessment.

Approach	Formal Model	Automation	Machine Readable	Full Cross-Layer	Correctness Verification
MACM	✓✓	✓✓	✓	✓	✓✓
DFD	✗	✗	•	✗	✗
UMLsec	✓	✓	✓	✗	–
SecureUML	✓	✓	✓	✗	–
SysML-Sec	✓	✓	✓	✓	•
AADL + Security Annex	✓	✓	✓	✓	•
ArchiMate + Overlay	✗	✗	✓	✗	✗
MAL / securiCAD	✓	✓	✓	✗	•
ADVISE	✓	✓	✓	✗	•

Legend: ✓✓: Introduced in this article; ✓: Fully supported; •: Partially supported; –: Almost absent; ✗: Not supported;

To systematically identify and compare these approaches, we conducted a literature review. The search was performed in Scopus, chosen because it typically returns the largest number of results across the main academic databases for queries of this kind, thus maximizing coverage in the initial retrieval phase. The query used is reported below; it returned a total of 1344 papers. From these, 54 studies were retained after an initial screening phase, which were further narrowed down to the 13 approaches presented in Table 1.

```
TITLE-ABS-KEY (("model-based" OR "model based" OR "model driven" OR "model-
    driven" OR "security modeling" OR "architectural modeling" OR "system
    model*" OR "domain specific language*") AND ("security assessment" OR "
    security evaluation" OR "security analysis" OR "threat modeling" OR "
    secure system*" OR "security framework" OR "security engineering")) AND
    (LIMIT-TO (LANGUAGE, "English")) AND (PUBYEAR > 2000) AND SUBJAREA(COMP)
```

We included generalizable, model-driven security assessment methodologies that integrate security constraints into architectural models for automated or semi-automated analyses (e.g., threat generation, attack simulation). We excluded studies lacking a core architectural component, those with non-security contexts, purely statistical models, or overly domain-specific applications.

Lightweight, manual frameworks such as STRIDE/DFD [1] and ArchiMate overlays [2,3] support the qualitative identification of threats and risks, but lack explicit mechanisms to formally verify model correctness. In these cases, syntactic and semantic validation is not automated and relies almost entirely on user expertise; only trivial tool-supported checks, such as diagram integrity, are sometimes available.

In contrast, software-oriented formal models like UMLsec [4] and SecureUML [5] embed security requirements into design elements using formal stereotypes and constraint languages (e.g., OCL), which enable partial automation of local syntactic checks. Automated tools can detect violations of predefined rules, yielding a level of assurance around model structure and simple properties. However, comprehensive correctness verification-such as ensuring consistent propagation of security requirements across different components, or complete semantic alignment between architecture and intended security goals is still missing and must be performed manually.

System-level and embedded frameworks, such as SysML-Sec [6] and AADL Security Annex [7,8], increase the formalization by allowing model transformation into formal semantics for tools like model checkers. These tools partially automate the validation of structural and behavioral properties (e.g., communication path consistency, architectural soundness, and some threat scenarios). Semantic correctness, especially regarding cross-layer alignment and satisfaction of high-level security objectives, remains only partially addressed and requires manual interpretation or refinement of results.

Attacker-centric and simulation-based approaches, including MAL/securiCAD [9–11] and ADVISE [12,13], introduce domain-specific meta-models to simulate attack scenarios and quantify risk. In these tools, correctness verification is restricted to checking the internal coherence and well-formedness of the simulation model (e.g., type consistency, structure of attack graphs), but does not extend to semantic validation or assurance that all system properties are faithfully represented. The results thus strongly depend on the expressiveness and accuracy of the modeling language adopted.

Compared to MAL/securiCAD and ADVISE, MACM integrates automated syntactic and semantic validation (via graph constraints and database queries), ensuring that each representation is both correct and progressively made more complete and valid with respect to the real system under analysis. It is important to note that the analyzed techniques do not fully overlap; rather, they share some conceptual foundations while differing in modeling granularity, automation level, and correctness verification mechanisms.

3. Problem statement

The current state of the art exposes a structural limitation in model-based security assessment: the lack of verification mechanisms capable of ensuring the quality of the models on which these methodologies rely. Despite the widespread adoption of model-driven tools (e.g., Microsoft Threat Modeling Tool), they offer no means to evaluate the reliability of the resulting models. This absence of verification mechanisms constitutes a methodological gap with significant implications, as such checks are necessary prerequisites for reliable automated analysis. Without tools capable of assessing the intrinsic quality of models, current approaches operate as black

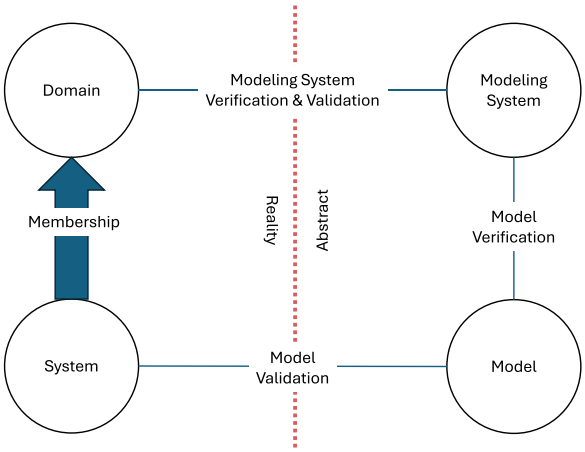

Fig. 1. Extended Cycle Modeling.

boxes, accepting models that may be incomplete, incoherent, or invalid, and nonetheless producing outputs that may be partially correct or, in the worst case, misleading or harmful.

The literature on modeling and simulation provides well-established terminology and methodological structures to ensure the credibility of a *single model*, as defined in works such as [21,22]. These approaches focus on guaranteeing the correctness and validity of a model as it progresses from conceptual model to computational model. However, such methodologies implicitly assume that the underlying modeling formalism is itself correct, expressive, and adequate for the domain. This assumption does not hold in security assessment. In this subject, models do not originate from validated physical laws but from domain-specific modeling techniques whose expressiveness and constraints determine what can or cannot be represented. Consequently, the modeling technique itself becomes a potential point of failure.

To address this issue, we extend the classical simulation-verification cycle, by explicitly introducing both the domain to which the real system belongs and the modeling technique (or meta-model/schema), which defines the set of admissible models (illustrated in Fig. 1). This extended structure provides the foundation for identifying the key properties that govern both the model and the modeling system, enabling verification and validation processes that go beyond a single instantiated model.

At the model level, the central concern is whether a given model correctly and accurately represents a real system within the intended domain. At the modeling-system level, the question shifts to whether the modeling technique (or meta-model) is capable of generating only meaningful, realizable, and sufficiently expressive models. These two layers give rise to a set of foundational properties that determine the overall reliability of model-based security assessment.

- **Model correctness**: the model describes at least one realizable system in the domain.
- **Model validity**: all represented features are true in the real system.
- **Model completeness**: the model contains all the relevant features of the real system.
- **Modeling-system correctness**: the modeling technique guarantees that every generated model is correct.
- **Modeling-system completeness**: the modeling technique is expressive enough to represent all relevant concepts.

Building on the above points, we identify three key fundamental gaps in current model-based security assessment: the absence of guarantees regarding the correctness of the modeling technique, the lack of mechanisms to validate and complete the instantiated model and, it can be questioned whether any modeling technique can capture all the relevant concepts of a complex, evolving domain or achieve complete domain expressiveness within a single, definitive approach.

This work focuses on addressing the first two gaps, which motivate the following research questions:

RQ1 - *To what extent can a modeling technique ensure that all the models it produces correspond to realizable systems in the domain?*

RQ2 - *Is it possible to define a process that ensures the completeness and validity of a model with respect to the information required for security assessment processes?*

A separate consideration concerns the third gap, which is not formulated as a research question in this work. As prior research established, full modeling-system completeness can be approached through incremental enrichment of the modeling formalism and knowledge bases [23–26]. Accordingly, we discuss how the adopted technique supports the representation of systems in our case studies, while recognizing that expressiveness of domain completeness remains an ongoing problem.

In line with the research questions, this work provides two main methodological contributions, complemented by an empirical evaluation on real-world systems.

(1) A modeling technique designed to ensure correctness at both the model and modeling-system level. We introduce a formal structuring of the Multi-purpose Application Composition Model (MACM) that defines how systems are represented and how models are constructed.

A key aspect of this contribution is the availability of automated guidance during model construction, which helps prevent the introduction of misleading structures and avoids masking missing or incomplete information. This structuring ensures that instantiated models remain coherent, well-formed, and free from unrealizable configurations.

(2) A cyclical, evidence-driven refinement process for progressively achieving model validity and completeness. Building on the ESSecA methodology, we define an iterative process that integrates empirical evidence and domain knowledge to refine the MACM model while preserving its guarantees. Each iteration updates the model with newly discovered system behaviour, missing assets, or corrected assumptions, ensuring that the representation increasingly aligns with the real system.

(3) Evaluation through real-world case studies. The proposed approach is shown through two case studies. The eWeLink ecosystem illustrates the representational capabilities and correctness properties of MACM, while the JetRacer autonomous-vehicle environment highlights the complete cyclical workflow, including iterative refinement and automated security assessment. Together, these case studies suggest the practical effectiveness of the proposed methodology.

4. Multi-purpose Application Composition Model

Before introducing the modeling solution proposed in this work, we first clarify the characteristics that such a modeling system must satisfy. To support the elicitation of these characteristics, we consider an example of system architecture. Specifically, we refer to the architecture of an IoT system, involving real-world commercial products: the ITEAD Sonoff Micro device[2] integrated with the eWeLink platform[3], already analyzed in [25]. The ITEAD Sonoff Micro is a smart USB adapter that allows a standard USB port to function as a remotely controlled switch. Its features include programmable charge interruption, power status monitoring, and remote control of connected devices.

The architecture is shown in Fig. 2 and shows the main components involved in the eWeLink ecosystem and their network-level interactions. At the edge of the system, the Sonoff Micro is connected to the user's LAN through the home Wireless Router. The user interacts with the system through the eWeLink app, installed on a smartphone, connected to the same LAN via Wi-Fi. Beyond the local network, both the smartphone and the Sonoff device establish connections to the Internet (WAN), where the Coolkit Cloud Platform provides backend services for cloud-side management, while the Config API is used for device registration and authentication.

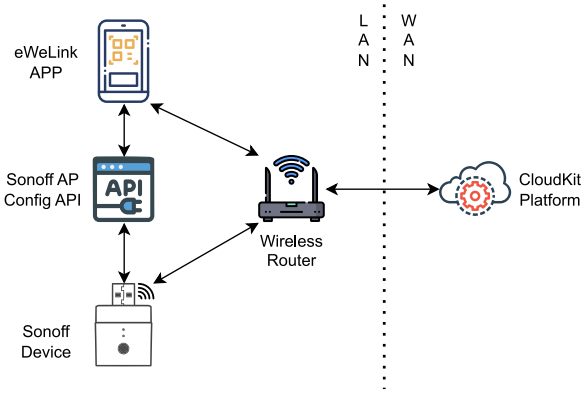

Fig. 2. eWeLink Architecture.

The first important aspect to clarify is that the modeling system is **security-oriented**, meaning that it does not aim to represent all the characteristics of the real system being modeled, but only those that are relevant from a security perspective. Therefore, from a security standpoint-and to the best of our knowledge-it is not relevant whether the smartphone in Fig. 2 is painted grey, whereas it is relevant whether it is connected to a router.

Another important feature that the modeling system must satisfy is known as **Snapshot Semantics**: it captures the architecture of a system as a snapshot - a static and coherent view of its structure at a specific point in time. It does not yet model runtime behavior, dynamic reconfiguration, failover mechanisms, or auto-scaling processes. For example, the Sonoff Micro device requires a pairing process with the account of the end-user and, at this stage, the modeling system does not aim to capture such operational procedures.

Moreover, the modeling system must support the representation of both concrete systems, for example, for security analysis, and planned or hypothetical architectures, for example, during the design phase, following a security-by-design approach (**Abstract and Realized Assets**). Therefore, not all modeled assets must necessarily exist in the deployed system: they may also represent components that are expected, planned, or potentially to be provisioned.

[2] https://itead.cc/product/sonoff-micro-5v-usb-smart-adaptor/
[3] https://ewelink.cc

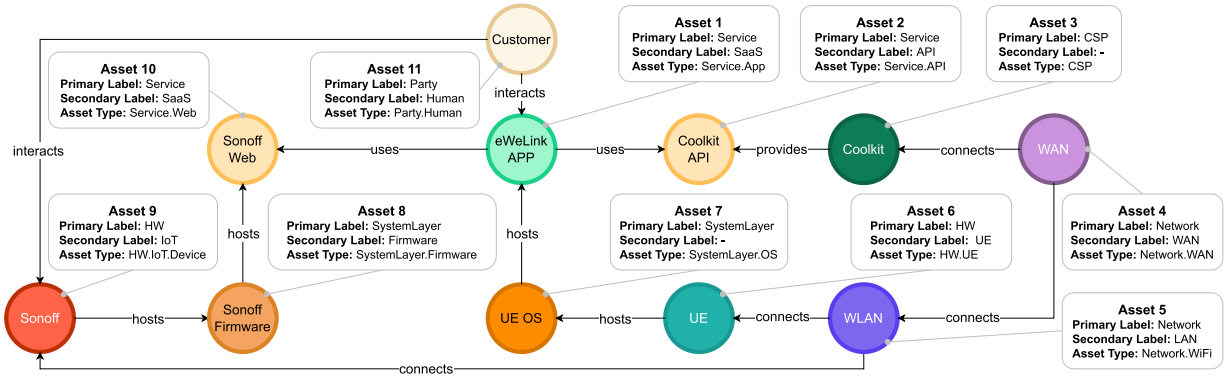

Fig. 3. eWeLink MACM.

Some constraints defined in the following for the modeling system (for example, the uniqueness of service hosting) are valid under the *snapshot* assumption, but may not hold globally in systems with dynamic infrastructures (for example, Kubernetes). In such contexts, the model must be interpreted as one possible concrete deployment instance.

Finally, the modeling system assumes a closed and finite set of nodes and edges (**Finite and Closed World**): external, non-modeled elements are considered unknown and have no implicit effect on validation.

Looking at the architecture just described, the first requirement we can impose on the modeling system is the ability to explicitly represent the assets that compose the target system, organizing them into coherent categories (or types). It is already evident that categorization is more effective when performed at different levels of granularity. It is possible, for example, to distinguish between hardware and software assets; among software assets we may find services; among services we may identify web servers, API servers, web apps, databases, and so on.

Moreover, the modeling system must allow the description of the relationships that exist among such assets. These relationships may also differ in nature: we may have hosting relationships, indicating that an asset is hosted by another asset; we may have connection relationships between networks and endpoints; we may have usage relationships among the various services.

Finally, to support automatic verification and the generation of correct models, it is necessary that such categories, properties, and relationships should be defined within a schema that rigorously specifies which nodes, which properties, and which links are allowed. The requirements just outlined guide the formal definition presented in the following sections.

The attentive reader may have already noticed that all the characteristics just described can be obtained by adopting graphs as the underlying structure, and more specifically by using **Property Graphs** [14,15]. In summary, property graphs are graphs in which both nodes and edges are *typed* and can be characterized by properties. Moreover, property graphs can be accompanied by a **schema**, which provides a formal description of the node types, properties, and relationships allowed in the model; within such a schema it is also possible to define constraints that, in our case, are useful to ensure the structure and coherence with real systems, namely the *validity* and *completeness* properties of the models, introduced in Section 3.

In light of this, we can say that the **Multi-purpose Application Composition Model** (**MACM**) modeling system must produce a set of Property Graphs. From the semantic perspective, each node of the graph represents an **asset** of the SuA, while each edge represents a **relationship** between two assets. In the following, a formal definition of the MACM will be given, accompanied with the list of current syntactic and semantic constraints.

The MACM model representing the entire infrastructure described above is reported in Fig. 3. Note that the original model reported in [25], relying on the first version of the MACM, suffered from several limitations due to its semi-formal approach, like some ambiguities and lack of support from automated tools.

Each node in the MACM is annotated with a pair of labels that specify its role in the model: a mandatory *Primary Label*, providing a high-level categorization of the asset, and an optional *Secondary Label*, offering a domain-specific refinement. Nodes also include a small set of mandatory properties: the *Asset Type*, giving the most fine-grained classification; the *ID*, uniquely identifying the node; and the *name*, describing the asset. Edges, instead, are characterized by a single label denoting the *relationship type*, possibly accompanied by optional parameters (not used in this example).

These assumptions guide the formal structure of the modeling system and the semantics of the constraints defined in later sections.

4.1. Correctness Anti-patterns

As discussed in Section 3, proving the full correctness of a modeling system would require demonstrating that *every* model it can generate corresponds to at least one real and coherent implementation. For the MACM modeling system this is not currently feasible: the expressiveness of the model, the heterogeneity of IT infrastructures, and the lack of a complete formal characterization of all possible real systems make such a proof unattainable.

Table 2
Extract of the Anti-Pattern List reported in Table A.8.

ID	Anti-Pattern
AP2	The modeling system must not allow describing a scenario in which a service lacks a deployment context.
AP7	The modeling system must not allow describing a scenario in which hardware directly executes application services (it can only execute firmware or operating systems).
AP11	The modeling system must not allow describing a scenario in which a network provides services or hosts systems.
AP25	The modeling system must not allow describing a scenario in which an operating system or firmware directly hosts other operating systems or firmware. It requires a dedicated virtualization or containerization layer.
AP30	The modeling system must not allow describing a scenario in which cycles are created in hosting relations. Indeed, it is impossible for a chain of mutually hosting components to return to the starting point, such as: hardware HW1 hosts HW2, HW2 hosts HW3, and HW3 hosts HW1 again. Such a structure cannot occur in real systems, where hosting relations are always hierarchical and acyclic.

For this reason, instead of proving that all generable models are realizable, we adopt a complementary strategy: starting from the most general asset categories (Primary Labels), we identify entire *classes of configurations that are certainly not realizable* in real IT infrastructures and ensure that the modeling system *cannot* construct models belonging to these classes. These forbidden configurations, here referred to as **Correctness Anti-patterns**, capture structural or semantic violations that no concrete system could exhibit.

As an illustration, we present a few very general examples of configurations which, although they can be abstractly generated by the model, are incompatible with any real infrastructure. One such case is that of assets lacking even the minimal infrastructural support, for instance an application service that is neither hosted by an operating system nor provided by an external provider **(AP2)**. Another involves relationships that violate fundamental structural constraints, such as a network establishing a hosting relation toward an operating system **(AP11)**, or a hardware device creating a direct connection to an application service as though it were a network endpoint **(AP7)** . There are also technologically impossible dependency chains, like an operating system hosted directly by another operating system without any virtualization layer **(AP25)**. Finally, some configurations would entail semantically impossible cycles, for example two services hosting each other **(AP30)**. These examples do not exhaust all the cases identified in this work, but they provide an initial intuition of the role played by Anti-patterns. The full list of Anti-patterns currently identified is reported in Table A.8, organized according to the involved asset categories.

The Anti-patterns therefore serve as *negative* correctness conditions: although they do not guarantee that every admissible model is realizable, they guarantee that a wide range of *non*-realizable or inconsistent systems are systematically excluded. By preventing the construction of these invalid categories of models, the modeling system increases its alignment with real-world constraints and improves the reliability of automated validation.

These Anti-patterns represent the first operational step toward strengthening correctness assurance within the MACM framework, and they directly guided the definition of the MACM Property Graph Schema introduced in the following section. In Table A.8, we also report all relationships between the Anti-patterns and the corresponding design choices adopted in the following section.

4.2. Formal definition

The MACM modeling system is based on the concept of a Property Graph, defined as a graph in which edges and nodes are characterized by sets of *labels* and sets of *properties*, in the form $(key, value)$ [14,15].

Let us consider an infinite set of labels $\mathbb{L}$ for edges and nodes, an infinite set of property names $\mathbb{P}$, an infinite set of atomic values $\mathbb{V}$, and a finite set of datatypes Q (e.g., String, Integer).

Definition 1 (Property Graph)**.** A Property Graph [14,16] is defined as a tuple $G = (N, E, \rho, \lambda, \sigma)$, where:

- N is a finite set of nodes;
- E is a finite set of edges (disjoint from N);
- $\rho : E \to (N \times N)$ is a function, known as the *incidence function*, which associates each edge with a pair of “source" and “target" nodes;
- $\lambda : (N \cup E) \to \mathrm{SET}^{+}(\mathbb{L})$[4] is a function that assigns to each node/edge the set of labels describing them, taken from the infinite label set $\mathbb{L}$;
- $\sigma : (N \cup E) \times \mathbb{P} \to \mathrm{SET}^{+}(\mathbb{V})$ is a function that assigns to each node/edge a set of properties, taken from the infinite set $\mathbb{P}$, and to each $(node/edge, property)$ pair a set of values, taken from the infinite set of atomic values $\mathbb{V}$.

Since the sets $\mathbb{L}, \mathbb{P}, \mathbb{V}$ are infinite, it is necessary to define which subsets are allowed for a given model. This leads to the concept of a **Property Graph Schema**, which defines the admissible types of nodes and edges, and the properties allowed for each of them.

Definition 2 (Property Graph Schema)**.** A Property Graph Schema is defined as a tuple $S = (L_N, L_E, \beta, \delta)$, where:

[4] Note that, given a set $\mathbb{K}$ we denote $\mathrm{SET}(\mathbb{K})$ the set of all finite subsets of $\mathbb{K}$, and $\mathrm{SET}^{+}(\mathbb{K})$ excludes the empty set.

- $L_N \subset \mathbb{L}$ is a finite set of labels, representing node types;
- $L_E \subset \mathbb{L}$ is a finite set of labels, representing edge types;
- $\beta : (L_N \cup L_E) \times \mathbb{P} \to Q$ is a function that defines the allowed properties for each node or edge type, and the corresponding datatype, taken from a finite set Q;
- $\delta : (L_N, L_N) \to \text{SET}^+(L_E)$ is a function that defines the admissible edge types between two nodes of given types.

Note that in Definition 2, no hierarchy is defined over the elements of the set L_N. This is unfortunately a limitation for the intended use of MACM, which led us to extend this definition by making it more rigid in certain aspects.

Since MACM is designed to model entire IT infrastructures, characterized by heterogeneous components and services, it is appropriate to classify the assets, and therefore the graph nodes, with multiple levels of granularity. Therefore, as anticipated above, we introduce the *Primary Labels*, which define a high-level classification of nodes in the model, denoting the main roles or categories of assets in the system, and the *Secondary Labels*, which provide a finer semantic classification of nodes within the general role defined by the primary labels. Accordingly, the label set L_N is divided into two sets, L_P and L_S. Thus, as it will be defined formally later, we developed an extended definition of a property graph as $M = (N, E, \rho, \lambda_N, \lambda_E, \sigma)$, where λ_N concerns the node types, while λ_E is related to edge types.

Furthermore, two new sets are introduced in the schema, containing the constraints to be satisfied for a MACM to be considered correct. In particular, we distinguish between two categories of constraints: syntactic constraints and semantic constraints. **Syntactic constraints** are mandatory and must be satisfied for the MACM model to be considered structurally valid. A violation of a syntactic constraint results in an invalid or unprocessable model. **Semantic constraints** represent optional validation rules that capture domain-specific aspects, best practices, or semantic consistency conditions. Although not strictly mandatory, violations of semantic constraints generate warnings and typically indicate potential design issues or misconfigurations.

Consequently, the definition of a MACM Property Graph Schema becomes the following.

Definition 3 (MACM Property Graph Schema)**.** A MACM Property Graph Schema is defined as a tuple $S_M = (L_P, L_S, R, \beta, \delta, \Sigma, \Gamma)$, where:

- $L_P \subset \mathbb{L}$ is a finite set of primary labels, which provide a high-level classification of the nodes in the model;
- $L_S \subset \mathbb{L}$ is a finite set of secondary labels, with a function $LS : L_P \to \text{SET}(L_S)$ mapping each primary label to its corresponding subset of admissible secondary labels; the secondary labels provide a finer-grained semantic classification of the nodes within the general role;
- $R \subset \mathbb{L}$ is a finite set of labels representing edge types;
- $\beta : (L_P \times L_S \cup R) \times \mathbb{P} \to Q$ is a partial function that assigns to each node label pair $(PrimaryLabel, SecondaryLabel)$ or edge type the admissible properties and the corresponding datatype from a finite set Q;
- $\delta : (L_P, L_P) \to \text{SET}^+(R)$ is a function defining the allowed edge types between two nodes with given primary labels;
- Σ is the set of *syntactic constraints*;
- Γ is the set of *semantic constraints*.

The *semantic constraints* may vary according to the application domain and the schema configuration, so they will be described in the following sections. The *syntactic constraints* are fixed and here listed:

- $P_m \subseteq \text{dom}(\beta)$, the set of mandatory properties (must always be present for nodes/edges of the specified type);
- $P_o \subseteq \text{dom}(\beta)$, the set of optional properties (may be omitted), with $P_o \cap P_m = \emptyset$ and $P_o \cup P_m = \text{dom}(\beta)$;
- $P_u \subseteq P_m$, a subset of mandatory properties that must have unique values (e.g., *id*);
- $Cardinality : \text{dom}(\beta) \to \mathbb{N} \times (\mathbb{N} \cup +\infty)$, a function that assigns to each pair $(t, p) \in \text{dom}(\beta)$ an interval (m_p, M_p), where m_p is the minimum and M_p the maximum allowed cardinality for the set of values of property p in objects of type t.

A (property) graph is a MACM only if it is consistent with the MACM Property Graph Schema, resulting both syntactically and semantically correct, according to the following definition:

Definition 4 (Schema-Instance Consistency for MACM)**.**

Let $S_M = (L_P, L_S, R, \beta, \delta, \Sigma, \Gamma)$ be a MACM schema, and let $M = (N, E, \rho, \lambda_N, \lambda_E, \sigma)$ be a property graph, where:

- $\lambda_N : N \to L_P \times (L_S \cup \emptyset)$ assigns to each node the pair $(PrimaryLabel, SecondaryLabel)$;
- $\lambda_E : E \to R$ assigns to each edge a single relation type.

We say that M is *correct with respect to* S_M, and thus that M *is a MACM*, if all the following conditions hold:

1. **Label Validity**
   - $\forall n \in N \implies \lambda_N(n) \in L_P \times (L_S \cup \emptyset)$;
   - $\forall e \in E \implies \lambda_E(e) \in R$.
2. **Valid Relationship Patterns**
   $\forall e \in E : \rho(e) = (n_s, n_t) \implies \lambda_E(e) \in \delta(\lambda_N(n_s), \lambda_N(n_t))$.
3. **Admissible and Typed Properties**
   Assuming that for every $v \in \mathbb{V}$, the function $type(v)$ returns the datatype of v, then:
   - $\forall (n,p) = V_p \in \text{dom}(\sigma), \big(\lambda_N(n), p\big) \in \text{dom}(\beta) \wedge \forall v \in V_p, type(v) = \beta\big(\lambda_N(n), p\big)$;
   - $\forall (e,p) = V_p \in \text{dom}(\sigma), \big(\lambda_E(e), p\big) \in \text{dom}(\beta) \wedge \forall v \in V_p, type(v) = \beta\big(\lambda_E(e), p\big)$.
4. **Mandatory Properties**
   $\forall (x,p) \in P_m \subseteq \text{dom}(\beta) \wedge \forall o \in N \cup E : \lambda(o) = x \implies (o,p) \in \text{dom}(\sigma)$.
5. **Unique Properties**
   Let $P_u \subseteq P_m$ be the set of mandatory properties that must have unique values, then:
   - $\forall n_1, n_2 \in N : \big(n_1 \neq n_2 \wedge \lambda_N(n_1) = \lambda_N(n_2)\big) \implies \forall p \in P_u, \sigma(n_1,p) \neq \sigma(n_2,p)$
   - $\forall e_1, e_2 \in E : \big(e_1 \neq e_2 \wedge \lambda_E(e_1) = \lambda_E(e_2)\big) \implies \forall p \in P_u, \sigma(e_1,p) \neq \sigma(e_2,p)$
6. **Cardinality of Property Values**
   - $\forall n \in N, \forall \big(\lambda_N(n), p\big) \in \text{dom}(\beta) : (m_p, M_p) = Cardinality\big(\lambda_N(n), p\big) \implies m_p \leq |\sigma(n,p)| \leq M_p$;
   - $\forall e \in E, \forall \big(\lambda_E(e), p\big) \in \text{dom}(\beta) : (m_p, M_p) = Cardinality\big(\lambda_E(e), p\big) \implies m_p \leq |\sigma(e,p)| \leq M_p$.
7. **Additional Constraints**
   All constraints in Σ and in Γ must be satisfied.

4.3. Schema configuration

So far, we have provided the definition of a MACM model as a Property Graph and formalized its schema. At this point, the remaining step is to configure the schema by explicitly specifying all the sets defined above.

Primary Labels. We define the set of admissible primary labels $L_P = \{\texttt{Party}, \texttt{CSP}, \texttt{Service}, \texttt{HW}, \texttt{Network}, \texttt{Virtual}, \texttt{SystemLayer}\}$.

Secondary Labels. For each $\ell_p \in L_P$, the associated secondary label set $LS(\ell_p) \subseteq L_S$ are:

$$
\begin{aligned}
LS(\texttt{Network}) &= \{\texttt{WAN}, \texttt{LAN}, \texttt{PAN}\} \\
LS(\texttt{CSP}) &= \emptyset \\
LS(\texttt{Virtual}) &= \{\texttt{VM}, \texttt{Container}\} \\
LS(\texttt{SystemLayer}) &= \{\texttt{OS}, \texttt{Firmware}, \texttt{ContainerRuntime}, \texttt{Hypervisor}\} \\
LS(\texttt{Service}) &= \{\texttt{App}, \texttt{Server}\} \\
LS(\texttt{Party}) &= \{\texttt{Human}, \texttt{LegalEntity}, \texttt{Group}\} \\
LS(\texttt{HW}) &= \{\texttt{Server}, \texttt{IoT}, \texttt{Device}, \texttt{Microcontroller}, \texttt{SOC}, \texttt{MEC}, \texttt{PC}\}
\end{aligned}
$$

Relationship Types. The set of admissible relationship types is $R = \{\texttt{uses}, \texttt{hosts}, \texttt{provides}, \texttt{connects}, \texttt{interacts}\}$.

Relationship pattern validity. The primary labels of the source and target nodes, together with the relationship type, must correspond to one of the patterns defined in Table 3. Each row of the table has to be intended as $\delta(SourcePrimaryLabel, TargetPrimaryLabel) = RelationshipType$.

Properties. The Property configuration within the schema is defined through the function β, as introduced in Definition 3, that assigns to each pair $(PrimaryLabel,\ SecondaryLabel)$ or to each edge type the admissible properties and their corresponding datatype.

According to the definition, we would need to list a potentially long set of entries of the form $\beta\big((x,y),p\big) = q$, where x is the primary label, y is the secondary label, p is the property, and q is the datatype. Since these properties apply to all nodes regardless of type, we will use the wildcard pair $(*,*)$ to denote all node types and thus simplify the configuration.

As previously stated in the introduction of this section, there are three mandatory properties for nodes: the ID, the name, and the Asset Type. In particular, the Asset Type represents an additional, fine-grained classification of an asset. An Asset Type is tightly

Table 3
Relationship pattern validity.

ID	Source Primary Label	Relationship Type	Target Primary Label
RP1	Party	interacts	Service, HW, Network, Party, Virtual, SystemLayer, CSP
RP2	Service	uses	Service, Virtual
RP3	Service	hosts	Service
RP4	Virtual	hosts	SystemLayer
RP5	SystemLayer	hosts	SystemLayer, Virtual, Service, Network
RP6	SystemLayer	uses	HW
RP7	HW	hosts	HW, SystemLayer
RP8	CSP	provides	Service, Network, HW, Virtual, SystemLayer
RP9	Network	connects	Network, Virtual, HW, CSP

coupled with the concepts of Primary and Secondary Label. Indeed, given an Asset Type, the pair $(PrimaryLabel, SecondaryLabel)$ is uniquely determined. The list of all Asset Types defined in this configuration, along with their corresponding label pair and a brief description, is available in the Asset Type Catalogue[5]. We now introduce the following definition, useful for expressing some constraints.

Definition 5 (Asset Type Set)**.** Let T be the set of Asset Types in the MACM model. Each asset type $t \in T$ is uniquely associated with a primary label $\ell_p \in L_P$ and a secondary label $\ell_s \in L_S \cup \{\emptyset\}$. We define a total mapping $\tau : T \to L_P \times (L_S \cup \{\emptyset\})$ such that for each asset type $t \in T$, we have $\tau(t) = (\ell_p, \ell_s)$.

Therefore, the following configurations apply to the nodes:

$$\beta\big((*,*), id\big) = \texttt{Integer}, \quad \beta\big((*,*), name\big) = \beta\big((*,*), asset\_type\big) = \texttt{String}$$

In terms of the edge properties configuration, properties are defined for the protocols used in some of the relationships between the assets, whenever communications occur over a network. We have chosen to adopt all the layers of the standard ISO/OSI communication stack, excluding the physical layer. Each layer is associated with a property:

$$p \in O = \{\texttt{data\_link\_protocol}, \texttt{network\_protocol}, \texttt{transport\_protocol}, \texttt{session\_protocol}, \texttt{presentation\_protocol}, \texttt{application\_protocol}\}$$

The following configuration applies:

- $\beta(\text{connects}, p) = \texttt{String}, \quad \forall p \in \{\texttt{data\_link\_protocol}, \texttt{network\_protocol}, \texttt{application\_protocol}\}$
- $\beta(\text{uses}, p) = \texttt{String}, \quad \forall p \in \{\texttt{transport\_protocol}, \texttt{session\_protocol}, \texttt{presentation\_protocol}, \texttt{application\_protocol}\}$

At this point, it is necessary to determine which of the above configurations $\beta(\cdot,\cdot)$ belong to the special sets P_m (mandatory properties), P_o (optional properties), or P_u (properties with unique values):

- $((*,*), id) \in P_u$
- $\forall x \in \{((*,*), name),\ ((*,*), asset\_type)\},\ x \in P_m$
- $\forall x \in \left\{ \begin{array}{l} (connects, data\_link\_protocol), (connects, network\_protocol), \\ (connects, application\_protocol), (uses, transport\_protocol), \\ (uses, session\_protocol), (uses, presentation\_protocol), \\ (uses, application\_protocol) \end{array} \right\},\ x \in P_o$

Since $P_u \subseteq P_m$, the property id is both unique and mandatory. The cardinality of each property is defined by the function $Cardinality(\cdot, p)$ and reported in Table 4.

According to this configuration, each property related to a protocol may be omitted; however, if present, its associated value set must comply with the defined limits. All protocol properties allow at most one value, except for the application-level property, which allows one or more values if used.

Disambiguation. *In this section, we have defined the Property Graph as a graph in which nodes are characterized by labels and properties. We also stated that the labels $l \in L$ define the "type" of a node. In the case of MACM, we have defined a hierarchy consisting of two types of labels: Primary Labels and Secondary Labels. The former provide a higher-level classification, while the latter serve to further specialize this classification. For example, to represent a Local Area Network (LAN) network, we use a node whose Primary Label is* `Network` *and whose Secondary Label is* `LAN`*. However, in MACM we have also defined Asset Types, which represent the most fine-grained classification assigned to a node. In the case of a LAN network, the Asset Type would be* `Network.LAN`*. It should first be noted that an Asset Type is not always composed of* `PrimaryLabel.SecondaryLabel`*. Furthermore, Asset Types are not labels but properties, and therefore it would be inaccurate to say that a node representing a LAN network is of "type"* `Network.LAN`*.*

[5] https://pennet.vseclab.it/catalogs/asset-types

Table 4
Cardinality of properties.

Property	Cardinality
$((*,*), id)$, $((*,*), name)$, $((*,*), asset\_type)$	$(1,1)$
$(connects, data\_link\_protocol)$, $(connects, network\_protocol)$, $(connects, application\_protocol)$, $(uses, transport\_protocol)$, $(uses, session\_protocol)$, $(uses, presentation\_protocol)$	$(0,1)$
$(uses, application\_protocol)$	$(0,+\infty)$

Nevertheless, throughout this document, the "type" of a node will often be referred to by indicating either its labels or its Asset Type. The reader can easily distinguish between the two classifications-label-level or Asset Type-by noting whether the indicated "type" contains a dot or not.

Constraints. The last element to be configured in the schema is the set of semantic constraints Γ, which, as previously explained, are used to express semantic characteristics specific to the application domain-in this case, IT systems.

In Definition 5, we defined the set of Asset Types T and stated that each Asset Type $t \in T$ is associated with a pair $(PrimaryLabel, SecondaryLabel)$. We now define a constraint ensuring that this relationship holds for every Asset Type of each node in the MACM:

Semantic Constraint 1 (Asset Type validity). $\forall n \in N, \sigma(n, \texttt{asset\_type}) \in T \wedge \tau\big(\sigma(n, \texttt{asset\_type})\big) = \lambda_N(n)$

The second constraint states that, in general, a node can have no more than one incoming relationship of type `hosts` or `provides`. In other words, an asset cannot be hosted or provided by multiple nodes at the same time, ensuring a single, well-defined deployment origin. Note that this does not mean that each asset in the MACM must have an incoming `hosts/provides` edge.

This constraint guarantees the snapshot assumption of the MACM model (see Section 4). In design-time scenarios or highly dynamic infrastructures, this condition applies to the specific instance of the architecture captured by the model, and not to the system's full lifecycle.

Semantic Constraint 2 (Single host/provide per Asset).
$\forall n_t \in N : \exists e \in E \wedge \rho(e) = (*, n_t) \wedge \lambda_E(e) \in \{\texttt{hosts}, \texttt{provides}\}, \nexists e' \in E : \wedge \rho(e') = (*, n_t) \wedge \lambda_E(e') \in \{\texttt{hosts}, \texttt{provides}\}$

The third constraint requires that each Service, SystemLayer or Virtual node must be hosted or provided by at least one other node. Since the Semantic Constraint 2 must hold, we can say that each Service, SystemLayer or Virtual node must be hosted or provided by *exactly* one other node. This guarantees that every Service, SystemLayer or Virtual has a unique deployment context in the system.

Semantic Constraint 3 (Mandatory host/provide per Service, SystemLayer and Virtual).
$\forall n_t \in N : \lambda_N(n_t) = (\texttt{Service}, *) \vee \lambda_N(n_t) = (\texttt{SystemLayer}, *) \vee \lambda_N(n_t) = (\texttt{Virtual}, *), \exists n_s \in N, \exists e \in E : \lambda_E(e) \in \{\texttt{hosts}, \texttt{provides}\} \wedge \rho(e) = (n_s, n_t)$

Semantic Constraint 4 requires that each `uses` relationship between two nodes must be supported by an alternative communication path that does not itself depend on a `uses` or an `interacts` edge. This ensures that functional dependencies are semantically meaningful only when there exists a viable infrastructure or network path that enables actual communication.

Semantic Constraint 4 (Alternate path for uses). *Let $M = (N, E, \rho, \lambda_N, \lambda_E, \sigma)$ a MACM property graph and $M' = (N, E', \rho, \lambda_N, \lambda_E, \sigma)$ the subgraph of M with $E' = \{e \in E \mid \lambda_E(e) \notin \{\texttt{uses}, \texttt{interacts}\}\}$, then $\forall e \in E$, where $\lambda_E(e) = \texttt{uses}$ and $\rho(e) = (n_s, n_t)$, exists a path in M' from n_s to n_t.*

We introduce constraints for *SystemLayer* nodes, representing middlewares. This primary label includes four asset types: *SystemLayer.OS* (e.g., Operating System), *SystemLayer.Firmware*, *SystemLayer.ContainerRuntime* (e.g., Docker), and *SystemLayer.HyperVisor* (e.g., Xen, VMware). First, we constrain the $\delta(\texttt{SystemLayer}, \texttt{SystemLayer}) = \texttt{hosts}$ pattern from Table 3. A *SystemLayer.OS* may host a *SystemLayer.ContainerRuntime* or a *SystemLayer.HyperVisor*, but other direct hosting between *SystemLayer* nodes is invalid. For instance, a *HyperVisor* cannot host an *OS* without an intermediate *Virtual* node. The following rule defines that a *SystemLayer.OS* can host only *SystemLayer.ContainerRuntime* or *SystemLayer.HyperVisor* nodes.

Semantic Constraint 5 (SystemLayer hosting SystemLayer node validity).
$\forall e \in E, \big(\lambda_E(e) = \texttt{'hosts'} \wedge \rho(e) = (n_s, n_t) \wedge \lambda_N(n_s) = \lambda_N(n_t) = (\texttt{SystemLayer}, *)\big) \implies \big(\sigma(n_s, \texttt{asset\_type}) = \texttt{'SystemLayer.OS'} \wedge \sigma(n_t, \texttt{asset\_type}) \in \{\texttt{'SystemLayer.ContainerRuntime'}, \texttt{'SystemLayer.HyperVisor'}\}\big)$

The second relationship pattern we wish to constrain is $\delta(\texttt{SystemLayer}, \texttt{Virtual}) = \texttt{hosts}$. In this case, since the Primary Label `Virtual` is used to represent either virtual machines (of type `Virtual.VM`) or containers (of type `Virtual.Container`), only the following relationships are valid: `SystemLayer.ContainerRuntime hosts Virtual.Container` and `SystemLayer.HyperVisor hosts Virtual.VM`.

Semantic Constraint 6 (SystemLayer hosting Virtual node validity). *If a SystemLayer node hosts a Virtual node, the source node must be of type* `SystemLayer.ContainerRuntime` *or* `SystemLayer.HyperVisor`. *In the first case, the target node must be of type* `Virtual.Container`, *in the latter must be of type* `Virtual.VM`.

$$\forall e \in E, \big(\lambda_E(e) = \text{'hosts'} \ \wedge \ \rho(e) = (n_s, n_t) \wedge \lambda_N(n_s) = (SystemLayer, *) \ \wedge \ \lambda_N(n_t) = (Virtual, *) \implies \big(\sigma(n_s, \texttt{asset\_type}) \in \{\text{'SystemLayer.ContainerRuntime'}, \text{'SystemLayer.HyperVisor'}\}\big), \sigma(n_t, asset\_type) = \begin{cases} \text{'Virtual.Container'} & \text{if } \sigma(n_s, asset\_type) = \text{'SystemLayer.ContainerRuntime'} \\ \text{'Virtual.VM'} & \text{if } \sigma(n_s, asset\_type) = \text{'SystemLayer.HyperVisor'} \end{cases}$$

Regarding SystemLayer nodes that host Service nodes, the former can only be operating systems (`SystemLayer.OS`) or firmware (`SystemLayer.Firmware`). Therefore, the following constraint applies.

Semantic Constraint 7 (SystemLayer hosting Service node validity). *If a SystemLayer node hosts a Service node, the source node must be of type* `SystemLayer.OS` *or* `SystemLayer.Firmware`.

$$\forall e \in E, \ \big(\lambda_E(e) = \text{'hosts'} \wedge \rho(e) = (n_s, n_t) \wedge \lambda_N(n_s) = (\texttt{SystemLayer}, *) \wedge \lambda_N(n_t) = (\texttt{Service}, *)\big) \implies \sigma(n_s, \texttt{asset\_type}) \in \{\text{'SystemLayer.OS'}, \text{'SystemLayer.Firmware'}\}$$

Furthermore, with regard to nodes of type `SystemLayer`, we also want to constrain the pattern $\delta(\texttt{Virtual}, \texttt{SystemLayer})$. Specifically, it is valid for a `Virtual.VM` node to host a node of type `SystemLayer.OS` or `SystemLayer.Firmware`, but it is not valid for a `Virtual.VM` node to host a `SystemLayer.HyperVisor` node.

Semantic Constraint 8 (Virtual hosting SystemLayer node validity). *If a Virtual node hosts a SystemLayer node, the target node must be of type* `SystemLayer.OS` *or* `SystemLayer.Firmware`.

$$\forall e \in E, \big(\lambda_E(e) = \text{'hosts'} \wedge \rho(e) = (n_s, n_t) \wedge \lambda_N(n_s) = (\texttt{Virtual}, *) \wedge \lambda_N(n_t) = (\texttt{SystemLayer}, *)\big) \implies \big(\sigma(n_t, \texttt{asset\_type}) \in \{\text{'SystemLayer.OS'}, \text{'SystemLayer.Firmware'}\}\big)$$

A final constraint concerning `SystemLayer` nodes aims to restrict the cases in which they are hosted by hardware, specifically constraining the pattern $\delta(\texttt{HW}, \texttt{SystemLayer}) = \texttt{hosts}$. In particular, it is not valid for a node of type `HW` to directly host a node of type `SystemLayer.ContainerRuntime`.

Semantic Constraint 9 (Hardware hosting SystemLayer node validity). *If a HW (hardware) node hosts a SystemLayer node, the target node can not be of type* `SystemLayer.ContainerRuntime`.

$$\forall e \in E, \ \big(\lambda_E(e) = \text{'hosts'} \wedge \rho(e) = (n_s, n_t) \wedge \lambda_N(n_s) = (\texttt{HW}, *) \wedge \lambda_N(n_t) = (\texttt{SystemLayer}, *)\big) \implies \sigma(n_t, \texttt{asset\_type}) \neq \texttt{SystemLayer.ContainerRuntime}$$

At this point, we define a constraint for the edge parameters related to protocols. The list of supported protocols is available online[6]. For each protocol, the type of relationship to which it applies, its corresponding layer in the ISO/OSI stack, and a brief description are provided. Therefore, the following constraint must hold.

Semantic Constraint 10 (Protocol validity). *Let $Y \subseteq R \times P \times V$ be the set of triples (r, o, v) representing the protocols allowed in the MACM, where r is a type of relationship (e.g.,* `uses`*), o is a property key associated with a layer of the ISO/OSI model (e.g., application_protocol), and v is the name of a valid protocol (e.g., HTTP).*

$$\forall e \in E, \ \forall p \in O \subseteq P, \ \big(\lambda_E(e) \in \{\texttt{connects}, \texttt{uses}\} \wedge (e, p) \in \text{dom}(\sigma)\big) \implies \forall v \in \sigma(e, p), \ (\lambda_E(e), p, v) \in Y$$

The last two constraint concerns the structure of a MACM property graph M: it must be *weakly connected* and *acyclic* over some types of relationships in R_{acyc}.

Semantic Constraint 11 (Graph Connectivity). *Let $M' = (N, E', \rho, \lambda_N, \lambda_E, \sigma)$ be the undirected version of M, then $\forall n_i, n_j \in N$, there exists a path in M' connecting them.*

Semantic Constraint 12 (Acyclicity over Specified Relations). *Let $R_{\text{acyc}} = \{\texttt{hosts}\} \subseteq R$ be the subset of relation types that must not form cycles and let $M_{\text{acyc}} = (N, E, \rho, \lambda_N, \lambda_{E,\text{acyc}}, \sigma)$, where $\forall e \in E, \lambda_{E,\text{acyc}}(e) \in R_{\text{acyc}}$. Then M_{acyc} must be a Directed Acyclic Graph (DAG).*

At this stage, the reader should have gathered that, in the simplest form of a MACM, each node is characterized by a *Primary Label*, an (optional) *Secondary Label*, and an *Asset Type*, where these three elements describe, at different levels, the type of asset that the node represents. In addition, there is an *ID* (unique) and a *name* describing the asset. Clearly, further properties can be added as needed. As for the edges, they are simply characterized by a type, the pair of nodes they connect and, optionally, properties related to protocols.

5. MACM Processing and evaluation

The formalization of the MACM model as a property graph enables us to adopt Neo4j[7], a graph database that fully implements the characteristics of a Property Graph as a basis for both storing, automatically verify correctness of the models and update the models, granting their syntactic and semantic coherence.

[6] https://pennet.vseclab.it/catalogs/protocols
[7] https://neo4j.com

5.1. Storing and manipulating a MACM

Neo4j uses Cypher as its query language, a declarative query language that allows for creating and manipulating graphs. For example, considering the MACM shown in Fig. 3 stored in Neo4j, if we wanted to obtain the name, labels, and type of the node hosted by the node `Sonoff Firmware`, we could use the following query:

```
match (ns:SystemLayer {name:"Sonoff Firmware"}) -[:hosts]->(nt)
return nt.name, labels(nt), nt.type
```

Note that `ns` and `nt` are *variables* in Cypher, while `labels()` is a built-in function that returns the labels; in our case, the Primary and Secondary Label. If, instead, we wanted to know on which network the service `Sonoff` is exposed, we could use the following query:

```
match (network:Network)-[]->(:HW)-[:hosts*1..]->(sonoff_web:Service {name:"
    Sonoff Web"})
return network.name, labels(network), network.type
```

This query contains a complex pattern: we are considering the case in which there is a node of type `Network` connected to a node of type `HW`, which in turn is connected by one or more `hosts` edges to the target node `Sonoff Web`.

5.2. Verification of MACM correctness

To check the consistency of a system model with the MACM schema, we represent each constraint as a Cypher query that detects violations directly on the graph. When a constraint is respected the result is a void, while when constraints are not respected, the queries return the subgraphs that outline the error, helping the modeler to correct the MACM accordingly.

Each Cypher query implementing a constraint is implemented as a `trigger` that runs automatically whenever a graph is created or updated. This ensures that violations are detected immediately and the system model remains consistent throughout its lifecycle. Since the validation is cyclic, whenever new nodes or relationships are added, the resulting MACM must still satisfy all rules. The overall verification algorithm is equivalent to the logical **AND** of all individual rules: the model is correct if and only if all constraints return no violations.

Example 1 (Query for Constraint 4)**.**

Semantic Constraint 4 states that any `uses` relationship between two nodes must be supported by an alternative communication path that does not itself rely on `uses`/`interacts` edges. This ensures that a declared dependency is feasible at the architectural level, for example through a network connection. A simplified trigger for verifying this constraint might be written using APOC[8] Neo4j library. We developed a trigger called *check_uses_path* that calculates the number of violations using the following query:

```
    MATCH (a)-[:uses]->(b)
    WHERE NOT EXISTS { MATCH p = (a)-[*1..]-(b)
    WHERE ALL(rel IN relationships(p) WHERE NOT type(rel) IN ["uses", "
        interacts"])
```

The query is executed whenever a MACM is created or modified, and it raises an alert if the number of detected violations (e.g. query results) is greater than zero.

Example 2 (Violation of Semantic Constraint 4)**.** Consider two services, `eWeLink_APP` and `Sonoff_Web`, hosted on two different hardware nodes, `UE` and `Sonoff`, respectively. If `eWeLink_APP` has a `uses` relationship to `Sonoff_Web` but there is no network (no `connects` path) linking `UE` and `Sonoff`, the communication is physically impossible and the constraint is violated. In our scenario, we modified the MACM presented above, as shown in Fig. 4.

As in evidence, this MACM isolates the exact nodes and relationships from your MACM and intentionally omits any `connects` (or other non-`uses`) path between `UE` and `Sonoff`, thus reproducing a concrete violation of Semantic Constraint 4.

The implementation of APOC Neo4j triggers additionally provides safeguards against incorrect MACM modifications, such as the addition of invalid components or relationships. This capability proves essential for the cyclical penetration testing methodology, that will be presented in Section 6. The complete list of triggers is available on GitHub[9]. Note that the verification that the edges belong to the list `uses`, `interacts`, `provides`, `connects`, `hosts` is implicitly handled in the implementation of the rule described in Table 3, which represents a more stringent requirement by not only restricting the edges to that list but also enforcing constraints on the primary labels of both the source and target nodes. Note also that the triggers are employed not only to enforce semantic constraints, but also to handle structural constraints, thereby ensuring a consistent implementation.

5.3. Automated model updating

The MACM supports automated updating, i.e. given an initial system model it is possible to apply a transformation able to enrich the model with new information and/or take into account changes in the system (as an example a change in deployment). Given that the system model is represented as a Property Graph, graph-grammar theory, and, in particular, *production rules* for graph modification can be applied to formalize this process.

[8] https://neo4j.com/docs/apoc/2025.07/background-operations/triggers/

[9] https://github.com/VSecLab/Docs/blob/main/MACM/ModelChecking.md

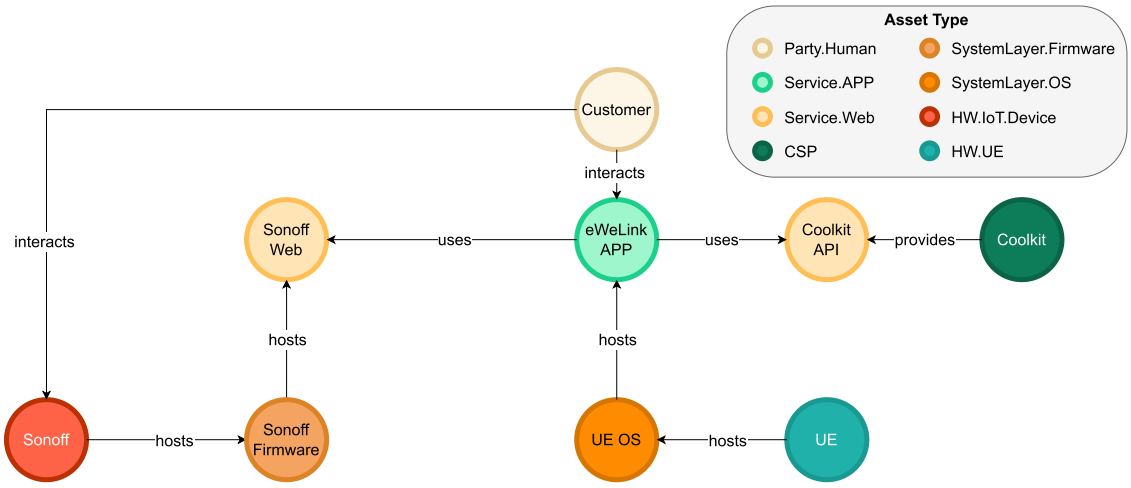

Fig. 4. Incorrect MACM.

A *production* is formally defined as a triple $\Pi = \langle G^L, G^R, \mathcal{I} \rangle$, where G^L denotes the subgraph of G to be replaced, G^R is the subgraph to be added to G, and $\mathcal{I}$ represents the embedding transformations, i.e., the rules that determine how G^R must be connected to G among the infinitely many possible alternatives. More specifically, an embedding transformation is a set of quadruples $\mathcal{I} = \{(EE, RN, ET, ED)\}$, referred to as embedding transformation fragments. Here, EE is the expression identifying the nodes of G^L that must be connected to the new nodes in G^R; RN specifies the nodes of G^R to be connected to those indicated in EE; ET defines the types of edges to be used for connecting nodes in EE to those in RN; and finally, $ED \in \{0, 1\}$ indicates the edge direction, where $ED = 1$ denotes an edge from EE to RN, and $ED = 0$ denotes the opposite. A detailed application of this process is illustrated in Section 7.

6. Iterative modeling and automation

Modern security assessment processes are inherently dynamic, as system architectures evolve over time and new vulnerabilities emerge continuously. Consequently, a static assessment approach-based on a single, fixed representation of the system-proves insufficient to ensure consistent accuracy and coverage of the analysis. To address this limitation, the formal system model and the automated reasoning processes must be integrated into a cyclic workflow, where the outcomes of each assessment iteration contribute to refining the model itself.

This section introduces the integration between the formal MACM model and the ESSecA methodology – initially presented in [27] and later extended in [23]. The objective is to establish a closed feedback loop in which the results of Threat Modeling, Risk Analysis, and Penetration Testing are not merely outputs, but structured inputs for subsequent model updates. This continuous interaction between modeling and execution increases both the *completeness* and the *validity* of the system model, thereby enabling progressively more precise Threat Models and Penetration Test Plans in subsequent iterations. The proposed approach highlights how automation and iterative refinement can improve the overall security assessment process by incorporating model-driven automation mechanisms, allowing the system representation to evolve dynamically as new evidence is collected during testing activities.

6.1. Related works

In considering automation within security assessment, current research highlights a clear asymmetry between Threat Modeling and Penetration Testing. In the field of Threat Modeling, several tools are already capable of automatically deriving structured representations of threats from a system model; however – as noted in Section 2 – these models are not subjected to formal correctness checks and may therefore be incomplete or inconsistent.

Regarding Penetration Testing, automation remains fragmented and limited to specific operational phases. Most contributions focus on narrow domains, such as Web application pentesting, where automated tools identify vulnerabilities, attempt exploits, and generate structured reports [28,29]. In other cases, automation is restricted to the initial reconnaissance phase-network discovery, host identification, and service enumeration-as shown, for instance, in the framework proposed by Zeng et al [30]. Other studies target vertical domains, such as the Internet of Vehicles, where tests are adapted to scenario-specific threats [31]. Only recent research efforts address more advanced aspects, such as post-exploitation modeling and adaptive attack planning, integrating simulation and dynamic updates of attack plans, as in the work by Wang et al [32].

However, a structural limitation is shared by both Threat Modeling and Penetration Testing techniques: to the best of our knowledge, no existing approach introduces a real *feedback loop* in which the outcomes of security or penetration tests are formally reintegrated into the system model to update its completeness, or validity. The absence of this re-elaboration step keeps models static and prevents automation from supporting a truly iterative evaluation cycle.

6.2. The ESSecA methodology and architecture

Fig. 5. ESSecA Methodology.

ESSecA is a methodology for automating security assessment, Threat Modeling, and Penetration Testing across various types of IT infrastructures, including IoT, Cloud, and Cyber-Physical Systems. As illustrated in Fig. 5, it leverages a system model to identify threats, perform Risk Analysis, and automatically generate Penetration Test Plans, integrating structured knowledge bases. ESSecA is founded on three core principles: it is *threat-based*, since threats are the primary lens for identifying system vulnerabilities; *catalog-based*, because the analysis relies on structured knowledge bases (catalogs) to produce results; and *model-driven*, as the entire process is guided by the system model. This complete methodology is fully implemented in an open-source tool, called PenNet, available on GitHub[10]. The architecture is depicted in Fig. 6.

To perform the first phase of the methodology, the security analyst constructs the *Initial System Model* using the MACM formalism, presented in Section 4, by identifying all the assets and relationships among them in the target system. The produced cypher query is then updated in the tool via the web *User Interface*. A first enhancement with respect to the original architecture is the introduction of the *MACM Checker*, the component that verifies the *correctness* of the model against the MACM Schema, as explained in Section 5.

The *Threat Modeling* phase aims to systematically determine the potential threats that may compromise the security of the SuA. The methodology describes each threat through three interconnected components: the *threat agent*, which represents the entity (human or automated) capable of initiating the malicious action; the *asset*, corresponding to the specific system element that could be targeted; and the *behavior*, which characterizes the nature and objectives of the malicious action. By combining these elements, each threat can be formally expressed as a triple (TA, A, B). The overall Threat Model is then defined as the set of all such triples relevant to the system, with each component selected from the predefined entries of the Threat Catalogue[11]. This structured representation ensures a consistent and reproducible mapping between adversaries, vulnerable assets, and their possible attack behaviors, thereby providing a solid foundation for subsequent risk analysis and penetration testing activities. The selection of threats associated with a specific asset in the SuA is determined by multiple factors. These include the asset's properties (i.e., Asset Type), the communication protocols it relies upon (e.g., HTTP), and any indirect threats inferred from the $Compromised$ field, which captures dependencies between assets. This multi-factor evaluation allows the methodology to account not only for direct threats but also for secondary threats that emerge from the compromise of related assets. The entire process of constructing the Threat Model-following the steps described above-is delegated to the *Threat Model Generator*, which automatically derives and assembles the fine-grained threat triples from the system model. A detailed description of the selection criteria and their algorithmic implementation can be found in [27], an IoT-specific example is provided in [26].

The *Risk Analysis* phase prioritizes the selected threats by estimating their risk according to the OWASP Risk Rating Methodology[12]. The entire quantitative evaluation process is carried out by the dedicated component, the *Risk Analysis Calculator*. Since a detailed description of this automated procedure lies outside the scope of this paper, we refer the interested reader to [24,33].

The fourth phase of the methodology is devoted to the generation of the *Penetration Test Plan*, which formalizes the set of security tests to be executed against the SuA. The Penetration Test Plan is conceived as a structured list of attacks, where each entry is defined by a quadruple $ATK = (Asset, Threat, Attack\ Pattern, Tool)$. In this representation, the Asset specifies the target system component as modeled in the MACM; the Threat denotes the specific malicious action or condition, selected from the Threat Model generated in the previous phase; the Attack Pattern describes the abstract procedure or tactic to exploit the threat, leveraging entries from the MITRE CAPEC[13] (Common Attack Pattern Enumeration and Classification) knowledge bases; and the Tool indicates the concrete software utility or framework that can execute the attack pattern in the given system context. The production of Penetration Test Plan is carried out by the *Penetration Test Plan Generator*, which relies on specific techniques that take into account tools and techniques applicability according to the system model[14]. We invite the interested reader to find the details and the techniques description in [23].

[10] https://github.com/VSecLab/AutomaticAttackPlanGenerator
[11] https://pennet.vseclab.it/catalogs/threat-catalog
[12] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology
[13] https://capec.mitre.org/
[14] https://pennet.vseclab.it/catalogs/tools

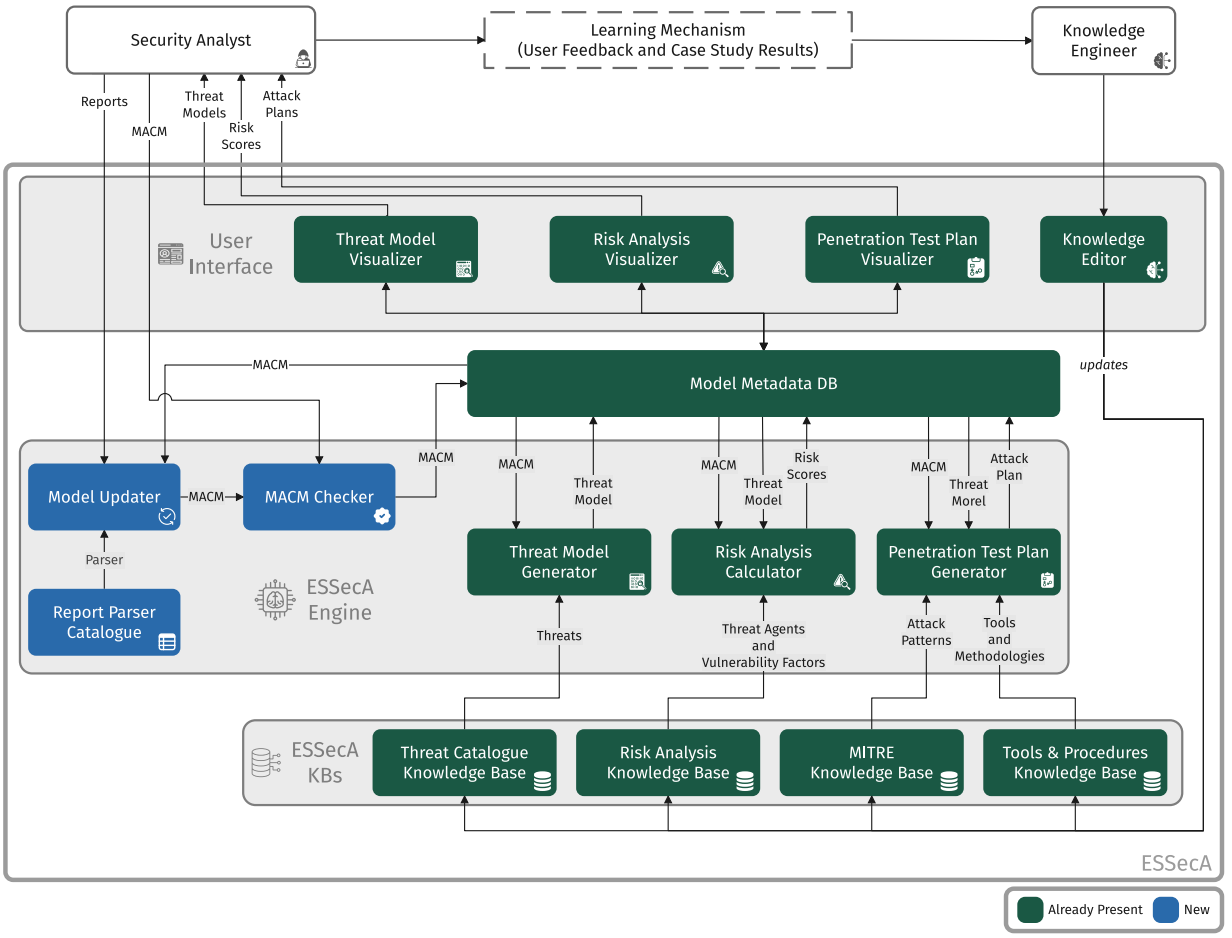

Fig. 6. ESSecA Architecture.

Once the Threat Model and the Penetration Test Plan have been produced, the penetration tester proceeds by selecting, through the *User Interface*, the specific asset to be targeted. At this point, the methodology offers two alternative execution paths: (i) *Threat-based path* - The tester manually navigates the Penetration Test Plan to identify the most relevant threat for the chosen asset. For that threat, the tester then selects an appropriate attack pattern and a compatible tool, effectively choosing one quadruple ATK from the plan. This quadruple fully describes a single, concrete attack to be executed against the system. (ii) *Methodology-based path* - For certain asset types with specialized characteristics (e.g., firmware or industrial control components), the framework bypasses the need for threat selection and directly proposes established, domain-specific testing methodologies.

These methodologies are drawn from recognized security standards or best practices and consist of a set of predefined actions or test procedures. The tester can then review these suggestions and select the specific actions to perform in the current cycle. By supporting both paths, the methodology allows the tester to adapt the penetration testing process to the context: following a fine-grained, threat-driven approach when applicable, or leveraging comprehensive, structured methodologies when a broader or more specialized analysis is required.

Currently, attack selection and pentest progression remain under tester control, as Penetration Test Plans are organized by phases but do not prescribe intra-phase sequencing. Thus, the framework enhances rather than replaces human expertise, offering decision support and automation.

6.3. Iterative ESSecA

The methodology described so far concludes after the Penetration Test Plan is created, leaving the tester to implement the suggested tests. Since penetration testing often employs a grey-box approach, where testers have limited visibility of the system architecture, the ESSecA methodology benefits from the introduction of the iterative modeling approach presented above. This approach defines a **cyclic process** in which the model is progressively enriched with evidence gathered during Threat Modeling and Penetration Testing. Each update increases the *completeness* of the system model, thereby enhancing the precision of both the Threat Model and the Penetration Test Plan as they evolve in parallel. Intuitively, if the initial model lacks some assets actually present in the target

system, the corresponding threats and penetration tests will not be considered by the ESSecA framework. Conversely, as the model is refined with additional or more accurate information, the Threat Model and the Penetration Test Plan become increasingly faithful to the real system, improving the overall effectiveness of the assessment. The complete workflow is illustrated in Fig. 7.

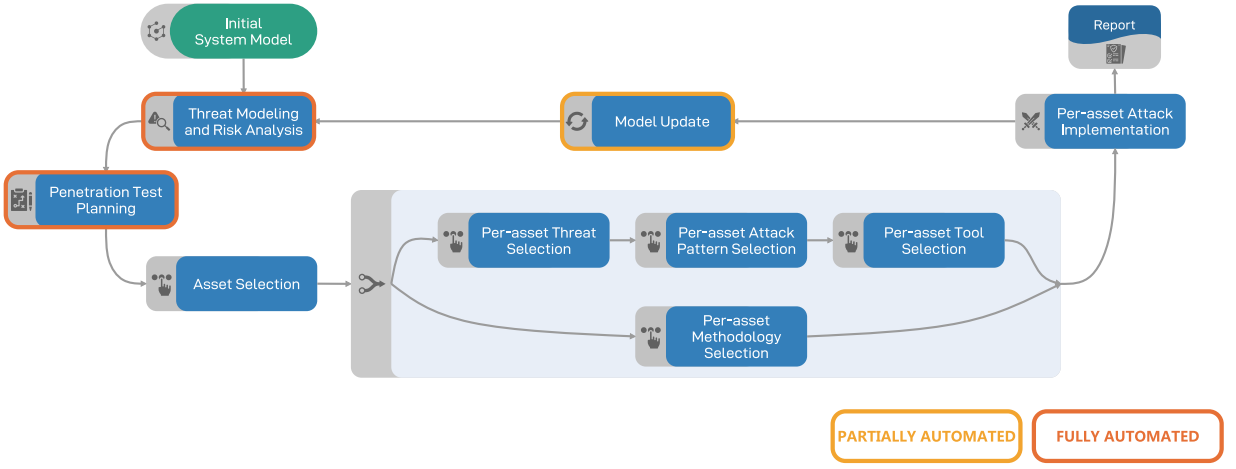

Fig. 7. Cyclic Methodology.

For instance, Nmap host discovery could reveal additional devices, adding new nodes and services to the model. These updates are then applied through ad hoc **productions**, as described in Section 5.3. The flowchart of the process is shown in the Fig. 8.

The appropriate production is selected based on the tool used for the attack and the nature of the collected evidence. Each entry in the Tool Catalogue is linked to a dedicated **parser** that translates tool outputs into model updates according to its production rule. All parsers implementing these *productions* are stored in the **Report Parser Catalogue** (Fig. 6). These productions ensure that the generated graphs are *correct* under the formalization introduced in Section 4, preserving syntactic and semantic consistency with the MACM schema. Since automated updates may still introduce anomalies, validation triggers (Section 5.2) are executed to detect and resolve inconsistencies.

Referring to the ESSecA architecture (Fig. 6), the reader can see that, when a test produces a report: (i) the pentester uploads it via the UI; (ii) the *Model Updater* checks for a corresponding parser; (iii) if present, the *Model Updater* uses the parser to extract the evidence and generates a Cypher query with the required model updates; (iv) the updated MACM model is verified through the *MACM Checker*; (v) if the verification succeeds, the proposed updates are shown to the pentester; (vi) upon confirmation, they are applied and stored in the database; (vii) the Threat Model and the Penetration Test Plan are then automatically updated to reflect the new state of the MACM model.

This simple but substantial improvement will be more evident with the application of some examples of algorithms that can be employed to update the MACM model, which will be presented in Section 7.

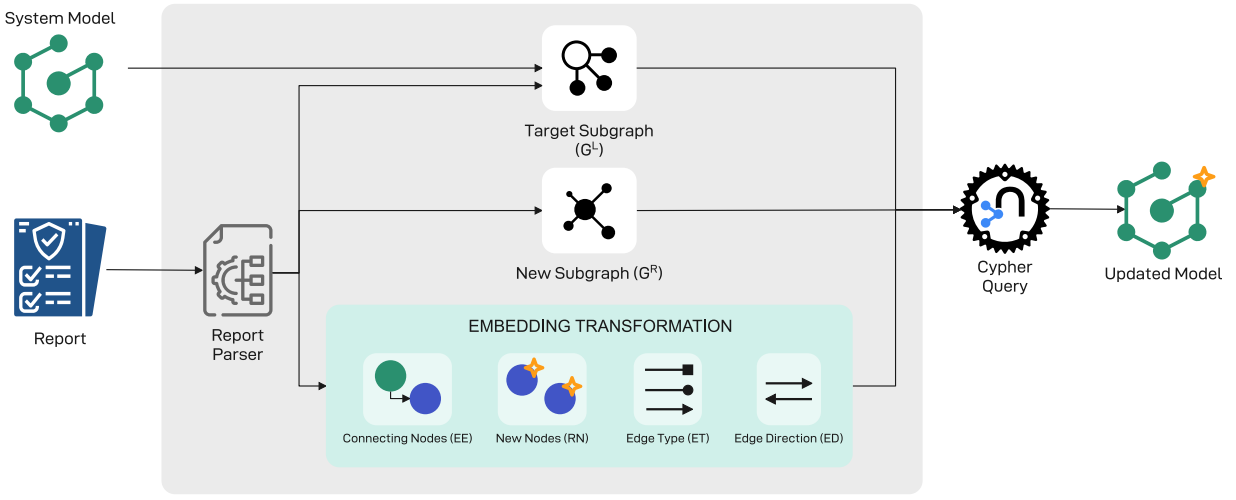

Fig. 8. Model Update Process.

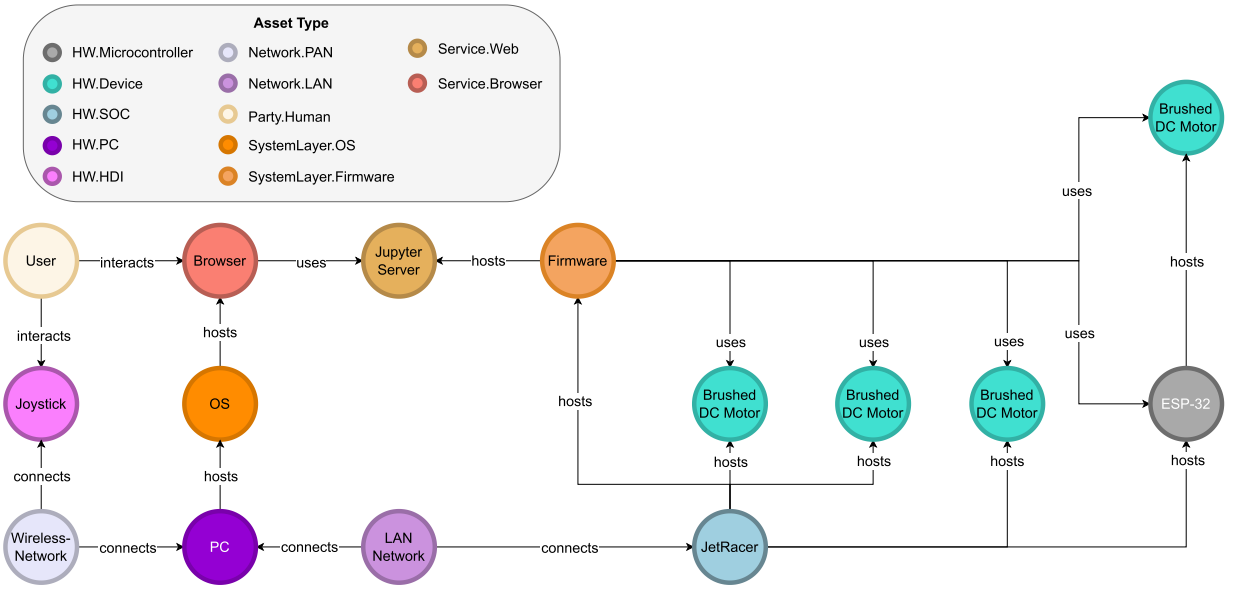

Fig. 9. Updated System Model after Cycle 1.

7. JetRacer: An IoT case study

This case study examines a modified version of the JetRacer, an autonomous mini-vehicle used for robotics and machine learning education that enables driving using AI. The vehicle was customized with a speed sensor and a chassis-mounted microcontroller, enabling speed measurement. Powered by the Jetson Nano (i.e. a compact AI computing device used for object identification and navigation), the JetRacer connects to a LAN network. A PC connected on the LAN can use a web service built with the Jupyter Lab API (hosted on the vehicle), in order to control JetRacer or manually pilot it via joystick.

7.1. Initial system model

We modeled a MACM that incorporates all components of the JetRacer. According to its specifications and documentation, the JetRacer (`HW.SOC`) integrates a set of sensors and a microcontroller. Considering the documentation, a firmware (modeled with asset type `SystemLayer.Firmware`) is hosted by the Jetson Nano. Another network is also modeled to represent the wireless connection between user's PC and the controller. This USB link establishes a Personal Area Network (i.e. `Network.PAN`) between the two devices. For correct operation, the joystick must be connected to the PC via USB, and the PC must share the same LAN as the JetRacer and its web service. A user sends commands from a PC browser, which are then relayed to the JetRacer's functionalities. This process is managed by the underlying operating system (represented as a `SystemLayer.OS` asset) and facilitated by a Jupyter server running on the JetRacer's firmware.

The rubber wheels of the racing robot are powered by two distinct motors: a DC brushed motor for propulsion and a servo motor for accurate steering control. Both are labeled as `HW.Device`, indicating components that the firmware employs to perform physical functions. As outlined earlier, the first MACM is shown in Fig. 9. In this model, each node's color corresponds to the asset type indicated in the legend.

This MACM has been generated and stored within the Neo4j database. As a result, it inherently benefits from Neo4j's underlying data structure, ensuring that all the constraints described in Section 4 are automatically verified during its creation and storage.

7.2. Cycle 1: Firmware analysis

After the model is built, our tool automatically identifies all relevant threats for each node by using its properties (i.e. the asset type). Table 5 reports only a selected subset.

For instance, a threat agent could compromise the JetRacer firmware, either by extracting data through firmware analysis or by altering it through the injection of malicious code. Other threats target the network layer, such as *Topology Disclosure*, where an attacker leverages network access to map the various nodes and their interconnections. Each identified threat is categorized using the STRIDE model [34], with the specific category represented by its initial letter.

Using this list as input, the tool generated a more complex Penetration Test Plan. Following our methodology, the number of identified attacks is high; however, Table 6 presents only a subset, corresponding to those we implemented.

Table 5
Subset of Threat Model related to the first MACM.

Asset	Threat	Description	STRIDE
JetRacer	Scanning	An adversary scans to understand the network configuration, mostly focusing on public information.	I
OS	Poisoning	Corruptibility of communication caches and the support data structure, such as routing or naming tables.	S,T
OS	Hijacking	It refers to the unauthorized takeover of systems, sessions, or communications, enabling attackers to gain control	S,T
Wifi Network	Topology Disclosure	An attacker can exploit forwarding updates to learn the network topology.	I
Wifi Network	Eavesdropping	An adversary retrieves valuable data from transmitted messages.	I
Firmware	Firmware Data Leakage	Through firmware vulnerabilities sensitive data leaks to unauthorized parties, leading to severe privacy and security breaches.	I

Table 6
Subset of Penetration Test Plan.

Asset	CAPEC ID	CAPEC Attack Pattern	Purpose	Tool	PenTest Phase
Firmware	574	Services Footprinting	Discovery of service types running on target firmware	Nmap	Discovery
Firmware	497	File Discovery	Discovery of files and their structures in target firmware	Binwalk, Firmwalker	Discovery
Wifi network	117	Interception	Interception of LAN packets	Wireshark	Information Gathering
Wifi network	292	Host Discovery	Discovery of active hosts in the LAN	Nmap	Discovery
OS	141	Cache Poisoning	Poisoning of the cache technologies	Ettercap	Execution
OS	593	Session Hijacking	Stealing an active session	Browser	Execution

We analyzed the firmware using the *OWASP Firmware Security Testing Methodology*[15], which recommends using *Binwalk*[16], a widely used tool for inspecting firmware images. Binwalk identifies embedded files, filesystems, and executable code, facilitating the discovery of services, configurations, and other key components within the firmware.

Through this analysis (further details in [20]), we identified two installed applications: (i) the OpenSSH service, confirmed by the configuration file at */etc/ssh/sshd_config* within *ext-root*; and (ii) the Jupyter Notebook API, indicated by the configuration file at */home/jetson/.jupyter* within *ext-root*. The model was coherently updated through manual queries (automatically verified in their correctness through the above-described approach).

7.3. Cycle 2: Services discovery

Employing our threat-based approach once more within the methodology can yield comparable insights. Using the Threat Model presented in Table 5 and the Penetration Test Plan generated for the initial MACM in Table 6, our tool identifies several potential attacks in the *reconnaissance* penetration testing phase. For instance, when selecting the *JetRacer* asset and the *Scanning* threat, the Penetration Test Plan in Table 6 recommends using the *Nmap* tool to identify open ports. This procedure reveals the services operating within the firmware, which may serve as possible entry points for further exploitation. The activity can be performed using the following command:

```
nmap -sV -p- -oX scan_results.xml -T4 --min-parallelism 100 <target_ip>
```

As described in the MACM update mechanism (Section 5.3), the results of the service scan are now integrated into the model by leveraging a specific *production* (detailed previously in Section 5.3). By using the discovery tool, some services are discovered, including a Jupyter (Werkzeug) service on port 8000, a Tornado web service on port 8888, an rpcbind on port 111, and an SSH server on the default port 22. In our case, the production *Add discovered services to host* is applied. Since the MACM is stored in Neo4j, we rely on Cypher queries to express the embedding expressions.

- G^L is a graph consisting of a single *host* node, already present in the MACM and identified by `id` = *host_id*.
- G^R is a graph consisting of one or more `Service` nodes, each representing a discovered port and enriched with derived attributes (e.g., `name` = *product*).
- Embedding fragments: one for each discovered service, whose only condition is that the *host* node exists in G^L. Accordingly, the embedding transformation is structured as follows:
  - *EE* (Embedding Expression): `MATCH (host {id: host_id})`
  - *RN*: node `(new_service:Service {id:new_id, name:"new_name", type:"Service"})`
  - *ET*: edge `-[:hosts]-`
  - $ED = 1$

Thus, each fragment generates a subgraph as follows.

[15] https://github.com/scriptingxss/owasp-fstm
[16] https://github.com/ReFirmLabs/binwalk

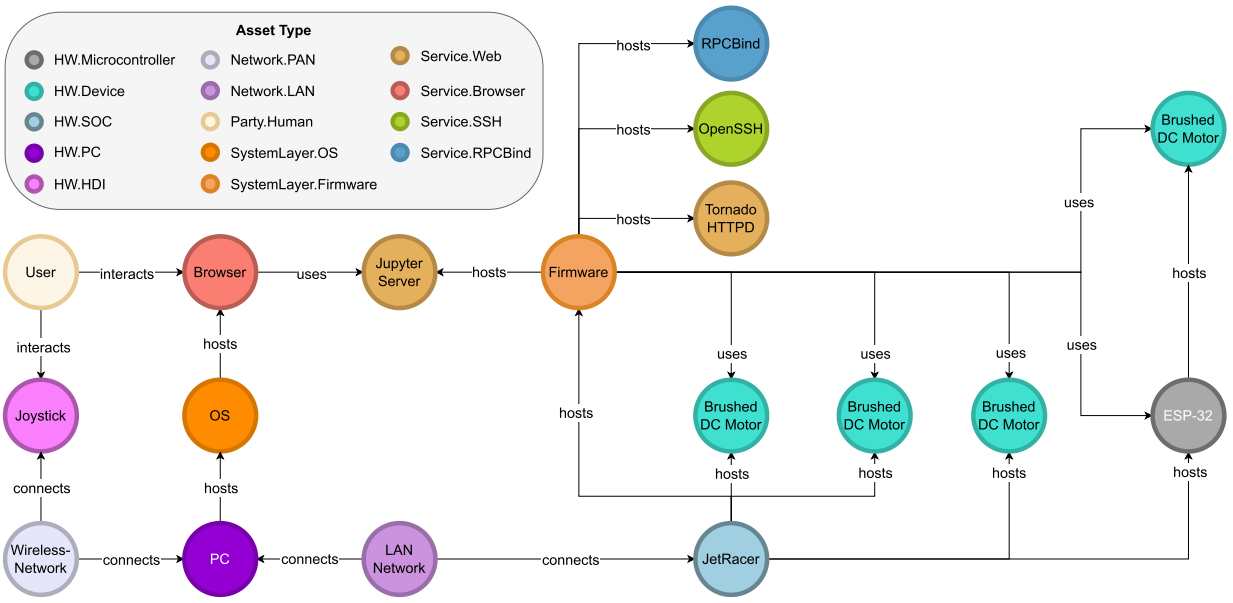

Fig. 10. Updated System Model after Cycle 2.

```
(host {id: host_id}) -[:hosts]-> (new_service:Service {id: new_id, name: "
    new_name", type: "Service"})
```

For example, in this cycle, the Cypher query for the insertion of the newly discovered service (i.e. Jupyter) is the following:

```
CREATE (service15:Service:Web {id:'15', name:'Jupyter Server',
    type:'Service.Web'})
    WITH service15
    MATCH (host {id:'9'})
    MERGE (service17)<-[:hosts]-(host)
```

The selection of the appropriate production rule in each scenario is determined by the tool employed to execute the attack, as described above. The result of the cycle is the new MACM that contains all the discovered services, as shown in Fig. 10.

7.4. Cycle 3: ARP poisoning

In this cycle, the *Operating System* asset can be selected, filtering the Threat Model and Penetration Test Plan from Table 5 and Table 6. Considering the *Poisoning* threat affecting the victim's OS, the Penetration Test Plan recommends using *Ettercap* to carry out *Cache Poisoning*, as suggested by attack pattern 141 of CAPEC. In our model, the *JetRacer* has a LAN IP shown on the display and its MAC address obtainable by sniffing any network traffic. To attack the system, a further device must be connected to the LAN network.

Using *Ettercap*, the attacker can perform an ARP Poisoning attack. The tool maps IP addresses to physical devices by sending ARP Request messages, then the attacker selects the victim (Target 1) and the JetRacer (Target 2). During the attack, ARP Reply messages are sent to make each device believe the attacker is the other party, inserting the victim's MAC in replies to the JetRacer and the JetRacer's MAC in replies to the victim.

Another way to execute the attack is to use the following command line.

```
ettercap -Tq -i eth0 -M arp:remote -S <victim_ip> <jetracer_ip>
```

7.5. Cycle 4: Session token sniffing

The ARP Poisoning attack's execution did not trigger an update in the MACM, suggesting that no new threats or Penetration Test Plans were introduced. During Cycle 4, the previously defined penetration tests are executed, specifically targeting the Wi-Fi network asset. The Interception Attack Pattern is employed to capture the session ID token.

After a successful ARP poisoning attack, the adversary can intercept packets containing the victim's session ID, which is transmitted with every request to the web service. This session cookie-stored in the victim's browser-is exchanged between the victim and the JetRacer. By analyzing HTTP traffic with tools such as Wireshark (as described in the Penetration Test Plan in Table 6), the attacker can capture the session cookie and leverage it to perform a session hijacking attack.

It is important to highlight that this disclosure of the session ID is feasible due to the unencrypted nature of the communication and the lack of authentication in the ARP protocol.

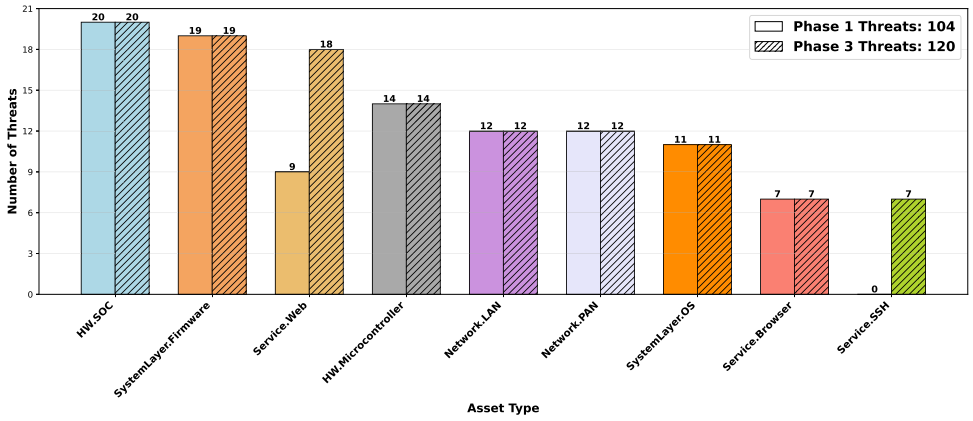

Fig. 11. Threats comparison between Phase 1 and Phase 3.

Table 7
Summary of the performed Penetration Test Cycles.

Cycle	Attacked Asset	Pentest Plan / Methodology	Tool	Attack Phase
1	JetRacer (Firmware)	OWASP Firmware Security Testing Methodology	binwalk, firmwalker	Discovery
2	JetRacer (Firmware)	Service Footprinting	Nmap	Discovery
3	OS, Wi-Fi Network	Cache Poisoning	Ettercap	Execution
4	Wi-Fi Network	Interception	Wireshark	Information Gathering
5	OS	Session Hijacking	Browser	Execution

7.6. Cycle 5: HTTP session hijacking

As expected in Cycle 5, the attack is carried out by targeting the `SystemLayer.OS` assets, applying the *Session Hijacking* attack pattern, and using a standard web browser. After obtaining the session cookie-usually beginning with *username*-the attacker can substitute its value in the browser with the stolen one, thereby taking control of the victim's session. It is important to note that, as in previous cycles, the attack generates a report detailing the exploited threats and their consequences, but does not trigger any update to the MACM.

7.7. Findings

The cyclical approach demonstrates significant improvement in model completeness and security assessment coverage. As illustrated in Fig. 11 and Fig. 12, the iterative refinement process reveals a substantial increase in identified threats and attacks between Phase 1 (initial model) and Phase 3 (refined model after discovery cycles).

Fig. 11 presents the threat evolution. The threat count increased from 104 in Phase 1 to 120 in Phase 3 (+15.4 %). The most significant growth is observed in the *Service.Web* asset type, where threats grew from 9 to 18, and in the newly discovered *Service.SSH*, which accounts for 7 threats. Other asset types remained relatively stable: *Network* maintained 12 threats, *SystemLayer.OS* maintained 11 threats, and *SystemLayer.Firmware* maintained 19 threats across both phases. The stability of these threat counts indicates that the initial model adequately captured the core system components.

Fig. 12 shows the evolution of the attack surface. The total number of attacks increased from 192 in Phase 1 to 279 in Phase 3 (+45.3 %), with the most significant growth observed in the *Service.Web* asset type (from 77 to 154 attacks). This substantial increase results from the discovery of additional web services (Jupyter and Tornado servers) during Cycle 2. The newly discovered *Service.SSH* contributes 10 attacks, further expanding the attack surface. The *Network* asset consistently accounts for a high number of attacks (76 in both phases), reflecting the prevalence of network-based penetration testing tools in the catalog (e.g., *Wireshark*, *Nmap*).

These results demonstrate that iterative model refinement is essential for comprehensive security assessment. The discovery of a few web services and one SSH service led to a 45 % increase in the overall the number of possible attacks, with web attacks doubling (+100 %) and some new SSH-related attacks identified supported by specific tools (e.g. Hydra). This emphasizes the critical importance of complete system modeling and discovery through cyclical penetration testing, as even a small number of overlooked services can dramatically expand security exposure.

Taking into account the general results described before, some attacks have been successfully carried out, as summarized in Table 7. The MACM models for each cycle, the complete Threat Model and Penetration Test Plan, as well as the Nmap output for the attack executed in Cycle 2, are publicly available on GitHub[17].

[17] https://github.com/VSecLab/OpenData/tree/main/JetRacer-IoT_Journal_2025

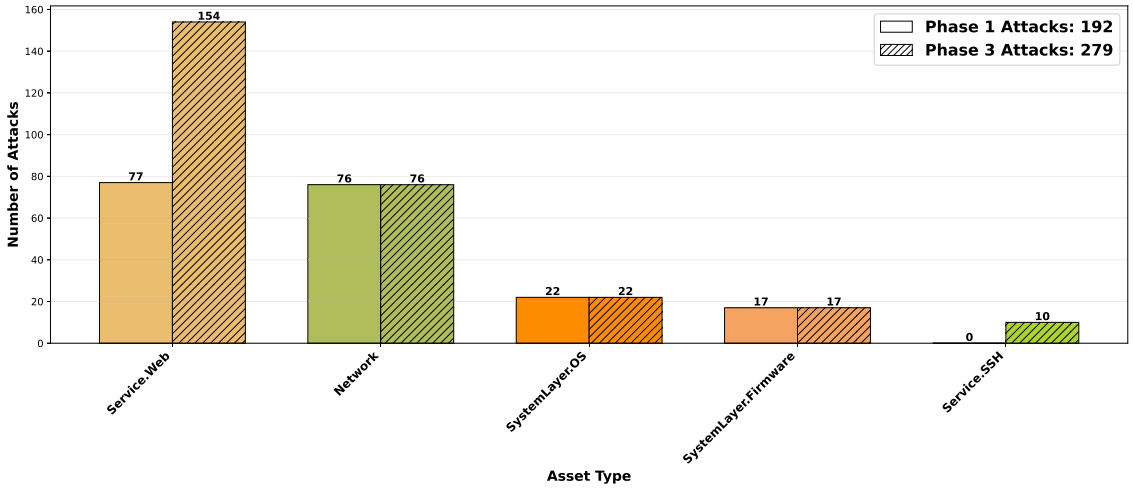

Fig. 12. Attacks comparison between Phase 1 and Phase 3.

8. Discussion, limits, and conclusion

This section unifies the discussion of correctness, completeness, and validity with the methodological limits of the approach, and concludes by summarizing the contributions and future directions of the work.

Correctness and domain realizability (RQ1)

The first point concerns the extent to which the modeling technique can guarantee that the models it generates are realizable within the domain. In open and heterogeneous domains, absolute correctness is not attainable: the domain is only partially known and cannot be exhaustively characterized. This limitation is not a formal impossibility result but rather a knowledge-bound constraint: the completeness of the domain and the space of admissible systems cannot be fully known.

MACM does not aim to overcome this fundamental limitation. Instead, it provides a notion of *relative correctness*, defined with respect to the classes of unrealizable systems captured by the modeling constraints. These constraints-expressed as syntactic and semantic invariants over the property-graph schema-implement the anti-patterns defined in Table A.8 and systematically exclude logically or physically impossible system configurations. Thus, correctness is ensured only within the scope of these assumptions and should not be interpreted as a maximal or exhaustive guarantee.

Beyond exclusion, the constraints serve a constructive function. By rejecting updates that violate structural or semantic assumptions, the system guides the analyst toward well-formed representations and highlights omissions or inconsistencies. Constraints therefore act both as *guards* and as *diagnostic indicators*, supporting the refinement of the model and contributing indirectly to completeness by preventing the introduction of structures that obscure missing information.

Completeness and validity through iterative refinement (RQ2)

The second research question asks whether one can define a process that guarantees both completeness and validity of the model for security assessment. In our approach, this is achieved through the combined action of MACM and ESSecA: since completeness and validity cannot be ensured by static verification alone – being tied to information that emerges only through observation of the real system – then the cyclic refinement mechanism operationalizes their improvement. At each iteration, penetration-testing results reveal missing or inaccurate elements, which are incorporated into the model. This allows the model to evolve toward increasing completeness and validity, while MACM's formal constraints ensure correctness is preserved at every step.

This improvement in model completeness is not an end in itself. In a model-based security assessment such as ours, the model is the foundation upon which the threat model and the penetration test plan are produced. A more complete and more accurate model therefore has direct and significant impact on the completeness and validity of both the threat model and the test plan: missing components lead to missing threats, and inaccurate structures lead to incorrect attack paths or invalid tool selections.

The use of dedicated productions that map tool outputs to model updates ensures that these refinements preserve correctness by construction. As a result, the model becomes progressively more complete-by adding missing components-and more valid-by updating or removing inaccurate ones-while correctness is maintained at every step, ultimately strengthening all downstream assessment activities.

However, iterative refinement yields improvements that are monotonic but not measurable. No upper bound can be placed on completeness or validity, and undocumented or inconsistent system behaviors cannot be recovered through modeling alone. These limitations reflect intrinsic constraints of security modeling in complex, dynamic, partially observable environments.

Expressiveness, extensibility, and methodological limits

The extensibility of the property-graph formalism ensures that the technique remains expressive across heterogeneous domains. While global completeness is unattainable, MACM achieves domain-relative completeness through a catalog-driven and systematically extensible schema capable of incorporating new asset types and technologies. This extensibility was validated empirically in the case studies, where all relevant system components were representable without violating schema constraints.

Overall, the technique strikes a balance between formal rigor and practical applicability. Constraint-based correctness, automated model verification, and correctness-preserving empirical refinement provide strong guarantees within the limits of what is theoretically achievable. At the same time, the method remains subject to inherent limitations related to domain incompleteness, partial observability, and evolving system behavior. These are not flaws of the technique but structural properties of the problem space, guiding its scope and applicability.

Summary of contributions and future work

This work presented a rigorous formalization of the Multi-purpose Application Composition Model (MACM), a property-graph-based formalism for modeling IT systems with security assessment in mind. By defining syntactic and semantic constraints and implementing validation via Cypher-based mechanisms, the approach ensures model correctness and reproducibility across the assessment workflow. The integration of MACM into the ESSecA methodology supports cyclical refinement based on penetration testing results, enabling progressive enhancement of Threat Modeling, Risk Assessment, and Penetration Test Planning.

Two case studies demonstrated the applicability of the approach: the eWeLink IoT ecosystem demonstrated the modeling capabilities of MACM, while the JetRacer autonomous vehicle showcased the full cyclical workflow, from initial modeling to iterative updates driven by empirical findings. Across the cycles, the model evolved from 104 potential threats and 192 possible attacks to 120 threats and 279 possible attacks, with five real attacks executed, culminating in a successful session hijacking demonstration. These findings underscore the impact of iterative refinement on security exposure analysis.

As future work, we plan to extend MACM to capture runtime dynamics of distributed systems, including autoscaling, reconfiguration, and adaptive behaviors, and to further automate the feedback loop between testing tools and model updates. These developments will support continuous security assurance in increasingly complex and evolving environments.

Declaration of generative AI and AI-assisted technologies in the manuscript preparation process

During the preparation of this work the authors used ChatGPT in order to paraphrase some portion of text and to check the grammar of the manuscript. After using this tool/service, the authors reviewed and edited the content as needed and take full responsibility for the content of the published article.

CRediT authorship contribution statement

Felice Moretta: Conceptualization, Data curation, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Software, Validation, Visualization, Writing – original draft, Writing – review & editing; **Umberto Barbato:** Conceptualization, Data curation, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Software, Validation, Visualization, Writing – original draft, Writing – review & editing; **Massimiliano Rak:** Conceptualization, Data curation, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Supervision, Validation, Visualization, Writing – original draft, Writing – review & editing; **Daniele Granata:** Conceptualization, Data curation, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Software, Validation, Visualization, Writing – original draft, Writing – review & editing.

Declaration of competing interest

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

Acknowledgements

This work was partially supported by the DEFEDGE project (E53D23016380001) under the PRIN program, and the SecCo-OC project within the SERICS framework (D33C22001300002), both funded by italian MUR.

Appendix A. Anti-Patterns

Table A.8
Anti-Pattern List.

ID	Anti-Pattern	Modeling Feature	Category
AP1	The modeling system must not allow describing a scenario in which a Party executes software or performs digital operations, unless this happens through a Human/Device interface.	**RP1**	Party
AP2	The modeling system must not allow describing a scenario in which a service lacks a deployment context.	**RP5, SemC4, SemC7**	Service
AP3	The modeling system must not allow describing a scenario in which a service establishes physical connections without passing through the infrastructure that hosts it (OS, network, hypervisor).	**SemC4**	Service
AP4	The modeling system must not allow describing a scenario in which a service hosts devices, systems, or other services as if it were an execution environment.	**RP3**	Service
AP5	The modeling system must not allow describing a scenario in which a service directly uses people, hardware, networks, systems, virtual components, or a Cloud Service Provider.	**RP2**	Service
AP6	The modeling system must not allow describing a scenario in which a service provides components or resources, since it is not an infrastructure element.	**RP2, RP3**	Service
AP7	The modeling system must not allow describing a scenario in which hardware directly executes application services (it can only execute firmware or operating systems).	**RP7, SemC9**	Hardware
AP8	The modeling system must not allow describing a scenario in which hardware is hosted by components other than other hardware.	**RP3, RP4, RP5**	Hardware
AP9	The modeling system must not allow describing a scenario in which hardware directly hosts virtual machines or containers: a virtualization layer is required.	**RP7**	Hardware
AP10	The modeling system must not allow describing a scenario in which hardware provides components or resources, since it does not represent a service-delivery entity.	**RP7**	Hardware
AP11	The modeling system must not allow describing a scenario in which a network provides services or hosts systems.	**RP9**	Network
AP12	The modeling system must not allow describing a scenario in which a network connects to elements without a network interface or to assets that are neither networks nor valid endpoints.	**RP9**	Network
AP13	The modeling system must not allow describing a scenario in which a network uses other components, since it is a passive communication channel.	**RP9**	Network
AP14	The modeling system must not allow describing a scenario in which a virtual machine is independent of a hypervisor or hardware, or is provided directly by a Cloud Service Provider.	**RP5, RP8, SemC3**	Virtualization
AP15	The modeling system must not allow describing a scenario in which a virtual machine or a virtualized component uses other assets, since it always operates through a hypervisor and lower layers.	**RP4**	Virtualization
AP16	The modeling system must not allow describing a scenario in which a virtual machine or a virtualized component provides other assets, since it is not an infrastructure service-delivery layer.	**RP4**	Virtualization
AP17	The modeling system must not allow describing a scenario in which an hypervisor directly hosts an operating system or firmware, since these must always be deployed over appropriate virtualization layers.	**RP5, SemC8**	Virtualization
AP18	The modeling system must not allow describing a scenario in which a container exists without a container runtime hosting it, or is provided directly by a Cloud Service Provider.	**RP5, SemC3**	Containerization
AP19	The modeling system must not allow describing a scenario in which a containerization system is hosted directly on hardware: an operating system is required.	**RP7, SemC9**	Containerization
AP20	The modeling system must not allow describing a scenario in which a container runtime directly hosts an operating system or firmware, since these must always be deployed over appropriate containerization layers.	**RP5, SemC8**	Containerization
AP21	The modeling system must not allow describing a scenario in which operating systems or firmware lack a deployment context.	**RP7, RP8, SemC3**	OS/-Firmware
AP22	The modeling system must not allow describing a scenario in which an operating system or firmware directly uses services, networks, other systems, virtual assets, or a Cloud Service Provider.	**RP6**	OS/-Firmware
AP23	The modeling system must not allow describing a scenario in which an operating system or firmware interacts directly with other assets without passing through appropriate services or interfaces.	**RP5,RP6**	OS/-Firmware
AP24	The modeling system must not allow describing a scenario in which an operating system hosts virtual machines or containers, since such components require a dedicated virtualization or containerization layer.	**RP5, SemC6**	OS/-Firmware
AP25	The modeling system must not allow describing a scenario in which an operating system or firmware directly hosts other operating systems or firmware. It requires a dedicated virtualization or containerization layer.	**RP5, SemC5**	OS/-Firmware
AP26	The modeling system must not allow describing a scenario in which a Cloud Service Provider uses system assets or establishes direct connections with other infrastructure components.	**RP8**	Cloud Service Provider
AP27	The modeling system must not allow nodes whose Asset Type is inconsistent with the assigned Primary and Secondary Labels.	**SemC1**	General
AP28	The modeling system must not allow describing a scenario in which an asset is hosted or provided by multiple components.	**SemC2**	General
AP29	The modeling system must not allow describing a scenario in which subsystems are not connected by any relation. In such a case, these represent independent systems that must be modeled separately.	**SemC11**	General
AP30	The modeling system must not allow describing a scenario in which cycles are created in hosting relations. Indeed, it is impossible for a chain of mutually hosting components to return to the starting point, such as: hardware HW1 hosts HW2, HW2 hosts HW3, and HW3 hosts HW1 again. Such a structure cannot occur in real systems, where hosting relations are always hierarchical and acyclic.	**SemC12**	General
AP31	The modeling system must not allow inserting protocols into relations that do not represent network communications.	**SemC10**	General

References

[1] Microsoft Corporation, Microsoft Threat Modeling Tool, 2025, (https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool), Microsoft Learn, accessed July 29, 2025.
[2] A. Josey, M. Lankhorst, I. Band, H. Jonkers, D. Quartel, et al., An Introduction to the ArchiMate® 3.0 Specification, White Paper from The Open Group (2016) 35.
[3] A. Ellerm, M.E. Morales-Trujillo, Modelling Security Aspects with ArchiMate: A Systematic Mapping Study, in: 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), IEEE, Portoroz, Slovenia, 2020, pp. 577–584. https://doi.org/10.1109/SEAA51224.2020.00094
[4] J. Jurjens, UMLsec: Extending UML for Secure Systems Development, in: J.-M. Jezequel, H. Hussmann, S. Cook (Eds.), "UML" 2002 - The Unified Modeling Language, 2460, Springer Berlin Heidelberg, pp. 412–425. https://doi.org/10.1007/3-540-45800-X_32
[5] T. Lodderstedt, D. Basin, J. Doser, SecureUML: A UML-Based Modeling Language for Model-Driven Security, in: Lecture Notes in Computer Science, Springer Berlin Heidelberg, pp. 426–441. https://doi.org/10.1007/3-540-45800-x_33
[6] L. Apvrille, Y. Roudier, SysML-Sec - A Model Driven Approach for Designing Safe and Secure Systems, in: Proceedings of the 3rd International Conference on Model-Driven Engineering and Software Development, SCITEPRESS - Science and and Technology Publications, pp. 655–664. https://doi.org/10.5220/0005402006550664
[7] D. Gluch, AADL Security Annex [Draft], Technical Report, 2019.
[8] Y. Cao, Y. Dong, X. Wei, X. Wu, et al., AADL Vulnerability Modeling and Security Analysis Method, in: 2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C), IEEE, Sofia, Bulgaria, 2019, pp. 399–406. https://doi.org/10.1109/QRS-C.2019.00080
[9] P. Johnson, L. Robert, M. Ekstedt, et al., A Meta Language for Threat Modeling and Attack Simulations, 2018. https://doi.org/10.1145/3230833.3232799
[10] L. Cerovic, StrideLang: Creation of a Domain-Specific Threat Modeling Language Using STRIDE, DREAD and MAL .
[11] S. Katsikeas, P. Johnsson, S. Hacks, R. Lagerström, et al., VehicleLang: A Probabilistic Modeling and Simulation Language for Modern Vehicle IT Infrastructures, Comput. Secur. 117 (2022) 102705. https://doi.org/10.1016/j.cose.2022.102705
[12] M. Kordi, F. Mariotti, P. Lollini, A. Bondavalli, A. Ceccarelli, A. Bondavalli, M. Trapp, E. Schoitsch, B. Gallina, F. Bitsch, et al., Security Modeling Challenges and Research Directions Around the ADVISE Meta Framework, 14989 LNCS, Springer Science and Business Media Deutschland GmbH, pp. 275–283. https://doi.org/10.1007/978-3-031-68738-9_21
[13] E. LeMay, M.D. Ford, K. Keefe, W.H. Sanders, C. Muehrcke, Model-Based Security Metrics Using ADversary VIew Security Evaluation (ADVISE), 6042046, pp. 191–200. https://doi.org/10.1109/QEST.2011.34
[14] R. Angles, The Property Graph Database Model, https://renzoangles.net/wp/wp-content/uploads/2018/05/amw2018.pdf.
[15] R. Angles, C. Gutierrez, Survey of Graph Database Models 40 (1) 1–39. https://doi.org/10.1145/1322432.1322433
[16] L. Bellomarini, A. Gentili, E. Laurenza, E. Sallinger, et al., Model-Independent Design of Knowledge Graphs, https://repositum.tuwien.at/handle/20.500.12708/193568. https://doi.org/10.34726/5426
[17] M. Elkhodr, S. Shahrestani, H. Cheung, The Internet of Things: New Interoperability, Management and Security Challenges, Int. J. Netw. Secur. Appl. 8 (2) (2016) 85-102. https://doi.org/10.5121/ijnsa.2016.8206
[18] J. Jurjens, UMLsec: Extending UML for Secure Systems Development, in: J.-M. Jezequel, H. Hussmann, S. Cook (Eds.), "UML" 2002 — The Unified Modeling Language, Springer Berlin Heidelberg, Berlin, Heidelberg, 2002, pp. 412–425.
[19] M.C. Ghanem, T.M. Chen, M.A. Ferrag, M.E. Kettouche, ESASCF: Expertise Extraction, Generalization and Reply Framework for Optimized Automation of Network Security Compliance, IEEE Access 11 (2023) 129840–129853. https://doi.org/10.1109/ACCESS.2023.3332834
[20] D. Granata, M. Rak, F. Moretta, F. Fiumara, et al., A Cyclical Penetration Testing Automation Methodology: The JetRacer Case Study, in: L. Barolli (Ed.), Advanced Information Networking and Applications, 246, Springer Nature Switzerland, pp. 340–353. https://link.springer.com/10.1007/978-3-031-87766-7_30. https://doi.org/10.1007/978-3-031-87766-7_30
[21] Terminology for model credibility, SIMULATION 32 (3) (1979) 103–104. https://doi.org/10.1177/003754977903200304
[22] R.G. Sargent, Verification and validation of simulation models, in: Proceedings of the 2010 winter simulation conference, IEEE, 2010, pp. 166–183.
[23] M. Rak, F. Moretta, D. Granata, Advancing ESSecA: a step forward in Automated Penetration Testing, in: Proceedings of the 19th International Conference on Availability, Reliability and Security, ARES '24, Association for Computing Machinery, New York, NY, USA, 2024. https://doi.org/10.1145/3664476.3670459. https://doi.org/10.1145/3664476.3670459
[24] D. Granata, M. Rak, G. Salzillo, Risk Analysis Automation Process in IT Security for Cloud Applications, in: D. Ferguson, M. Helfert, C. Pahl (Eds.), Cloud Computing and Services Science, Springer International Publishing, Cham, 2022, pp. 47–68.
[25] D. Granata, M. Rak, G. Salzillo, U. Barbato, et al., Security in IoT pairing & authentication protocols, a threat model and a case study analysis, in: CEUR WORKSHOP PROCEEDINGS, 2940, CEUR-WS, 2021, pp. 207–218.
[26] G. Salzillo, M. Rak, F. Moretta, Threat Modeling based Penetration Testing: The Open Energy Monitor Case study, in: 13th International Conference on Security of Information and Networks, SIN 2020, Association for Computing Machinery, New York, NY, USA, 2021. https://doi.org/10.1145/3433174.3433181. https://doi.org/10.1145/3433174.3433181
[27] M. Rak, G. Salzillo, D. Granata, ESSecA: An automated expert system for threat modelling and penetration testing for IoT ecosystems, Computers and Electrical Engineering 99 (2022) 107721. https://www.sciencedirect.com/science/article/pii/S0045790622000350. https://doi.org/10.1016/j.compeleceng.2022.107721
[28] K. Abdulghaffar, N. Elmrabit, M. Yousefi, Enhancing Web Application Security through Automated Penetration Testing with Multiple Vulnerability Scanners, Computers 12 (11) (2023). https://doi.org/10.3390/computers12110235
[29] A. Almohannadi, J.R. Nurse, T. Watson, A Survey of Automated Vulnerability Analysis and Exploitation Techniques on Web Applications, IEEE Access (2021).
[30] X. Zeng, Y. Li, Q. Wang, Integration of Multiple Web Vulnerability Scanners in a Single Framework: Enhancing Detection and Reducing False Positives, J. Inf. Secur. Appl. (2019).
[31] B. Abdulkhalek, A Systematic Literature Review on Internet of Vehicles Security Challenges and Penetration Testing Solutions (2024). https://doi.org/10.36227/techrxiv.172296707.71456018/v1
[32] Z. Wang, S. Li, L. Zhang, C. Hu, L. Yan, A Red Team automated testing modeling and online planning method for post-penetration, Comput. Secur. 144 (2024) 103945. https://www.sciencedirect.com/science/article/pii/S0167404824002505. https://doi.org/10.1016/j.cose.2024.103945
[33] V. Casola, A. De Benedictis, M. Rak, U. Villano, Toward the automation of threat modeling and risk assessment in IoT systems, IoT 7 (2019) 100056. https://www.sciencedirect.com/science/article/pii/S2542660519300290. https://doi.org/10.1016/j.iot.2019.100056
[34] M. Corporation, The STRIDE Threat Model, 2002, (https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-threats), Accessed: 2025-05-16.