The ISC Int'l Journal of Information Security

July 2026, Volume 18, Number 2 (pp. 15–33)

http://www.isecure-journal.org

SmartUcon: A Model-Driven Framework for Secure and Privacy-Aware Usage Control in Smart Environments

Mohammad Asmand Joneghani [1], and Leila Samimi-Dehkordi [1,*]

[1] *Department of Computer Engineering, Faculty of Technology and Engineering, Shahrekord University*

ARTICLE INFO.

Article history:
Received:
Revised:
Accepted:
Published Online:

Keywords:
Domain-Specific Modeling Language, IoT Security, Model-Driven Engineering, Smart Environments, Usage Control

Type:

doi:

Abstract

Ensuring secure and continuous access in smart environments requires dynamic and context-aware policy enforcement, which traditional access control models cannot adequately support. This paper presents SmartUcon, a model-driven domain-specific modeling language (DSML) based on the UCON+ model, designed for specifying and managing usage control policies in IoT-enabled and cyber-physical systems. SmartUcon provides a Sirius-based graphical editor for intuitive policy modeling, along with an EGL-based code generator that automatically transforms high-level models into executable Axiomatics Language for Authorization (ALFA) policies. The semantics of the DSML are defined translationally via explicit mapping rules to ALFA, ensuring semantic preservation. The framework further incorporates design-time conflict detection and model-time validation using the Epsilon Validation Language (EVL) to reduce policy misconfigurations. To evaluate its applicability, 14 case studies were developed across five domains—smart vehicles, smart homes, smart cities, healthcare, and industrial systems—while a user study with 32 participants and statistical significance tests confirm the usability advantages of graphical modeling. A structural comparison with related metamodels confirms its superior coverage of obligations, advice, and device interactions. Overall, SmartUcon offers a practical and extensible model-driven solution for secure and trustworthy policy enforcement in smart environments.

1 Introduction

With the rapid growth of smart environments and the proliferation of Internet of Things (IoT) and cyber-physical systems, the need for fine-grained, continuous, and context-aware access control has become increasingly critical. These environments often involve sensitive data exchanges and security-critical interactions, which require reliable mechanisms for information security and trustworthy policy enforcement [1]. Traditional access control models, such as Mandatory Access Control (MAC) [2], Discretionary Access Control (DAC) [3], and Role-Based Access Control (RBAC) [4], were primarily designed for static environments and lack the flexibility required to support dynamic and heterogeneous ecosystems. Attribute-Based Access Control (ABAC) [5] improved flexibility by leveraging user, resource, and environmental attributes, yet it still makes one-time decisions at access request and does not adequately support ongoing

* Corresponding author.
Email addresses: mohammad.asmand@stu.sku.ac.ir, samimi@sku.ac.ir

ISSN: 2008-2045

control or dynamic trust adaptation [6].

To address these limitations, the Usage Control (UCON) model was introduced, extending access control to enable continuous monitoring and dynamic enforcement of policies during the entire usage session [7]. UCON considers not only authorizations but also obligations and environmental conditions throughout the interaction, and it introduced the three phases of control: pre-authorization, ongoing-authorization, and post-authorization. However, even UCON remains insufficient for highly dynamic environments, such as smart homes, vehicles, and healthcare systems, where real-time reevaluation, contextual trust management, and policy adaptation are essential [8, 9].

UCON+ was proposed as an enhanced model to address these shortcomings by introducing hierarchical structures, separating decision-making components, and providing stronger support for the three control phases, along with the integration of additional mechanisms such as trust management and delegation [8]. Despite these advances, existing tools and frameworks in software engineering still provide limited support for expressing and enforcing complex usage control policies in smart environments. Many domain-specific languages (DSLs) and metamodels have been introduced, but most are rooted in traditional paradigms and fail to fully capture the requirements of IoT-enabled contexts, especially in terms of security assurance, policy consistency, and traceability [10].

In this paper, we present **SmartUcon**, a model-driven domain-specific modeling language (DSML) based on the UCON+ model. The goal of SmartUcon is to bridge the gap between high-level policy specification and executable enforcement in dynamic smart environments. Instead of requiring manual policy coding, the proposed approach enables policy designers to specify usage control policies at a higher level of abstraction, thereby improving consistency and reducing modeling complexity.

The contributions of this paper can be summarized as follows:

- We design a UCON+-based [8] metamodel that captures essential policy components, including conditions, obligations, advice, context, and device interactions.
- We implement a graphical modeling tool using the Sirius framework [11] that supports intuitive, visual construction of usage control policies.
- We develop an EGL-based transformation mechanism [12] that automatically generates code of Axiomatics Language for Authorization (ALFA) [13] from models, reducing manual coding effort.
- We evaluate SmartUcon through 14 case studies across diverse domains (e.g., smart homes, smart vehicles, healthcare, and industry) and compare it with existing metamodels in terms of structural complexity and coverage.
- We assess the usability of the graphical tool through a user study and questionnaire-based evaluation, providing empirical insights into its practicality, security assurance, and user acceptance.

Despite these contributions, a closer examination of existing UCON+ DSLs and related modeling approaches reveals several limitations that motivate this work. First, many existing solutions primarily focus on policy specification at an abstract level, without providing a fully integrated model-driven pipeline that connects modeling with executable enforcement. Second, support for dynamic smart environment concepts—such as sensors, contextual conditions, and event-driven adaptations—is often limited or only partially addressed. Third, several approaches rely on textual policy specification or partially automated transformations, which can introduce complexity and increase the likelihood of human errors, particularly for non-expert users.

In contrast, SmartUcon aims to address these gaps by combining model-driven engineering principles with explicit support for context-aware policy modeling and automated transformation toward executable policies. This integration facilitates a more practical and structured approach to policy engineering in smart environments.

The remainder of this paper is organized as follows. Section 2 provides the necessary background on model-driven engineering, domain-specific languages, and access/usage control models. Section 3 introduces a motivating example to illustrate the challenges of UCON+ in smart environments. Section 4 reviews related work in access and usage control modeling. Section 5 presents the proposed DSML, including its abstract syntax, concrete syntax, and semantics. Section 6 evaluates the applicability, usability, and quality of the proposed approach through case studies and structural comparison. Finally, Section 7 concludes the paper and outlines future research directions.

2 Background

Model-Driven Engineering: Model-Driven Engineering (MDE) is a software development paradigm that emphasizes the use of high-level models as primary artifacts rather than source code. Models abstract system functionality and constraints, and meta-

models define their structure and semantics. MDE promotes automation through model transformations, enabling tasks such as validation, analysis, and code generation. By shifting focus from low-level implementation to abstract modeling, MDE reduces complexity, increases productivity, and improves system quality [14, 15]. A key element in MDE is the model-to-code transformation process, where executable artifacts are automatically generated from models, ensuring consistency and maintainability across different platforms [12].

Domain-Specific Languages (DSLs): The DSL languages are specialized languages tailored to a particular application domain. Unlike general-purpose languages (GPLs), DSLs raise the level of abstraction by focusing on domain-specific concepts, enabling stakeholders to specify and analyze systems without being concerned with low-level implementation details [16]. DSLs are widely used in diverse areas such as embedded systems, financial systems, and industrial control, where precision and domain expressiveness are essential [17].

Domain-Specific Modeling Languages (DSMLs): A particular class of DSLs are DSMLs, which are designed within the MDE framework [18]. DSMLs are defined by a metamodel that formally captures the concepts, relationships, and constraints of a specific domain, and are often accompanied by a concrete syntax (graphical or textual) and transformation rules to generate executable artifacts. Compared to textual DSLs, DSMLs provide stronger integration with modeling tools, automated transformations, and validation against their metamodels. In the security domain, DSMLs enable policy designers to specify access and usage control policies at a higher level of abstraction, validate them against formal constraints, and automatically transform them into executable code or configuration files. This model-driven approach increases precision, reduces human errors, and ensures consistency between the conceptual model and its implementation.

Access Control Models: Access control is a fundamental mechanism for regulating access to resources in computing environments. Classical models include Discretionary Access Control (DAC) [3], where resource owners assign permissions; Mandatory Access Control (MAC) [2], where centralized policies enforce strict rules based on security labels; and Role-Based Access Control (RBAC) [4], where permissions are associated with roles rather than individual users, enhancing scalability. Attribute-Based Access Control (ABAC) [5] extends these approaches by making fine-grained decisions based on user, resource, and environmental attributes. Although ABAC improves flexibility, it is typically based on one-time authorization decisions and does not support continuous monitoring during resource usage.

Usage Control: Usage Control (UCON) was introduced to address the limitations of traditional access control models [7]. UCON unifies concepts from access control, trust management, and digital rights management. It continuously monitors authorizations, obligations, and conditions throughout the entire usage session. UCON supports mutability of attributes and reevaluates policies at three phases: pre-authorization, ongoing, and post-authorization [19, 20]. Despite these advantages, UCON has limited applicability in highly dynamic and heterogeneous environments such as IoT-enabled smart homes, vehicles, and healthcare systems [8].

UCON+: To overcome these limitations, UCON+ extends UCON by incorporating hierarchical policy structures, continuous reevaluation across all three phases, and advanced features such as policy delegation and trust management [8]. UCON+ is particularly suitable for cyber-physical and IoT environments, where environmental conditions and user contexts change rapidly. However, current tools and frameworks provide limited support for modeling and enforcing UCON+ policies. This gap underlines the need for a UCON+-oriented domain-specific modeling language that formalizes complex and context-aware security requirements within the scope of model-driven engineering.

3 Motivating Example: UCON+ in a Smart Home Environment

To illustrate the need for a model-driven approach, consider a **smart lock** in a smart home. The lock must enforce *continuous, context-aware access* for a guest, based on identity, trust level (>0.7), time (8:00–20:00), and homeowner proximity (<5m). UCON+ [8] supports *pre-, ongoing-, and post-authorization* with obligations (e.g., log entry), advice (e.g., notify owner), and delegation — unlike traditional models limited to the ongoing phase.

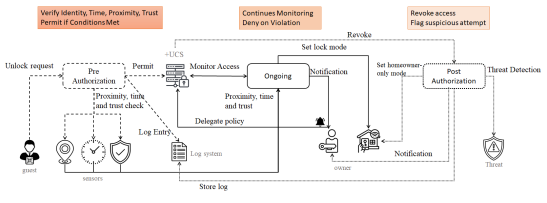

Figure 1. UCON+ control flow for smart lock access control.

Challenge: Manual ALFA is complex and error-prone. The *ongoing phase* alone requires multiple rules for continuous monitoring. For example:

```
rule Ongoing_TimeWindowBreak {
        effect Deny
        condition
        environment.currentTime < time("
             08:00:00")
        or environment.currentTime > time("
             20:00:00");
        advice notify-owner on Deny;
        obligation set-homeowner-only-mode on
             Deny;
}

rule Ongoing_OwnerAway {
        effect Deny
        condition environment.
             ownerProximityMeters > 5.0;
        advice notify-owner on Deny;
        obligation set-homeowner-only-mode on
             Deny;
}

rule Ongoing_TrustDrop {
        effect Deny
        condition subject.trustLevel < 0.70;
        advice notify-owner on Deny;
        obligation set-homeowner-only-mode on
             Deny;
}
```

Listing 1: Ongoing Phase: Time, Proximity, and Trust Monitoring (ALFA)

This snippet shows the *syntactic burden*: nested conditions, multiple obligations, and advice across rules. The complete ALFA policy (7 files, 150+ lines) is available as a **supplementary PDF**:

Download Full ALFA Code (PDF)

This complexity motivates **SmartUcon**: a DSML that enables *graphical modeling*, *metamodel validation*, and *automatic ALFA generation*, reducing errors and effort.

4 Related Work

Existing research on access and usage control spans smart contracts, IoT, cloud, and cyber-physical systems, with several works emphasizing formalization and verification of policies. T"oberg *et al.* [21] model RBAC policies for Ethereum contracts using code generation and formal verification, while Zhang *et al.* [22] and Grompanopoulos *et al.* [23] employ TLA/TLA+ to formalize UCON_{ABC} semantics and verify decision continuity. Complementary studies address enforcement in smart environments: UCIoT [24] enables usage control in IoT homes; SIUV [25] and the work of Ashutosh *et al.* [26] target the automotive domain; and Laamech *et al.* [27] integrate usage control with semantic reasoning for privacy preservation. Surveys such as Akaichi and Kirrane [28] and Gupta *et al.* [29] synthesize challenges and opportunities but do not provide model-driven support.

In parallel, DSL and MDE techniques have increasingly been used to improve the abstraction, automation, and correctness of security policy engineering. Several contributions introduce DSLs for access or usage control, including deontic-rule-based specifications [30], the IEC 62443 DSL for industrial systems [31], and DSLs dedicated to UCON+ [8] and HoBAC [32]. Model-driven approaches also appear in domains such as conversational interfaces [33] and smart environments through SmAuto [17], which demonstrates the benefits of automated validation and code generation. Other works, such as Reina Quintero *et al.* [7], combine DSLs with MDE transformations to integrate UCON modeling into mainstream software practices, whereas Zero-Trust–oriented solutions like ZTA-IoT [34] incorporate dynamic authorization but lack model-based automation.

Recent research highlights a growing convergence between Zero-Trust principles, usage control, and model-driven development. Works such as ZTA-IoT [34] and HoBACDSL [32] show how trust-based reasoning and DSLs can enhance expressiveness and adaptability, while SmAuto [17] and the IEC 62443 DSL [31] demonstrate the increasing integration of MDE for automation and standard compliance.

Research gap. As shown in Table 1, while existing DSLs and metamodels address certain aspects of usage control, a comprehensive solution that fully integrates UCON+ semantics with a complete model-driven engineering pipeline remains absent. For instance, Reina Quintero *et al.* [7] propose a DSL for UCON policies with transformation to XACML, but their work targets the original UCON model and does not incorporate the hierarchical policy structures, trust management, and continuous reevaluation capabilities defined in UCON+. Similarly, SmAuto [17] offers a domain-specific language for general smart environment application development, yet it is not designed for security policy specification and lacks constructs for authorization, obligations, or advice.

In contrast, SmartUcon bridges this gap by providing: (i) a metamodel explicitly aligned with UCON+ constructs, including support for context-aware sensors, events, and device interactions; (ii) an integrated Sirius-based graphical editor that reduces modeling complexity; and (iii) an automated EGL-based transformation to executable ALFA policies. This combination enables end-to-end policy engineering—from high-level visual specification to deployable enforcement—specifically tailored for the dynamic and heterogeneous nature of IoT-enabled smart environments. Unlike prior efforts that focus on isolated aspects of modeling or enforcement, SmartUcon unifies these

dimensions within a single, extensible framework, thereby offering a more practical and cohesive solution for usage control in smart environments.

5 SmartUcon: The Proposed DSML

5.1 Overview

SmartUcon is designed to facilitate the specification and implementation of UCON+ policies in smart environments. As demonstrated in the motivating example, directly encoding policies in a low-level language such as ALFA requires significant expertise and is prone to errors, particularly when addressing the three phases of usage control (pre-, ongoing, and post-authorization), as well as obligations, advice, and delegation. The DSML addresses these challenges by providing a higher level of abstraction through modeling constructs that capture the essential concepts of UCON+ in a structured manner.

SmartUcon is developed within a Model-Driven Engineering (MDE) framework and consists of four main components: **(1) Abstract Syntax:** a metamodel that formally specifies the concepts of UCON+ (subjects, resources, actions, environments, conditions, obligations, advice, and delegation) and their relationships. **(2) Concrete Syntax:** a Sirius-based graphical editor [11] that provides intuitive notations for policy designers to create and manipulate UCON+ models. **(3) Semantics:** defined in a translational manner, where the meaning of each modeling construct is given by its transformation into equivalent constructs in a target policy language (ALFA). In this way, semantics and code generation are inherently connected. **(4) Model Transformation and Code Generation:** an EGL-based mechanism that performs these transformations, automatically generating executable ALFA policies from high-level DSML models.

By grounding semantics in transformation, SmartUcon ensures that policies are both formally specified and directly executable. This systematic, model-driven approach enables stakeholders to focus on policy intent at a conceptual level while ensuring correctness, reducing manual effort, and bridging the gap between high-level UCON+ specifications and low-level ALFA enforcement.

5.2 Abstract Syntax

The abstract syntax of SmartUcon is defined by a metamodel that captures the essential concepts of UCON+ and their relationships. The metamodel formalizes the structure of the language in terms of classes, attributes, and associations, and serves as the foundation for policy modeling. Figure 2 illustrates the proposed metamodel.

At the root of the model, the `UconPlus` class encapsulates one or more `PolicySet` instances, representing logical groupings of policies and their associated networks. A `PolicySet` aggregates `Policy` elements and may contain obligations and advice, with combination operators (e.g., *DenyOverrides*, *PermitOverrides*) defined through the `Combining` class.

Each `Policy` is composed of one or more `Rule` elements. A rule represents an authorization decision and may take the form of either *Permit* or *Deny*. Rules are associated with `TargetClause` and `Condition` classes, which specify the logical expressions that must be satisfied for evaluation. These expressions are defined in the `Expression` class using logical and comparison operators. Policies and rules can also trigger `Obligation` and `Advice` instances, providing mechanisms for actions to be executed or notifications to be delivered alongside authorization results.

The metamodel further incorporates entities to model the execution environment. The `Device` and `Network` classes capture the structure of smart environments, while the `Sensor` class provides context-awareness by continuously supplying environmental data. The `Context` and `Event` classes represent dynamic conditions and state changes, enabling continuous and adaptive enforcement of policies.

Attributes in the metamodel are categorized into *subject*, *resource*, *action*, and *environmental* attributes. They may be predefined (e.g., identity, time, location) or custom-defined to capture domain-specific requirements. This extensibility ensures that the DSML can be adapted across diverse smart environments.

Overall, the metamodel extends traditional access control representations by explicitly supporting obligations, advice, context, delegation, and sensor-driven environmental conditions, thereby capturing the full expressive power of UCON+.

On Aggregation vs. Composition in the Metamodel The proposed metamodel is implemented using the Ecore framework, which does not explicitly distinguish between aggregation and composition in the same way as UML. Instead, containment references in Ecore are used to represent ownership and lifecycle dependencies between elements. In this work, containment relationships are used to ensure structural integrity and enforce hierarchical organization of policy elements. While some of these relationships may conceptually correspond to aggregation, they are modeled as containment in Ecore due to tooling constraints and to guarantee model consistency. This design choice does not affect

Table 1. Comparison of Related Works (AC=Access Control, UC=Usage Control, CG=Code Generation, GM=Graphical Modeling, CA=Context-awareness, II=IoT-Integration, ✓=fully supported, ∼ =partially supported, ×= no support)

Relatedwork	Year	AC	UC	DSL	MDE	CG	GM	CA	II	Smart Environment
Töberg *et al.* [21]	2022	✓	×	×	✓	✓	×	×	×	Smart Contracts
Zhang *et al.* [22]	2004	∼	✓	×	×	×	×	×	×	None
Grompanopoulos *et al.* [23]	2021	∼	✓	×	×	×	×	×	×	None
Marra *et al.* [24]	2017	∼	✓	×	×	×	×	∼	✓	Smart Home
Vaudey *et al.* [31]	2024	∼	×	✓	✓	×	✓	×	×	Industrial Systems
Planas *et al.* [33]	2023	✓	×	✓	✓	✓	×	×	×	None
Panayiotou *et al.* [17]	2024	×	×	✓	✓	✓	×	✓	✓	Smart Home, CPS
Laamech *et al.* [27]	2023	∼	✓	×	×	×	×	✓	✓	Smart Vehicle
Hariri *et al.* [25]	2021	✓	✓	×	×	×	×	✓	✓	Smart Vehicle
Ashutosh *et al.* [26]	2023	✓	×	✓	×	×	×	✓	✓	Smart Vehicle
Ameer *et al.* [34]	2024	✓	✓	×	×	×	×	∼	✓	IoT Systems
Diallo and Adda [32]	2024	✓	×	✓	×	✓	×	×	✓	Smart Home
Hariri *et al.* [8]	2023	∼	✓	×	×	×	×	✓	✓	Smart Home, Vehicle
Reina Quintero *et al.* [7]	2022	∼	✓	✓	✓	✓	×	✓	×	None
Akaichi and Kirrane [30]	2022	∼	✓	✓	×	×	×	✓	×	None
SmartUcon	2026	∼	✓	✓	✓	✓	✓	✓	✓	All Smart Environments

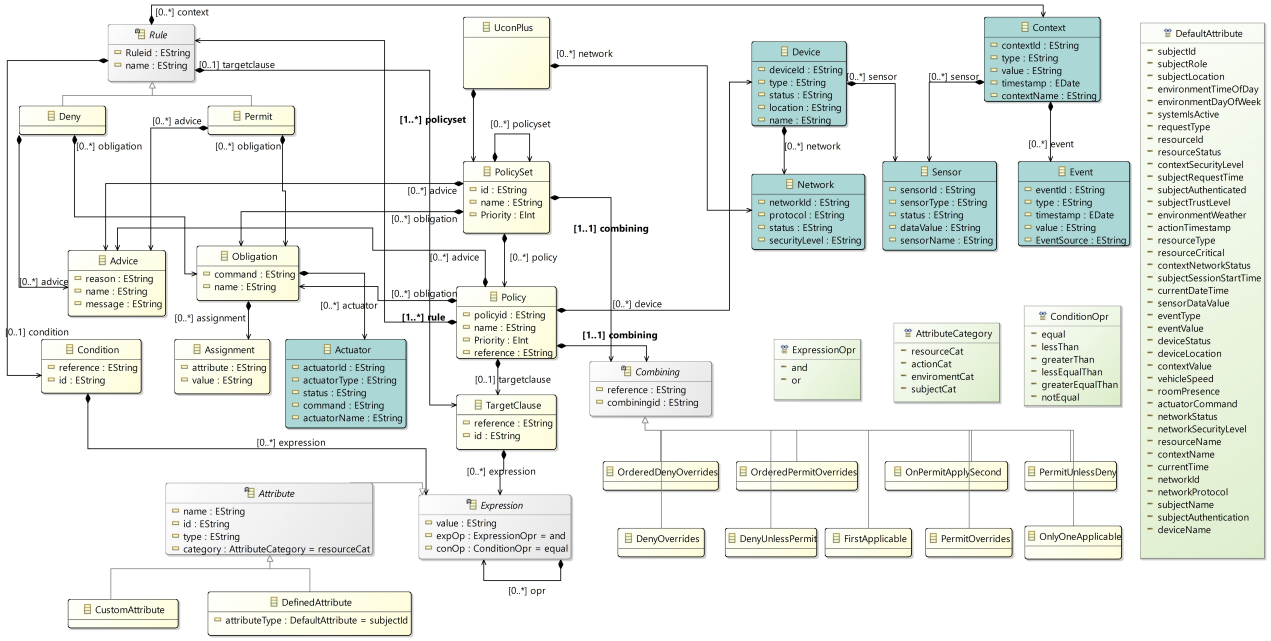

Figure 2. Proposed UCON+ Metamodel

the semantics of the modeling language, but rather reflects the underlying implementation mechanism of the modeling framework.

5.3 Concrete Syntax

The concrete syntax of SmartUcon is realized through a graphical editor that provides an intuitive representation of the concepts defined in the UCON+ metamodel. This editor is implemented with the Sirius framework, which directly supports Ecore-based metamodels, and enables policy designers to visually construct, edit, and validate usage control models.

Each element of the metamodel (e.g., policies, rules, conditions, obligations, advice, contexts, devices, and events) is represented by a dedicated graphical symbol or icon. Through drag-and-drop interactions, users

can add elements to the modeling canvas, define their relationships via graphical links, and configure their attributes in the properties panel. This visual representation reduces the complexity of policy specification and facilitates the design process for both experts and non-experts.

Figure 3 illustrates the editor applied to a lane-change control scenario in a smart vehicle. In this model, multiple *Permit* rules are specified to prevent accidents, optimize traffic flow, and ensure safety. Each rule is associated with conditions, contextual attributes, events, and sensors, and may include obligations for automated actions as well as advice for user or system notifications.

The editor supports hierarchical representation of policies and rules, multiple combining operators, and visualization of decision flows, while ensuring structural conformance to the metamodel. By providing an interactive and user-friendly interface, the concrete syntax enhances model comprehensibility, improves design quality, and reduces the likelihood of conceptual errors during policy specification.

Graphical vs. Textual Modeling Justification. One possible alternative to the proposed graphical DSML is a textual modeling language. However, textual approaches in this domain tend to resemble policy languages such as ALFA, which require programming expertise and impose a significant cognitive burden on non-expert users. In contrast, the primary objective of SmartUcon is to enable domain experts—who may not have strong programming backgrounds—to specify usage control policies at a conceptual level. The graphical modeling approach reduces syntactic complexity, improves comprehensibility, and minimizes user errors, as also confirmed by the usability evaluation results (Section 6.2). Therefore, the choice of a graphical DSML is not merely a design preference, but a deliberate decision to improve accessibility, reduce cognitive load, and bridge the gap between security concepts and practical policy specification.

5.4 Semantics Definition

The semantics of SmartUcon are defined translationally via a model-to-text transformation into the ALFA policy language. Although a fully formal operational or denotational semantics is beyond the scope of this work, we provide a semi-formal specification that clarifies how model constructs are interpreted and ensures consistency with the intended UCON+ behavior.

5.4.1 Mapping Rules

Each element of the SmartUcon metamodel is systematically mapped to a corresponding ALFA construct:

- **PolicySet** ↦ ALFA `policyset` with combining algorithm derived from the `Combining` class (e.g., `denyOverrides`).
- **Policy** ↦ ALFA `policy` containing one or more `rule` instances.
- **Rule** (either `Permit` or `Deny`) ↦ ALFA `rule` with an `effect` of `Permit` or `Deny`.
- **TargetClause** and **Condition** ↦ ALFA `condition` block composed of logical and comparison expressions over attributes.
- **Obligation** ↦ ALFA `obligation` block executed on `Permit` or `Deny` decisions.
- **Advice** ↦ ALFA `advice` block providing optional notifications.
- **Context**, **Event**, and **Sensor** ↦ ALFA attributes accessed via `subject`, `resource`, `action`, or `environment` designators.
- **Device** and **Network** ↦ resource attributes in ALFA policies.

This mapping is *structure-preserving*, meaning that the hierarchical organization of `PolicySet` → `Policy` → `Rule` → `Condition` is maintained in the generated ALFA code.

5.4.2 Semantic Preservation Property

Let M be a SmartUcon model and $T(M)$ its generated ALFA policy. For any access request evaluated under equivalent attribute values and context, the decision produced by $T(M)$ is consistent with the intended UCON+ semantics of M. This property is ensured by: (i) a deterministic transformation process, (ii) explicit mapping of all decision-relevant elements, and (iii) preservation of policy structure and evaluation order.

5.4.3 Alignment with UCON+

The transformation preserves the three evaluation phases of UCON+ (pre-, ongoing-, and post-authorization) by structuring ALFA policies accordingly. For ongoing conditions, ALFA `condition` blocks are embedded within `rule` elements, ensuring continuous reevaluation as attributes change.

5.4.4 Limitations

The transformation is deterministic and rule-based; for a given input model, the generated ALFA code is uniquely determined. However, we do not provide a formal proof of semantic equivalence between SmartUcon models and generated ALFA policies. Establishing such guarantees using formal verification techniques is left for future work.

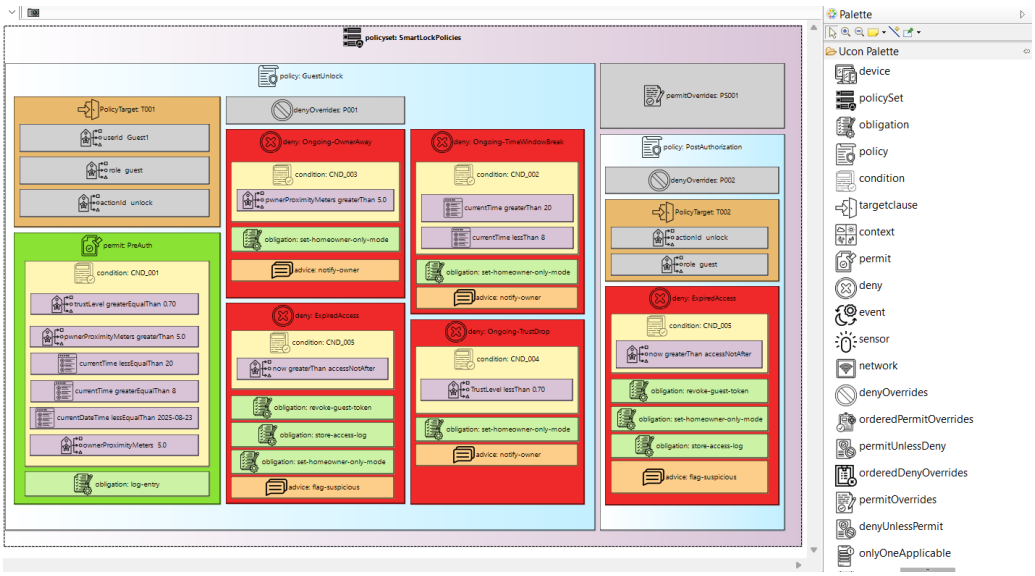

Figure 3. Graphical editor for modeling usage control policies (the motivating example)

5.5 Conflict Detection and Resolution

Due to the expressiveness of SmartUcon, policy conflicts may arise at different levels:

- **Rule Conflicts:** Two or more `Rule` instances may evaluate to `Permit` and `Deny` under the same conditions, leading to ambiguous authorization decisions.
- **Obligation–Advice Conflicts:** Obligations (mandatory actions) and advice (optional notifications) attached to the same `Policy` or `Rule` may prescribe contradictory actions (e.g., `log_entry` vs. `do_not_log`).
- **Contextual Inconsistencies:** Incomplete or conflicting definitions of `Context` and `Event` elements may cause runtime evaluation errors.

SmartUcon addresses these issues at three levels:

- **Design-Time Conflict Avoidance.** The metamodel enforces that each `PolicySet` must specify a `Combining` operator (e.g., `DenyOverrides`, `PermitOverrides`). This operator defines the resolution strategy for conflicting rules within the set. In the generated ALFA code, the combining algorithm is explicitly declared, ensuring deterministic evaluation.
- **Model-Time Validation.** Using the Epsilon Validation Language (EVL) [35], we can detect potential conflicts before code generation. For example, two `Rule` elements with identical `TargetClause` and `Condition` but opposite effects (`Permit` vs. `Deny`) can be detected by an EVL contraint.
- **Runtime Priority.** For obligations vs. advice, the transformation prioritizes obligations by placing them in ALFA `obligation` blocks (which must be fulfilled), while advice is placed in `advice` blocks (which are optional). If both exist, the policy engine will enforce obligations first.

While the current implementation provides conceptual conflict handling, a full-fledged automated conflict resolution engine—capable of suggesting policy merges or detecting semantic contradictions—is planned for future extensions.

5.6 Model Transformation and Code Generation

While semantics define the mapping in principle, their practical realization is achieved through automated model-to-text transformation. To this end, the Epsilon Generation Language (EGL) [12] is employed to generate ALFA policies [13] from DSML models conforming to the UCON+ metamodel. EGL enables systematic traversal of the model and the structured emission of policy code, ensuring that all semantics-preserving mappings are faithfully implemented.

In this research, dedicated EGL templates were developed to cover the complete range of UCON+ constructs, including attributes, policies, rules, obligations, advice, and delegation. Approximately 600 lines of EGL templates were implemented, which were applied to 14 representative case studies. The approach significantly reduces manual coding effort, prevents common syntactic and semantic errors, and guarantees consistency between the graphical model and the executable policy.

6 Evaluation

After the design and implementation of SmartUcon for UCON+ policies in smart environments, a systematic evaluation is required to validate its effectiveness and practicality. The evaluation serves two purposes: (i) to examine whether the language fulfills its design objectives in terms of expressiveness, usability, and precision, and (ii) to provide empirical evidence regarding its applicability across diverse domains.

To this end, the DSML is assessed along four complementary dimensions:

- **Applicability:** the ability of the language to represent different usage control scenarios in heterogeneous domains (e.g., smart homes, connected vehicles, and healthcare) and the complexity of the resulting models.
- **Usability:** the ease with which practitioners can learn and use the language through its graphical editor, including intuitiveness of notation and effort reduction compared to manual ALFA coding.
- **Security:** the assessment of the framework's ability to reduce policy misconfigurations and enforce structural correctness through static validation mechanisms.
- **Software Quality:** the evaluation of the DSML and supporting toolset against standard quality attributes in model-driven engineering, including understandability, maintainability, and extensibility of the generated models and code.

Together, these perspectives provide both quantitative and qualitative insights into the strengths and limitations of the proposed approach, and establish its relevance to real-world smart environments.

6.1 Applicability

One of the primary criteria for evaluating a DSML is its applicability in diverse and realistic scenarios. Applicability refers to the language's ability to accurately and precisely model the requirements of the target domain. In this research, to evaluate the applicability of SmartUcon, a set of 14 case studies in various domains—including smart vehicles, smart cities, smart homes, healthcare, and industrial environments—were designed and implemented. Each case study followed a three-stage modeling process:

- Graphical modeling using the designed editor based on the UCON+ metamodel.
- Automatic transformation of the model into ALFA code using the developed EGL template.
- Evaluation of model complexity by counting the number of classes, attributes, and relationships used.

6.1.1 Representative Case Studies

Fourteen case studies across five domains demonstrate SmartUcon's expressiveness and code generation efficiency. Each policy was modeled using the DSML, validated against the metamodel, and transformed into ALFA via EGL templates. On average, 95% of the expected ALFA code was automatically generated, requiring only minor manual adjustments.

- In Smart Vehicles, three cases were implemented: Case 1 Car Speed Adaptation with three rules for wet/icy roads, traffic density, and adverse weather; Case 2 Car Route Optimization using three rules for congestion, roadblocks, and low fuel; and Case 3 Car Lane Change with three rules based on proximity, traffic flow, and visibility.
- In Smart Cities, Case 4 City Emergency Situation used one Permit rule for accident or fire detection; Case 5 City Light Management applied two rules for low ambient light and energy constraints; and Case 6 City Traffic Management employed two Permit rules for peak and off-peak traffic optimization.
- In Smart Homes, Case 7 Home Resource Control included four rules for guest events, gaming, water usage, and network security; Case 8 Home Management used three Permit rules for door security, HVAC control, and automated lighting; and Case 9 Environmental Control applied three Permit rules for waste management, air quality, and garden irrigation.
- In Healthcare, Case 10 Hospital Nurse Management activated one Permit rule during patient surges; Case 11 Hospital Patient Monitoring used one rule for critical heart rate alerts; and Case 12 Hospital Room Entrance enforced one Deny rule for unauthorized access.
- In Industrial Environments, Case 13 Factory Machine Operators combined one Permit and one Deny rule for role, training, and time restrictions; and Case 14 Factory Production Machine Safety applied two Deny rules for high temperature and operational errors.

Full details of all 14 case studies — including policy models, generated ALFA code, execution logs, and performance metrics — are available in the supplementary PDF.

Results: All policies were successfully modeled, validated, and executed. The high automation rate confirms SmartUcon's ability to abstract UCON+ complexity while preserving full expressiveness across dynamic IoT domains.

6.1.2 Analysis of Evaluations

Model Complexity. To assess the coverage and capability of the proposed modeling language in handling diverse and complex scenarios, the complexity of the designed models was analyzed. Complexity was measured using three main metrics for each model: the number of classes, attributes, and relationships. Table 2 provides the detailed results of this analysis. These metrics demonstrate that the developed modeling language can handle models with both low and high levels of complexity. For instance, the **HomeResourceControl** model, with 55 classes, 103 attributes, and 35 relationships, is one of the most complex cases successfully designed and analyzed with the proposed language. At the same time, relatively small models such as **CityEmergencySituation**, with only 22 classes, also confirm that the language scales well to lightweight scenarios.

Table 2. Complexity of Designed Models

Model	Classes	Attributes	Relationships
CarSpeedAdaptation	52	108	39
CarRouteOptimization	50	116	40
LaneChange	47	105	37
CityEmergencySituation	22	58	16
CityLightManagement	35	83	25
TrafficManagement	38	92	31
HomeResourceControl	55	103	35
HomeManagement	45	94	32
EnvironmentalControl	37	142	41
HospitalNurseManagement	23	58	16
HospitalPatientMonitoring	27	63	18
RoomEntrance	24	63	18
FactoryMachineOperators	26	65	19
ProductionMachineSafety	40	84	29

Distribution of Metamodel Classes. To evaluate the depth of concept coverage by the modeling language, the distribution of key metamodel classes across the 14 case studies was analyzed. Classes such as Rule, Condition, TargetClause, Context, Event, Sensor, Advice, Obligation, Assignment, Actuator, Device, Network, and Combining form the core components of each model. The distribution of these classes across different case studies highlights the flexibility of the proposed language in addressing diverse requirements. For example, the *CarSpeedAdaptation* model includes 3 rules, 3 conditions, 4 target clauses, and 3 obligations, demonstrating multi-layered policy-making with diverse decision criteria. The *EnvironmentalControl* model uses 5 sensors and 3 pieces of advice, showing high diversity in inputs and system responses for smart home automation. Such results show that the designed metamodel components are effective not only in simple models but also in complex, large-scale scenarios.

Analysis of Generated ALFA Code. The evaluation of the ALFA code generated by the EGL templates indicates that the approach can automatically produce the majority of the required code with limited need for manual intervention. Table 3 summarizes the results for all case studies, reporting the generated, corrected, and manually added lines of code, along with their corresponding percentages relative to the final code size. Across all models, between 92% and 100% of the final code was automatically generated. The proportion of corrected lines remained generally below 12%, except in a few complex cases such as `EnvironmentalControl` and `HomeResourceControl`, where it reached 15.2% and 13.5%, respectively. Manual additions were rare, typically below 5%, and only exceeded this threshold in `HomeManagement` (8.1%). These findings confirm that the proposed approach is effective in automatically generating ALFA policies, while the small share of corrections and manual code highlights concrete opportunities for template refinement (e.g., namespace extraction, obligation handling, and advice generation). Overall, the results demonstrate both the feasibility of the approach and its scalability across diverse smart environment scenarios.

Overall Discussion. The evaluation of the proposed modeling language across 14 case studies demonstrates its effectiveness in modeling a wide range of smart environment scenarios, from traffic management and industrial safety to healthcare and home automation. The analysis of model complexity confirms that the language can handle both simple and highly composite structures, while the distribution of metamodel classes shows that its concepts remain applicable across diverse domains. The graphical editor played a key role in facilitating this process by making abstract policy concepts visually accessible and by streamlining the transition from models to executable ALFA code. Furthermore, the ability to analyze and validate policies at the modeling stage proved valuable for ensuring correctness and reducing errors before deployment. Overall, the results indicate that SmartUcon is both expressive and practical, effectively bridging the gap between high-level UCON+ concepts and their low-level implementations.

6.2 Usability Evaluation

To evaluate the usability of SmartUcon and its graphical editor, a standardized questionnaire was administered to 32 participants (professors and graduate students). The choice of academic participants was

Table 3. Generated, Corrected, and Manual ALFA Code across Different Models

Model	LOC				Percentage of Final LOC		
	Generated	Corrected	Manual	Final	% Auto	% Corrected	% Manual
CarSpeedAdaptation	86	6	5	91	94.5%	6.6%	5.5%
CarRouteOptimization	76	7	0	76	100.0%	9.2%	0.0%
LaneChange	83	7	1	84	98.8%	8.3%	1.2%
CityEmergencySituation	33	3	0	33	100.0%	9.1%	0.0%
CityLightManagement	58	6	1	59	98.3%	10.3%	1.7%
TrafficManagement	62	5	2	64	96.9%	8.1%	3.1%
HomeResourceControl	89	12	3	92	96.7%	13.5%	3.3%
HomeManagement	68	7	6	74	91.9%	10.3%	8.1%
EnvironmentalControl	92	14	1	93	98.9%	15.2%	1.1%
HospitalNurseManagement	35	3	0	35	100.0%	8.6%	0.0%
HospitalPationManagement	42	5	0	42	100.0%	11.9%	0.0%
RoomEntrance	35	2	2	37	94.6%	5.7%	5.4%
ProductionMachineSafety	75	7	0	75	100.0%	9.3%	0.0%
FactoryMachineOperators	49	4	0	49	100.0%	8.2%	0.0%

intentional, as the primary objective of the study is to evaluate usability, understandability, and cognitive load rather than domain-specific expertise in security policy engineering. The questionnaire assessed users' familiarity with technical concepts (UCON, UCON+, ALFA, and Ecore), the understandability of different model representations (graphical, tree-based, and textual code), and the overall experience of working with the graphical design environment. Responses were collected via the online platform *Porsline*.

The questionnaire contained 22 questions in multiple-choice, Likert-scale, and comparative formats (Table 4). Questions covered both background knowledge (Q1–Q7) and practical usability aspects, such as understandability (Q8–Q10), comprehension time (Q11–Q13), error likelihood (Q15), convenience of interaction (Q17), environment stability (Q18), and preferences for real-world application (Q21–Q22).

6.2.1 Key Findings

- **Understandability of Representations (Q8–Q10):** As shown in Figure 4, generated ALFA code was rated as moderately or poorly understandable by most participants (around 75%). The tree-based model received slightly better ratings, but understandability remained mixed. The graphical model was rated as the most understandable, with nearly half of participants choosing "High" or "Very High."
- **Comprehension Time (Q11–Q13):** Based on the results in Figure 5, almost 90% of participants understood the graphical model in under 10 minutes, while 72% required more than 10 minutes for the ALFA code. Tree-based models fell in between but were closer to the graphical representation.
- **Error Likelihood (Q15):** For this question, 24 out of 32 participants (75%) identified the graphical model as the least error-prone. Coding was considered riskier (21.4%), and tree-based modeling was perceived as least reliable overall (3.6%).
- **Convenience of Interaction (Q17):** The drag-and-drop interface was overwhelmingly preferred (26 votes or 81.2%). Point-and-click and coding were rated much lower (5 and 1 votes, respectively).
- **Environment Stability (Q18):** The graphical editor ranked highest (18 votes or 56.2%), fol-

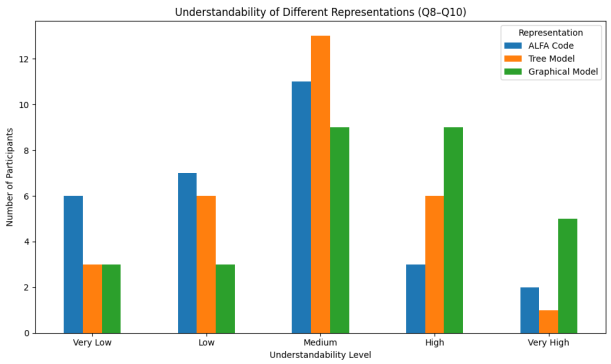

Figure 4. Comparison of understandability across ALFA code, tree-based, and graphical models (Q8–Q10).

Table 4. Questionnaire Questions

No.	Question Text	Response Type
1	What is your educational level?	Multiple-choice (Bachelor's to PhD)
2	Are you familiar with the UCON concept?	Yes / No
3	Are you familiar with the UconPlus concept?	Yes / No
4	How familiar are you with the UconPlus concept?	Likert-scale (Very High to Very Low)
5	How familiar are you with the ALFA language?	Likert-scale (Very High to Very Low)
6	How familiar are you with the Ecore modeling tool?	Likert-scale (Very High to Very Low)
7	How familiar are you with graphical editors?	Likert-scale (Very High to Very Low)
8	How would you rate the understandability of the generated code?	Likert-scale (Very High to Very Low)
9	How would you rate the understandability of the tree-based model?	Likert-scale (Very High to Very Low)
10	How would you rate the understandability of the graphical model?	Likert-scale (Very High to Very Low)
11	How much time is needed to understand the graphical model?	Less than 5 / 5 to 10 / More than 10 minutes
12	How much time is needed to understand the tree-based model?	Less than 5 / 5 to 10 / More than 10 minutes
13	How much time is needed to understand the program code?	Less than 5 / 5 to 10 / More than 10 minutes
14	Which method did you understand more quickly?	Graphical / Tree-based / Program Code
15	Which method do you think has less likelihood of error?	Graphical / Tree-based / Program Code
16	Is the graphical model built more quickly?	Yes / No / Not Sure
17	Which method is more convenient?	Drag & Drop / Point & Click / Coding
18	Which environment is more stable?	Graphical / Tree-based / Coding / Don't Know
19	How appropriate is the design of the graphical editor?	Likert-scale (Very High to Very Low)
20	If the graphical editor were to be changed, what would you prefer?	Reduce Complexity / Increase Clarity / New Features / No Change Needed
21	In a real UconPlus project, which method would you prefer?	Graphical / Tree-based / Coding / Combined
22	Please provide the reason for your choice.	Open-ended

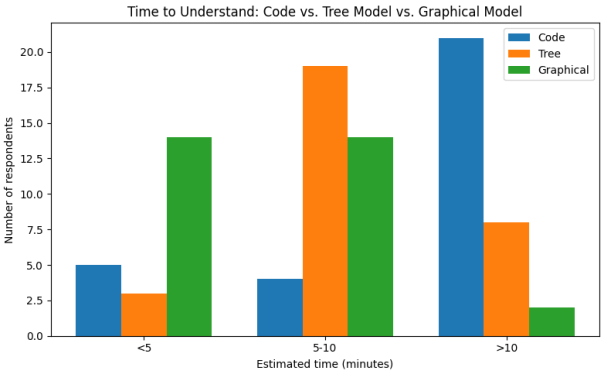

Figure 5. Time required to comprehend different model representations (Q11–Q13).

lowed by coding environment (3 votes or 9.4%) and tree-based modeling (1 vote or 3.1%). Ten participants (31.2%) selected "Don't Know."

- **Preferred Method for Real Projects (Q21):** Based on the results, 21 participants (70%) preferred the graphical editor for real UCON+ projects. Eight (26.7%) suggested a combined approach, while coding (3 votes) and tree-based (2 votes) were rarely chosen.

The radar chart in Figure 6 provides a concise overview of the evaluation results for the three approaches: Graphical Model, Tree Model, and Program Code. The comparison was performed across four key dimensions: *Understandability*, *Comprehension Speed*, *Error-proneness*, and *Stability*. The results clearly demonstrate that the Graphical Model consistently outperforms the other methods in all categories, offering higher understandability, faster learning, fewer errors, and greater stability. These findings confirm the effectiveness of the proposed Graphical Editor and highlight its potential as a reliable and user-friendly tool for designing and understanding usage control policies.

These results clearly indicate that the graphical modeling environment is considered **faster, more understandable, less error-prone, and more stable** than both coding and tree-based modeling.

6.2.2 Statistical Analysis

To validate the observed usability trends, statistical analysis was conducted on the responses of 32 participants.

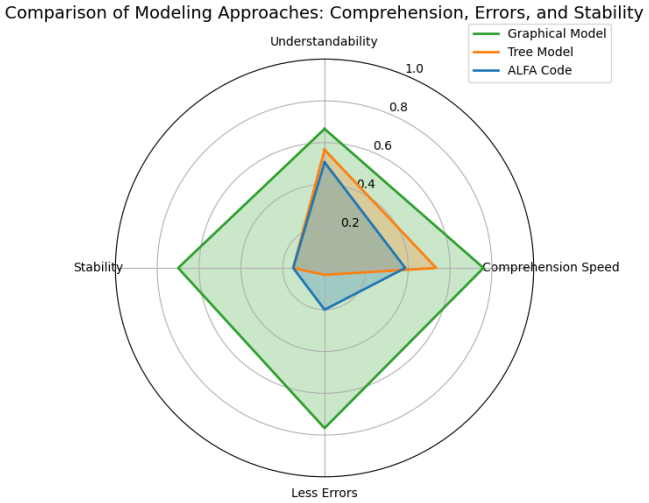

Figure 6. Radar chart comparing Understandability, Error-proneness, and Speed of Learning for three methods: Graphical Modeling, Tree Modeling, and Programing ALFA Code.

Understandability (Q8–Q10). Descriptive results show that the graphical model achieved the highest mean score ($M = 3.33, SD = 1.18$), followed by the tree-based model ($M = 2.87, SD = 0.97$), and program code ($M = 2.53, SD = 1.28$). A repeated measures ANOVA revealed a statistically significant effect of representation type on understandability ($F(2, 58) = 6.87, p = 0.002$). Post-hoc comparisons indicate that the graphical model is significantly more understandable than program code ($p = 0.002$), while differences between the graphical and tree-based models are not statistically significant ($p = 0.098$).

Comprehension Time (Q11–Q13). A non-parametric Friedman rank sum test showed a significant difference in comprehension time across methods ($\chi^2(2) = 28.47, p < 0.001$). Post-hoc pairwise comparisons using Wilcoxon signed-rank tests with Bonferroni correction indicate that both the graphical and tree-based models are understood significantly faster than program code ($p < 0.01$). The difference between the graphical and tree-based models is not statistically significant ($p = 0.054$), although a trend in favor of the graphical model is observed.

User Preference. A chi-square goodness-of-fit test was conducted to analyze participants' preferences. The majority of participants identified the graphical model as the fastest to understand (86.7%) and the least error-prone (76.7%). These distributions are statistically significant ($p < 0.001$), indicating a clear preference for the graphical representation. This result reinforces the consistency between subjective preference and measured usability outcomes.

Overall, the statistical results support the conclusion that graphical modeling improves understandability and reduces cognitive effort compared to alternative representations.

6.2.3 Overall Conclusion

The usability study revealed that the graphical editor received the highest ratings in terms of understandability, speed of comprehension, error reduction, and stability. Most participants also identified drag-and-drop interaction as the most convenient method and preferred the graphical approach for real-world UCON+ projects.

These findings highlight the importance of adopting **visual, intuitive, and interactive approaches** in the design of DSMLs. However, the relatively low understandability of the generated ALFA code suggests that additional supportive documentation, code visualization tools, or automated explanations could further improve user experience, particularly for participants with limited programming backgrounds.

The usability study intentionally used academic participants (professors, graduate students) rather than security practitioners, as the goal was to evaluate usability, understandability, and cognitive load of the graphical modeling approach—not domain expertise. Participants were chosen for their familiarity with modeling and technical systems, making them suitable proxies for interaction design assessment. Additionally, SmartUcon is designed for broader stakeholders, including non-security specialists. However, future studies with professional security practitioners are needed to validate real-world applicability.

Future improvements should focus on:

- Enhancing the clarity of generated code through better formatting or inline explanations.
- Providing interactive tutorials in the editor for non-expert users.
- Supporting combined modeling approaches where users can seamlessly switch between graphical, tree-based, and code representations.

6.3 Security Validation through EVL

To ensure the correctness and reliability of policies defined using the UconPlus metamodel, a comprehensive set of validation constraints (40 rules) is specified using the Epsilon Validation Language (EVL) [35]. These constraints operate at different abstraction levels, including structural integrity, semantic correctness, and context-awareness, and are evaluated directly on model instances during the design phase. The proposed validation framework covers the following categories:

- **Rule Consistency:** Detection of conflicting

rules within a policy, such as the presence of *Permit* and *Deny* rules with identical conditions or overlapping applicability.

- **Context Completeness:** Verification that all conditions and target clauses are well-formed, ensuring that expressions, attributes, and contextual elements are properly defined.
- **Obligation Feasibility:** Ensuring that obligations are executable, i.e., they are associated with valid actuators or assignments and contain well-defined commands.
- **Advice–Obligation Consistency:** Identification of inconsistencies between advisory messages and enforced actions, preventing contradictory or redundant behavior.
- **Combining Semantics:** Validation of policy and policy set combining algorithms (e.g., *PermitOverrides*, *DenyOverrides*) to ensure their correct and meaningful usage.
- **Context and Environment Integrity:** Ensuring that contextual elements such as sensors, events, and devices provide valid and complete information for policy evaluation.
- **Structural and Priority Constraints:** Verification of hierarchical consistency, including unique priorities, proper nesting of policy sets, and valid configuration of networks and devices.

Listing 2 presents an example of EVL constraints for detecting conflicting rules within a policy. Unlike naive implementations, the constraints rely on the actual structure of the metamodel and avoid undefined properties.

```
context Policy
constraint NoConflictingRules {
        check:
        not self.rule.exists(r1 |
        self.rule.exists(r2 |
        r1 <> r2 and
        r1.condition.isDefined() and
        r2.condition.isDefined() and
        r1.condition.expression = r2.condition.
            expression and
        r1.isKindOf(Permit) and
        r2.isKindOf(Deny)
        )
        )
        message: "Conflicting Permit and Deny
            rules with identical conditions"
}
context Rule
constraint RuleMustHaveCondition {
        check: self.condition.isDefined()
        message: "Rule must define a condition"
}
```

Listing 2: EVL constraints for detecting conflicting rules

Overall, the proposed validation framework does not guarantee the complete absence of security vulnerabilities. However, it significantly reduces the likelihood of policy misconfigurations by systematically identifying structural defects, semantic inconsistencies, and context-related issues at design time. This early validation capability improves the robustness, correctness, and maintainability of UconPlus-based policy models. Although explicit threat modeling is not included, the validation rules address common misconfiguration-related risks. Integrating formal threat modeling techniques is part of future work.

6.4 Software Quality Evaluation

In this section, the quality of the SmartUcon metamodel is evaluated. The quality of a software system, particularly in the domain of policy modeling and security control, depends not only on functional correctness but also on structural properties such as adaptability, extensibility, maintainability, and understandability. To ensure a systematic assessment, we adopt a set of established design metrics and perform a comparative structural analysis against related approaches.

6.4.1 Evaluation Framework

The evaluation framework relies on structural metrics that capture key quality characteristics of modeling languages and metamodels. These include the number of classes (NC), number of attributes (NA), number of relationships (NR), maximum inheritance depth ($\mathrm{DIT}_{\max}$), maximum fanout ($\mathrm{Fanout}_{\max}$), number of predecessor nodes (PRED), number of inherited features (INHF), and total number of features (NTF). Based on these low-level measures, three higher-level quality indicators are computed [36]:

Maintainability – average structural complexity, with lower values indicating easier maintenance.

$$\text{Maintainability} = \frac{\mathrm{NC} + \mathrm{NA} + \mathrm{NR} + \mathrm{DIT}_{\max} + \mathrm{Fanout}_{\max}}{5} \tag{1}$$

Understandability – average inheritance-based comprehensibility, where higher values indicate more understandable designs.

$$\text{Understandability} = \frac{\sum_{K=1}^{\mathrm{NC}} (\mathrm{PRED} + 1)}{\mathrm{NC}} \tag{2}$$

Extensibility – inheritance-based extensibility (Attribute Inheritance Factor), where higher values imply greater extensibility.

$$\text{Extensibility} = \frac{\mathrm{INHF}}{\mathrm{NTF}} \tag{3}$$

6.4.2 Structural Comparison of Related Approaches

To contextualize the proposed metamodel, a structural comparison was performed with four related approaches that define metamodels, domain-specific languages, or conceptual/ontological models. Although their levels of formality differ, each of them provides explicit structural elements (classes, attributes, and relations) that enable a meaningful comparison. The selected works are:

- **Planas et al.** [33] – a metamodel for modeling and enforcing access control policies in conversational interfaces.
- **Reina Quintero et al.** [7] – a domain-specific language for specifying UCON policies.
- **Akaichi and Kirrane** [30] – a semantic policy language for usage control expressed through a conceptual model.
- **Laamech et al.** [27] – a conceptual/ontological model for translating UCON policies into semantic rules using OrBAC and SWRL.

Table Table 5 shows the comparison results.

Table 5. Structural Metrics of Related Approaches

Metric	[33]	[7]	[30]	[27]	SmartUcon
NC	29	25	19	23	31
NA	21	16	–	–	59
NR	22	16	13	17	29
DIT_{max}	6	2	3	3	3
$Fanout_{max}$	5	5	5	4	6
PRED	39	15	12	13	16
INHF	34	42	5	8	74
NTF	43	32	13	17	98
Maintainability	16.6	12.8	8	9.4	25.6
Understandability	2.24	1.6	1.63	1.56	1.48
Extensibility	0.79	1.31	0.38	0.47	0.76

6.4.3 Analysis of Results

The comparison yields several observations: **Conceptual richness:** SmartUcon exhibits the largest number of attributes, relationships, and total features, reflecting its broader coverage of policy elements such as obligations, advice, events, contexts, and device integration.

Maintainability: SmartUcon shows the highest (worst) maintainability score among the compared approaches, which is directly attributable to its structural complexity. This result highlights a clear trade-off: while the increased number of elements and relationships improves expressiveness and modeling power, it also leads to reduced maintainability. In other words, the ability to capture richer policy semantics comes at the cost of higher maintenance effort. To mitigate this limitation, future improvements may focus on modularization of the metamodel, refinement of class responsibilities, and restructuring of relationships to reduce unnecessary coupling while preserving expressiveness.

Understandability: Planas [33] achieves the best (highest) understandability score of 2.24, benefiting from its deeper inheritance structure and balanced design. SmartUcon, despite its complexity, maintains a competitive but lower score of 1.48 due to its modular design and graphical representation, while Reina Quintero [7] scores 1.6.

Extensibility: Reina Quintero [7] achieves the highest extensibility score of 1.31, benefiting from its compact and flexible design. SmartUcon scores 0.76, leveraging inheritance for reuse and scalability, which is close to Planas [33] (0.79) but lower than Reina's.

7 Conclusion

This paper introduced a domain-specific modeling language for defining usage control policies in smart environments. Building on the UCON+ foundations, we proposed a structured metamodel that captures essential policy concepts and offers both conceptual clarity and practical support for implementation. The metamodel was realized through a graphical editor in Eclipse Sirius and integrated with an automated ALFA code generation engine using EGL, enabling a seamless transition from design to execution. Its applicability was validated through fourteen case studies in diverse domains and by a controlled user study with 32 participants. The evaluation demonstrated that graphical modeling significantly improves understandability, reduces design time, and minimizes errors compared to tree-based and textual approaches. A structural comparison further highlighted the metamodel's conceptual richness, considerable extensibility, and broad coverage of usage control dimensions, while also revealing the trade-off of increased structural complexity.

Contributions. The main contributions of this work can be summarized as follows: (i) the design of an extensible UCON+ metamodel tailored for smart environments; (ii) the development of a graphical editor that facilitates intuitive policy modeling; (iii) the implementation of an automated ALFA code generator; (iv) validation across 14 representative case studies; (v) an empirical evaluation with human participants confirming usability advantages of graphical modeling; and (vi) a structural comparison positioning SmartUcon against other conceptual, ontological,

and metamodel-based approaches.

Future Directions. Several opportunities remain for further development. Although the current implementation targets ALFA as the execution backend, the underlying metamodel is designed to be backend-agnostic. Extending the transformation engine to generate other policy languages (e.g., Rego for Open Policy Agent (OPA) and XACML) would require additional EGL templates but would not necessitate changes to the core metamodel. The metamodel could be enriched with advanced features such as conflict resolution, hierarchical composition, and multi-organization support. Integrating a lightweight policy evaluation engine inside the modeling environment would facilitate immediate testing. Semantic validation tools could improve accuracy and prevent inconsistencies. Finally, extending the editor into a collaborative Eclipse plugin or a web-based platform would enhance accessibility and adoption in large-scale projects.

Summary. In conclusion, the proposed modeling language provides a comprehensive and extensible foundation for usage control in intelligent environments. Its graphical modeling capabilities, combined with automated code generation and modular design, establish it as a practical tool for both researchers and practitioners. At the same time, its structural analysis shows that while SmartUcon achieves broader conceptual coverage and competitive extensibility compared to related works, its higher complexity leads to greater maintenance effort. This reflects an inherent trade-off in designing expressive metamodels: the challenge of balancing richness and extensibility with maintainability and ease of use. From a design perspective, this trade-off is intentional, as SmartUcon prioritizes expressive power and comprehensive policy representation for complex smart environments, where simplified models may fail to capture critical security and contextual requirements. Overall, SmartUcon contributes to building secure, privacy-respecting, and trustworthy smart environments.

Data Availability Statement

No datasets were generated or analyzed during the current study.

CRediT Author Statement

Mohammad Asmand Joneghani: Conceptualization, Methodology, Formal analysis, Investigation, Writing – original draft. **Leila Samimi-Dehkordi:** Supervision, Validation, Writing – review and editing, Visualization, Project administration, Correspondence.

Declaration of Generative AI and AI-Assisted Technologies in the Writing Process

During the preparation of this work, the authors used ChatGPT (OpenAI) to assist in improving the clarity and language of the manuscript. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the final version of the manuscript.

References

[1] Kaushik Ragothaman, Yong Wang, Bhaskar Prasad Rimal, and Mark Lawrence. Access control for iot: A survey of existing research, dynamic policies and future directions. *Sensors*, 23(4):1805, 2023. . URL https://doi.org/10.3390/s23041805.

[2] Ravi S. Sandhu and Pierangela Samarati. Access control: principles and practice. *IEEE Communications Magazine*, 32(9):40–48, 1994. . URL https://doi.org/10.1109/35.312842.

[3] Ravi S. Sandhu. Lattice-based access control models. *Computer*, 26(11):9–19, 1993. . URL https://doi.org/10.1109/2.241422.

[4] Ravi S. Sandhu. Role-based access control. *Adv. Comput.*, 46:237–286, 1998. . URL https://doi.org/10.1016/S0065-2458(08)60206-5.

[5] Vincent C Hu, David Ferraiolo, Rick Kuhn, Arthur R Friedman, Alan J Lang, and Margaret M Cogdell. Guide to attribute based access control (abac) definition and considerations (draft). *NIST special publication*, 800(162):1–54, 2013.

[6] Sondes Baccouri and Takoua Abdellatif. BIG-ABAC: Leveraging Big Data for Adaptive, Scalable, and Context-Aware Access Control. *CMES - Computer Modeling in Engineering and Sciences*, 143(1):1071–1093, 2025. ISSN 1526-1492. .

[7] Antonia M. Reina Quintero, Salvador Martínez Perez, Ángel Jesús Varela-Vaca, María Teresa Gómez-López, and Jordi Cabot. A domain-specific language for the specification of UCON policies. *J. Inf. Secur. Appl.*, 64:103006, 2022. . URL https://doi.org/10.1016/j.jisa.2021.103006.

[8] Ali Hariri, Amjad Ibrahim, Bithin Alangot, Subhajit Bandopadhyay, Antonio La Marra, Alessandro Rosetti, Hussein Joumaa, and Theo Dimitrakos. Ucon+: comprehensive model, architecture and implementation for usage control and continuous authorization. In *Collaborative Approaches for Cyber Security in Cyber-Physical Systems*, pages 209–226. Springer, 2023.

[9] Abdulgader Almutairi and François Siewe. CA-

UCON: a context-aware usage control model. In Tomoko Yonezawa and Waltenegus Dargie, editors, *CASEMANS@Ubicomp '11: The 5th ACM International workshop on context-awareness for self-managing systems, Beijing, China, 17 September 2011*, pages 38–43. ACM, 2011. . URL https://doi.org/10.1145/2036146.2036153.

[10] Quyet H. Cao, Giyyarpuram Madhusudan, Reza Farahbakhsh, and Noël Crespi. Usage control for data handling in smart cities. In *2015 IEEE Global Communications Conference, GLOBECOM 2015, San Diego, CA, USA, December 6-10, 2015*, pages 1–6. IEEE, 2015. . URL https://doi.org/10.1109/GLOCOM.2014.7417270.

[11] Vladimir Viyović, Mirjam Maksimović, and Branko Perisić. Sirius: A rapid development of dsm graphical editor. In *IEEE 18th International Conference on Intelligent Engineering Systems INES 2014*, pages 233–238, 2014. .

[12] Louis M. Rose, Richard F. Paige, Dimitrios S. Kolovos, and Fiona Polack. The epsilon generation language. In Ina Schieferdecker and Alan Hartman, editors, *Model Driven Architecture - Foundations and Applications, 4th European Conference, ECMDA-FA 2008, Berlin, Germany, June 9-13, 2008. Proceedings*, volume 5095 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2008. . URL https://doi.org/10.1007/978-3-540-69100-6_1.

[13] Qais Tasali, Chandan Chowdhury, and Eugene Y. Vasserman. A flexible authorization architecture for systems of interoperable medical devices. In Elisa Bertino, Ravi S. Sandhu, and Edgar R. Weippl, editors, *Proceedings of the 22nd ACM on Symposium on Access Control Models and Technologies, SACMAT 2017, Indianapolis, IN, USA, June 21-23, 2017*, pages 9–20. ACM, 2017. . URL https://doi.org/10.1145/3078861.3078862.

[14] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro. *EMF: Eclipse Modeling Framework*. Pearson Education, 2008. URL https://sisis.rz.htw-berlin.de/inh2009/12368099.pdf.

[15] Alberto Rodrigues da Silva. Model-driven engineering: A survey supported by the unified conceptual model. *Computer Languages, Systems and Structures*, 43:139–155, 2015. ISSN 1477-8424. URL https://doi.org/10.1016/j.cl.2015.06.001.

[16] Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and how to develop domain-specific languages. *ACM Comput. Surv.*, 37(4): 316–344, 2005. . URL https://doi.org/10.1145/1118890.1118892.

[17] Konstantinos Panayiotou, Constantine Doumanidis, Emmanouil G. Tsardoulias, and Andreas L. Symeonidis. SmAuto: A domain-specific-language for application development in smart environments. *Pervasive Mob. Comput.*, 101: 101931, 2024. . URL https://doi.org/10.1016/j.pmcj.2024.101931.

[18] Leila Samimi-Dehkordi, Bahman Zamani, and Shekoufeh Kolahdouz Rahimi. Leveraging product line engineering for the development of domain-specific metamodeling languages. *J. Comput. Lang.*, 51:193–213, 2019. . URL https://doi.org/10.1016/j.cola.2019.02.006.

[19] Ali Hariri. Extensible model and policy engine for usage control and policy-based governance: Industrial applications. 2024. URL https://tesidottorato.depositolegale.it/bitstream /20.500.14242/61583/1/final-thesis-35-Ali-Hariri.pdf.

[20] Jaehong Park, Xinwen Zhang, and Ravi S. Sandhu. Attribute mutability in usage control. In Csilla Farkas and Pierangela Samarati, editors, *Research Directions in Data and Applications Security XVIII, IFIP TC11/WG 11.3 Eighteenth Annual Conference on Data and Applications Security, July 25-28, 2004, Sitges, Catalonia, Spain*, volume 144 of *IFIP*, pages 15–29. Kluwer/Springer, 2004. . URL https://doi.org/10.1007/1-4020-8128-6_2.

[21] Jan-Philipp Töberg, Jonas Schiffl, Frederik Reiche, Bernhard Beckert, Robert Heinrich, and Ralf Reussner. Modeling and enforcing access control policies for smart contracts. In *2022 IEEE international conference on decentralized applications and infrastructures (DAPPS)*, pages 38–47. IEEE, 2022. URL https://doi.org/10.1109/DAPPS55202.2022.00013.

[22] Xinwen Zhang, Jaehong Park, Francesco Parisi-Presicce, and Ravi S. Sandhu. A logical specification for usage control. In Trent Jaeger and Elena Ferrari, editors, *9th ACM Symposium on Access Control Models and Technologies, SACMAT 2004, Yorktown Heights, New York, USA, June 2-4, 2004, Proceedings*, pages 1–10. ACM, 2004. . URL https://doi.org/10.1145/990036.990038.

[23] Christos Grompanopoulos, Antonios Gouglidis, and Anastasia Mavridou. Specifying and verifying usage control models and policies in TLA$^+$. *Int. J. Softw. Tools Technol. Transf.*, 23(5):685–700, 2021. . URL https://doi.org/10.1007/s10009-020-00600-0.

[24] Antonio La Marra, Fabio Martinelli, Paolo Mori, and Andrea Saracino. Implementing usage control in internet of things: A smart home use case. In *2017 IEEE Trustcom/BigDataSE/ICESS, Sydney, Australia, August 1-4,*

2017, pages 1056–1063. IEEE Computer Society, 2017. . URL https://doi.org/10.1109/Trustcom/ BigDataSE/ICESS.2017.352.

[25] Ali Hariri, Subhajit Bandopadhyay, Athanasios Rizos, Theo Dimitrakos, Bruno Crispo, and Muttukrishnan Rajarajan. SIUV: A smart car identity management and usage control system based on verifiable credentials. In Audun Jøsang, Lynn Futcher, and Janne Merete Hagen, editors, *ICT Systems Security and Privacy Protection - 36th IFIP TC 11 International Conference, SEC 2021, Oslo, Norway, June 22-24, 2021, Proceedings*, volume 625 of *IFIP Advances in Information and Communication Technology*, pages 36–50. Springer, 2021. . URL https://doi.org/10.1007/978-3-030-78120-0_3.

[26] Ashish Ashutosh, Armin Gerl, Simon Wagner, Lionel Brunie, and Harald Kosch. XACML for mobility (XACML4M) - an access control framework for connected vehicles. *Sensors*, 23(4): 1763, 2023. . URL https://doi.org/10.3390/s23041763.

[27] Nouha Laamech, Manuel Munier, and Congduc Pham. Translating usage control policies to semantic rules: A model using orbac and SWRL. In George A. Tsihrintzis, Carlos Toro, Sebastián A. Ríos, Robert J. Howlett, and Lakhmi C. Jain, editors, *Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 27th International Conference KES-2023, Athens, Greece, 6-8 September 2023*, volume 225 of *Procedia Computer Science*, pages 1881–1890. Elsevier, 2023. . URL https://doi.org/10.1016/j.procs.2023.10.178.

[28] Ines Akaichi and Sabrina Kirrane. Usage control specification, enforcement, and robustness: A survey. *CoRR*, abs/2203.04800, 2022. . URL https://doi.org/10.48550/arXiv.2203.04800.

[29] Maanak Gupta, Smriti Bhatt, Asma Hassan Alshehri, and Ravi S. Sandhu. *Access Control Models and Architectures For IoT and Cyber Physical Systems*. Springer, 2022. ISBN 978-3-030-81088-7. . URL https://doi.org/10.1007/978-3-030-81089-4.

[30] Ines Akaichi and Sabrina Kirrane. A semantic policy language for usage control. In Umutcan Simsek, David Chaves-Fraga, Tassilo Pellegrini, and Sahar Vahdat, editors, *Proceedings of Poster and Demo Track and Workshop Track of the 18th International Conference on Semantic Systems co-located with 18th International Conference on Semantic Systems (SEMANTiCS 2022), Vienna, Austria, September 13th to 15th, 2022*, volume 3235 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2022. URL https://ceur-ws.org/Vol-3235/paper10.pdf.

[31] Jolahn Vaudey, Stéphane Mocanu, Gwenaël Delaval, and Éric Rutten. An IEC 62443-security oriented domain specific modelling language. In *Proceedings of the 19th International Conference on Availability, Reliability and Security, ARES 2024, Vienna, Austria, 30 July 2024 - 2 August 2024*, pages 147:1–147:10. ACM, 2024. . URL https://doi.org/10.1145/3664476.3670938.

[32] Mamadou Mouslim Diallo and Mehdi Adda. Hobacdsl: Hobac-focused access control domain specific language. *Procedia Computer Science*, 241:40–47, 2024. ISSN 1877-0509. . URL https://www.sciencedirect.com/science/article/pii/S1877050924017186. 19th International Conference on Future Networks and Communications/ 21th International Conference on Mobile Systems and Pervasive Computing/14th International Conference on Sustainable Energy Information Technology.

[33] Elena Planas, Salvador Martínez, Marco Brambilla, and Jordi Cabot. Modeling and enforcing access control policies in conversational user interfaces. *Softw. Syst. Model.*, 22(6):1925–1944, 2023. . URL https://doi.org/10.1007/s10270-023-01131-3.

[34] Safwa Ameer, Lopamudra Praharaj, Ravi Sandhu, Smriti Bhatt, and Maanak Gupta. Ztaiot: A novel architecture for zero-trust in iot systems and an ensuing usage control model. *ACM Trans. Priv. Secur.*, 27(3):22:1–22:36, 2024. . URL https://doi.org/10.1145/3671147.

[35] Dimitrios S. Kolovos, Richard F. Paige, and Fiona A. C. Polack. On the evolution of OCL for capturing structural constraints in modelling languages. In Jean-Raymond Abrial and Uwe Glässer, editors, *Rigorous Methods for Software Construction and Analysis, Essays Dedicated to Egon Börger on the Occasion of His 60th Birthday*, Lecture Notes in Computer Science, pages 204–218. Springer, 2009. .

[36] Lorenzo Bettini, Davide Di Ruscio, Ludovico Iovino, and Alfonso Pierantonio. Quality-driven detection and resolution of metamodel smells. *IEEE Access*, 7:16364–16376, 2019. .

Mohammad Asmand Joneghani received the M.Sc. degree in Software Engineering from Shahrekord University, Iran. He is currently an Industry Liaison Expert at Shahrekord University, where he facilitates collaboration between academic researchers and industrial organizations. His research interests include software engineering, model-driven engineering, cybersecurity, artificial intelligence, and software-intensive systems. His professional activities focus on strengthening university–industry cooperation, technology transfer, and the development of innovative software and AI-based solutions.

Leila Samimi-Dehkordi is a Faculty Member in the Department of Computer Engineering at Shahrekord University, Iran. She received the B.Sc. degree in Computer Engineering from Iran University of Science and Technology, the M.Sc. degree in Algorithms and Computation from the University of Tehran, and the Ph.D. degree in Software Engineering from the University of Isfahan. Her research interests include model-driven software engineering, software language engineering, secure software systems, privacy-aware computing, and the application of large language models in software engineering and cybersecurity. Her recent research focuses on applying model-driven approaches and AI-assisted techniques to the design, analysis, verification, and security assessment of software-intensive systems.