Fuzzy Requirements Verification in SysML v2: Direct Modeling and Scenario-Based Analysis for Cyber-physical systems

Adel Khelifati[1][a], Malika Boukala-Ioualalen[1][b] and Ahmed Hammad[2][c]
[1]*Faculty of Computer Science, USTHB university, Algiers, Algeria*
[2]*FEMTO-ST Institute, UMR CNRS 6174, Besançon, France*
{*akhelifati, mioualalen*}*@usthb.dz, ahmed.hammad@femto-st.fr*

Keywords: SysML v2, Fuzzy logic, Soft requirements

Abstract: Cyber-physical systems (CPS) often involve vague or qualitative requirements such as comfort or energy efficiency, which are difficult to verify using the crisp Boolean logic traditionally employed in Systems Modeling Language v2 (SysML v2). This paper introduces an approach for modeling and verifying fuzzy requirements directly within SysML v2 without modifying its metamodel or relying on external tools. Fuzzy semantics are encoded using native constructs such as calculation definitions, requirements, and constraints, with satisfaction degrees computed via trapezoidal membership functions and evaluated using the native expression evaluation mechanism provided by the modeling environment. We illustrate the effectiveness and feasibility of our expression-based fuzzy verification approach using a smart building Heating, Ventilation, and Air Conditioning (HVAC) system example and clearly show how the modeling is achieved in standard SysML v2 notation. Furthermore, to extend verification capability under variability and uncertainty, we introduce a complementary external transformation of expression-based model elements into Python scripts to perform scenario-based evaluation. A batch-based exploration method is then presented to systematically analyze fuzzy requirement satisfaction under different runtime conditions, offering insights into system robustness and design-space analysis.

1 Introduction

Cyber-physical systems (CPS) are increasingly prevalent in modern engineering, encompassing systems that tightly couple software components with physical processes. They are critical in intelligent transportation, energy management, aerospace, industrial automation, and smart buildings. These systems must satisfy a broad spectrum of functional and non-functional requirements, including safety, performance, reliability, and more qualitative aspects such as comfort, usability, or energy efficiency. Many of these requirements are inherently imprecise, context-dependent, or subjective. For instance, a "comfortable temperature" or "low power usage" is not strictly defined but varies across users, contexts, and operational conditions.

Model-Based Systems Engineering (MBSE) provides a framework to manage the complexity of CPS by enabling formalized system specification, traceability, and early-stage analysis. Within MBSE, the Systems Modeling Language (SysML) (Object Management Group (OMG), 2012), maintained by the Object Management Group (OMG), has become a widely adopted standard. The recent evolution to SysML v2 (Object Management Group (OMG), 2024) introduces significant improvements in semantic precision, modularity, and expressiveness, along with a formal textual syntax that supports native expression evaluation and analysis.

However, despite these advancements, existing verification mechanisms in SysML v2 remain grounded in classical Boolean logic. Requirements are typically evaluated as either completely satisfied or entirely violated. This binary interpretation is insufficient when dealing with soft requirements, where partial compliance is expected and acceptable. Consequently, system engineers are limited in their ability to express and evaluate degrees of satisfaction, leading to a potential mismatch between stakeholder expectations and formal system verification.

To address this limitation, we propose an ap-

[a] https://orcid.org/0009-0006-4522-8123
[b] https://orcid.org/0000-0002-8713-4997
[c] https://orcid.org/0000-0003-3739-1650

proach that integrates fuzzy logic (Zadeh, 1965) into SysML v2 models for the expression-based specification and evaluation of fuzzy requirements. Our method leverages only native elements of SysML v2, such as `calc def`, `requirement`, `attribute`, and `constraint` to define fuzzy semantics, without modifying the metamodel. Using trapezoidal membership functions, we compute satisfaction degrees as real values in the range $[0, 1]$, enabling fine-grained verification of vague or context-sensitive requirements directly within the model. Standard constraint mechanisms can assert these satisfaction degrees, providing partial, traceable, and explainable verification.

In addition to in-model verification, we extend our approach with a complementary batch-based analysis mechanism. This extension allows engineers to explore how fuzzy requirement satisfaction evolves under varying operational scenarios, such as changes in environmental conditions or system usage patterns. While our core method operates entirely within the SysML v2 environment, the batch extension exports model elements involving expressions to an external Python script that simulates multiple configurations and assesses robustness through statistical analysis. This two-tiered process combines the rigor of expression-based modeling with the flexibility of scenario exploration.

The main contributions of this paper are as follows:

- We propose an approach to model and verify fuzzy requirements in SysML v2 using only native constructs and expression evaluation capabilities.
- We validate the method through a case study involving a smart building HVAC system, where comfort and energy-related requirements are evaluated under uncertainty.
- We propose a batch-based extension to simulate scenario variability and assess the robustness of fuzzy requirement satisfaction across multiple configurations.

The remainder of the paper is organized as follows: Section 2 introduces key concepts from SysML v2 and fuzzy logic. Section 3 presents related work on fuzzy modeling and verification in system design. Section 4 describes our modeling and verification method in detail and illustrates it through a case study. Section 5 details the batch-based scenario exploration. Finally, Section 6 concludes the paper and outlines directions for future research.

2 Background

This section introduces the fundamentals of SysML v2 and fuzzy logic relevant to our approach.

2.1 SysML v2 and Constraint Evaluation

SysML v2 (Object Management Group (OMG), 2024) provides standardized constructs for the specification, structuring, and symbolic evaluation of complex system models. Our approach utilizes core constructs including `part def`, `part`, `requirement`, `calc def`, `constraint`, and `satisfy` relationships.

Part Definitions and Usages. System components are defined using the `part def` construct, which specifies structural attributes such as physical properties or system parameters (Listing 1). Concrete instances of these components are created using `part`, with attribute values defined by the keyword `ref` (Listing 2).

Listing 1: Example of `part def`

```
part def A {
    attribute x: Real;}
```

Listing 2: Example of `part` usage

```
part a: A {
    ref x = 5.0;}
```

Requirements. Requirements are declared using the `requirement` construct, which may include a textual description (`doc`), attributes, and embedded constraints (Listing 3).

Listing 3: Example of a requirement

```
requirement r1 {
    doc /* The value of x shall remain
         below a fixed limit */
    attribute limit: Real = 10.0;
    constraint { a.x < limit }}
```

Calculation Definitions. Reusable computational functions are defined using `calc def` (Listing 4) and invoked throughout the model (Listing 5).

Listing 4: Calculation definition (`calc def`)

```
calc def add {
    in a: Real;
    in b: Real;
    return r: Real;
    a + b }
```

Listing 5: Calling a defined calculation

```
attribute sum: Real = add(3.0, 2.0);
```

Satisfaction Links (`satisfy`). The `satisfy` relationship explicitly links system components (`part`) to their corresponding requirements, ensuring traceability and verification (Listing 6).

Listing 6: Linking a part to a requirement

```
satisfy r1 by a;
```

Constraints and Their Evaluation. Constraints formally specify conditions that must be fulfilled by the model elements. Constraints can be defined globally or locally within elements.

Consider the example in Listing 7, where a constraint evaluates whether the sum of two attributes remains below a certain threshold using a calculation definition.

Listing 7: Complete constraint evaluation example

```
package addPackage {
  private import ScalarValues::*;
  calc def add {
      in a: Real;
      in b: Real;
      return r: Real;
      a + b }
  part def A {
      attribute x: Real;
      attribute y: Real; }
  part a: A {
      ref x = 5.0;
      ref y = 6.0; }
  requirement r1 {
      doc /* The sum of x and y shall
          remain below a fixed limit
          */
      attribute limit: Real = 10.0;
      constraint { add(a.x, a.y) <
          limit } }
      }
```

Initially, with attributes set to $x = 5.0$ and $y = 6.0$, the evaluation proceeds as:

$$add(5.0, 6.0) = 11.0 \quad \Rightarrow \quad 11.0 < 10.0\ (\texttt{false})$$

This evaluation step is illustrated in Figure 1 (after evaluation), confirming that the constraint is not satisfied due to the sum exceeding the threshold.

These SysML v2 constructs provide a robust and structured foundation for specifying and evaluating requirements in complex systems, facilitating subsequent integration with fuzzy logic in our proposed approach.

2.2 Fuzzy Logic

Fuzzy logic, introduced by Zadeh in 1965 (Zadeh, 1965), extends classical logic by allowing truth values to range continuously between 0 and 1. It is particularly well-suited for representing qualitative, vague, or imprecise requirements, such as “low energy consumption” or “comfortable temperature,” which cannot be captured effectively by crisp Boolean logic.

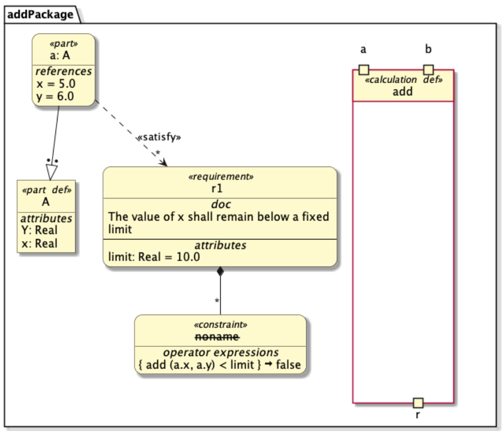

Figure 1: SysML v2 constraint evaluated to `false`, since $5.0 + 6.0 \geq 10.0$.

Fuzzy reasoning relies on membership functions to determine the degree to which a given input satisfies a fuzzy concept. Several types of membership functions exist, including triangular, trapezoidal, Gaussian, and sigmoid functions, each offering different modeling flexibility and complexity.

In this paper, we focus on the trapezoidal membership function due to its simplicity and widespread use in engineering applications. This function is defined by four parameters (a, b, c, d) and evaluated at a specific input value x, returning a satisfaction degree $\mu(x)$ between 0 and 1, as presented in Equation 1.

$$\mu(x) = \begin{cases} 0 & \text{if } x \leq a \text{ or } x \geq d \\ \frac{x-a}{b-a} & \text{if } a < x < b \\ 1 & \text{if } b \leq x \leq c \\ \frac{d-x}{d-c} & \text{if } c < x < d \end{cases} \tag{1}$$

3 Related Work

Fuzzy requirements modeling was first explored by Bubenko et al. (Bubenko et al., 1994) and Liu and Yen (Liu, 1998), who emphasized the use of fuzzy sets to represent qualitative and imprecise stakeholder requirements. Their work focused on capturing vague notions such as “low cost” or “easy to use,” and formalizing them through membership functions and trade-off analysis. These early contributions aimed to support requirements elicitation and refinement but did not offer executable or verifiable models.

Building on this foundation, Baresi et al. (Baresi et al., 2010) introduced FLAGS, an extension of the KAOS goal modeling framework that incorporates fuzzy and adaptive goals to support runtime requirement satisfaction and dynamic adaptation. While FLAGS enables partial satisfaction tracking and goal reconfiguration, it primarily addresses high-level goal modeling and lacks support for direct verification within design models.

Fuzzy logic has also been applied in extensions of the Unified Modeling Language (UML). Ma et al. (Ma et al., 2012) proposed fuzzy class diagrams by introducing fuzzy classes, attributes, and relationships to handle uncertainty in data models. Their approach facilitates conceptual modeling under imprecision but does not target operational constraints or verification. Similarly, Han et al. (Han et al., 2014; Han et al., 2016) proposed the FAME UML profile for modeling fuzzy self-adaptive systems using stereotypes and views for fuzzy control loops. However, FAME focuses on adaptation logic rather than integrating fuzzy verification directly into design models.

Recent frameworks such as PERSA (DionisioParaiba and Martins,), which combines fuzzy logic with the NFR-Framework, and the conceptual model proposed by Egesoy and Güzel (Egesoy and Güzel, 2021) demonstrate the relevance of fuzzy techniques for managing soft and adaptive requirements. Nonetheless, both approaches remain conceptual or external to standard modeling environments, offering little to no integration with modeling tools like SysML or UML.

In the context of Product Lifecycle Management (PLM), Taratoukhine and Yadgarova (Taratuknin and Yadgarova, 2015) applied fuzzy logic to synthesize product configurations from imprecise customer needs. Although effective in its domain, this work remains domain-specific and disconnected from general-purpose system modeling languages.

Within Model-Based Systems Engineering (MBSE), fuzzy logic has been used to support architectural evaluations under uncertainty. Dagli et al. (Dagli et al., 2009) introduced a fuzzy reasoning approach to explore architectural alternatives by combining fuzzy associative memories with evolutionary algorithms. While this approach is powerful for early design exploration, it does not focus on requirements verification nor leverage modeling standards such as SysML.

Yoo and Lee (Yoo and Lee, 2019) proposed integrating SysML parametric diagrams with MATLAB and the Analytic Hierarchy Process (AHP) to assess maintainability using fuzzy values. Their approach highlights the benefit of combining modeling and fuzzy analysis but requires tool coupling and external processing, limiting the cohesion and reusability of the model.

In addition to in-model evaluation, some approaches rely on external tools for scenario simulation or robustness analysis. For example, Yoo and Lee (Yoo and Lee, 2019) use MATLAB to simulate different maintainability scenarios. Other approaches may rely on co-simulation frameworks or script-based tools to evaluate system behavior under uncertainty. However, these techniques often involve heavy integration effort or custom toolchains.

In summary, existing approaches for modeling and verifying fuzzy requirements generally fall into one of the following categories: conceptual methods without execution support, metamodel extensions in UML, external fuzzy reasoning tools, or hybrid frameworks requiring tool integration. Very few solutions enable traceable verification of fuzzy requirements directly within a standard system modeling language.

In response to these limitations, we propose an approach that supports the direct modeling and evaluation of fuzzy requirements within SysML v2. Our method avoids metamodel extensions, relies solely on native language constructs, and provides quantitative and traceable verification through expression-based evaluation within the modeling environment. To further enhance the analysis capabilities under variability and uncertainty, we also propose an external lightweight extension for scenario-based exploration.

4 Direct Verification of Fuzzy Requirements in SysML v2

In this section, we present our approach for modeling and verifying fuzzy requirements directly within the SysML v2 modeling environment using only native constructs. To demonstrate the feasibility and effectiveness of our method, we apply it to a representative case study involving a smart building HVAC (Heating, Ventilation, and Air Conditioning) system.

4.1 Modeling Fuzzy Requirements with Native SysML v2 Constructs

Our approach leverages core SysML v2 constructs, including calculation definitions (`calc def`), requirements, attributes, and constraints. This ensures full compliance with the SysML v2 metamodel, while enabling expression-based evaluation of fuzzy semantics within the model. Satisfaction degrees are computed using trapezoidal membership functions expressed through standard calculation logic.

4.1.1 Trapezoidal Membership Function Implementation

We implement the fuzzy semantics using a trapezoidal membership function, chosen for its simplicity and interpretability in engineering applications. The function is encoded as a reusable `calc def`, shown in Listing 8.

Listing 8: Trapezoidal Membership Function

```
calc def trapezoid {
    in x: Real;
    in a: Real;
    in b: Real;
    in c: Real;
    in d: Real;
    return r: Real;
    if x < a or x > d ? 0
    if x > a and x < b ? ((x - a) / (b
        - a))
    if x >= b and x <= c ? 1
    if x > c and x < d ? ((d - x) / (d
        - c))
    if x >= d ? 0 }
```

This definition can be reused to define fuzzy satisfaction degrees for various attributes across the model.

4.1.2 Generic Fuzzy Requirement Specification

Fuzzy requirements are modeled using standard `requirement` blocks, enhanced with attributes that compute satisfaction degrees using the trapezoidal function. Constraints are then applied to evaluate whether the degree meets a minimum acceptable threshold, as shown in Listings 9, 10, and 11.

Listing 9: Generic Fuzzy Requirement

```
requirement simpleReq {
    doc /* The value shall be
        approximately between 6 and 8.
        */
    attribute val: Real = 7.5;
    attribute mu: Real = trapezoid(val
        , 5.0, 6.0, 8.0, 9.0);
    assert constraint {
        mu > 0.6 } }
```

Listing 10: System Element Definition

```
part def SystemElement {
    attribute input: Real; }
part p: SystemElement {
    ref input = 7.5; }
```

Listing 11: Linking Requirement and Part

```
satisfy simpleReq by p;
```

This modeling pattern is fully evaluatable within SysML v2, relying exclusively on expression evaluation through native constraint logic.

4.2 HVAC System Case Study

To demonstrate our approach, we model a smart building HVAC (Heating, Ventilation, and Air Conditioning) system tasked with ensuring both thermal comfort and energy efficiency. These are two inherently vague and context-sensitive requirements. The HVAC system must maintain an indoor temperature that is perceived as comfortable by occupants, while minimizing energy consumption. These objectives are naturally formulated as fuzzy requirements, reflecting user preferences and operational tolerances.

4.2.1 SysML v2 Model of HVAC System

The system is modeled in SysML v2 using the following structure:

- A `part def` block defines the HVAC system with attributes for temperature and power.
- Two fuzzy requirements are created: one for comfort, and one for power efficiency.
- Satisfaction degrees are computed using trapezoidal membership functions, with thresholds specified via constraints.

The parameters used for the trapezoidal functions are summarized in Table 1.

Table 1: Trapezoidal Parameters for HVAC Fuzzy Requirements

Requirement	**Trapezoidal parameters**
Thermal Comfort	$(18, 22, 24, 29)$ °C
Energy Efficiency	$(0, 100, 200, 300)$ Watts

Listing 12 shows the full model definition.

Listing 12: HVAC SysML v2 Model

```
part def HvacSystem {
    attribute temperature: Real;
    attribute power: Real; }
part hvac: HvacSystem {
    ref temperature = 27.0;
    ref power = 227.5;}
requirement comfortReq {
    doc /* The ambient temperature
        shall be comfortable. */
    attribute mu_comfort : Real =
        trapezoid(hvac.temperature,
        18.0, 22.0, 24.0, 29.0);
    assert constraint
        ComfortConstraint { mu_comfort
         > 0.6 } }
requirement powerReq {
    doc /* The power usage shall
        remain reasonably low. */
    attribute mu_power : Real =
        trapezoid(hvac.power, 0.0,
        100.0, 200.0, 300.0);
    assert constraint PowerConstraint
        { mu_power > 0.6 } }
satisfy comfortReq by hvac;
satisfy powerReq by hvac;
```

Figures 2 present the model after expression evaluation.

4.2.2 Evaluation and Discussion

Following the expression-based evaluation in SysML v2, the computed satisfaction degrees for the HVAC system are as follows:

- Comfort satisfaction degree: $\mu_{\text{comfort}} = 0.4$ (requirement **not satisfied**).
- Energy satisfaction degree: $\mu_{\text{power}} = 0.725$ (requirement **satisfied**).

These results indicate that the current temperature lies outside the defined fuzzy comfort zone, whereas energy consumption remains within the acceptable fuzzy interval. This kind of partial satisfaction feedback, where some requirements are met to a certain degree, and others are not, is particularly valuable during the early design stages. It enables engineers to reason about soft requirements with a finer granularity than binary logic allows and to iteratively refine system parameters or membership thresholds.

More broadly, this case study illustrates that SysML v2 can be effectively used to model and verify fuzzy requirements using only native constructs. Our approach requires no metamodel extensions or external engines. Instead, it relies on the language's built-in expression evaluation capabilities to provide a fully integrated and interpretable verification process.

Using fuzzy satisfaction degrees within SysML v2 offers practical benefits, including early detection of requirement mismatches, enhanced traceability and explainability through quantitative insights, and improved maintainability via modular updates. Overall, our approach enriches SysML v2's expressiveness, enabling quantitative reasoning over soft requirements without deviating from the standard language specification.

5 Batch-Based Scenario Exploration

While our proposed approach enables the evaluation of fuzzy constraints directly within a SysML v2 model using expression-based constructs, Cyber-Physical Systems (CPS) often operate in inherently dynamic and uncertain environments. For example, temperature and energy usage in a smart building may vary due to changes in weather, occupancy, or user behavior. Verifying a single configuration, although valuable, remains insufficient to assess how the system performs across a spectrum of real-world situations.

To address this limitation, we introduce a lightweight **batch-based extension** that supports the evaluation of fuzzy requirement satisfaction under multiple runtime conditions. This allows engineers to observe how the system responds to input fluctuations and to reason about robustness and sensitivity during early-stage design.

Although SysML v2 provides powerful expression-based modeling capabilities, it does not natively support scenario generation or statistical analysis. However, its precise structure and reusable expressions can be exported to an executable language. In our case, we transform the trapezoidal membership function and requirement logic into Python code. This transformation is non-intrusive and preserves the semantics of the original model.

5.1 Scenario Generation in Python

We define three scenarios reflecting environmental variability: **S1** (mild and stable conditions), **S2** (same average as S1, higher variability), and **S3** (stressful conditions with elevated temperature and energy usage).

Each scenario generates a set of temperature and power consumption values by sampling from a normal (Gaussian) distribution using the function `np.random.normal(mean, std, size)`, producing `size` samples (10 in our case) with an approximate mean of `mean` and variability controlled by `std`.

A small standard deviation (e.g., 1) simulates stable or predictable environmental conditions, while a larger standard deviation (e.g., 3) represents scenarios with greater variability or uncertainty. Listing 13 presents the complete Python script used for generating and evaluating these scenarios.

Listing 13: Python script for evaluating fuzzy requirements across multiple scenarios.

```
import numpy as np
# Trapezoidal membership function
def trapezoid(a, b, c, d, x):
```

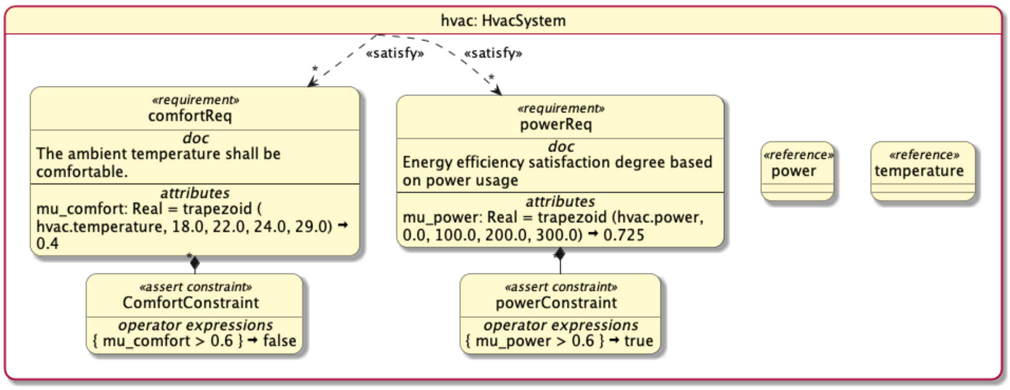
hvac: HvacSystem
«satisfy»
«satisfy»
«requirement»
comfortReq
doc
The ambient temperature shall be comfortable.
attributes
mu_comfort: Real = trapezoid (hvac.temperature, 18.0, 22.0, 24.0, 29.0) → 0.4
«requirement»
powerReq
doc
Energy efficiency satisfaction degree based on power usage
attributes
mu_power: Real = trapezoid (hvac.power, 0.0, 100.0, 200.0, 300.0) → 0.725
«reference»
power
«reference»
temperature
«assert constraint»
ComfortConstraint
operator expressions
{ mu_comfort > 0.6 } → false
«assert constraint»
powerConstraint
operator expressions
{ mu_power > 0.6 } → true

Figure 2: SysML v2 Model After Evaluation (Computed Satisfaction Degrees)

```
    if x <= a or x >= d: return 0.0
    elif a < x < b:
        return (x - a) / (b - a)
    elif b <= x <= c: return 1.0
    elif c < x < d:
        return (d - x) / (d - c)
    else:  return 0.0
# Membership functions for fuzzy
    requirements
def comfort_membership(temp):
 return trapezoid(18, 22, 24, 28, temp
    )
def energy_membership(power):
 return trapezoid(0,100,200,300, power
    )
# Scenario definitions
scenarios = {
 "S1":{"description":"Mild␣and␣stable"
    ,
        "temp": np.random.normal(23,
            1, 10),
        "power": np.random.normal(120,
             10, 10), },
 "S2":{"description":"Same␣mean,␣more␣
    variance",
        "temp": np.random.normal(23,
            3, 10),
        "power": np.random.normal(120,
             30, 10), },
 "S3":{"description":"Stress␣condition
    ",
        "temp": np.random.normal(27,
            1, 10),
        "power": np.random.normal(250,
             15, 10), },}
# Evaluate satisfaction degrees
for name, scenario in scenarios.items
    ():
    mu_comfort = [comfort_membership(t
        ) for t in scenario["temp"]]
    mu_energy = [energy_membership(p)
        for p in scenario["power"]]
    scenario["avg_comfort"] = np.mean(
        mu_comfort)
    scenario["avg_energy"] = np.mean(
        mu_energy)
# Output results
for name, s in scenarios.items():
  print(f"{name}␣-␣{s['description']}"
    )
  print(f"␣␣Avg␣Comfort:␣{s['
      avg_comfort']:.3f}")
  print(f"␣␣Avg␣Energy:␣␣{s['
      avg_energy']:.3f}")
```

5.2 Results and Interpretation

Figure 3 illustrates the average satisfaction degrees computed for each scenario.

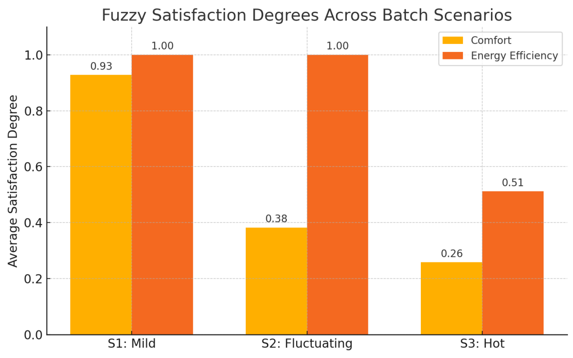

Figure 3: Average fuzzy satisfaction degrees across batch-generated scenarios.

The results reveal that under **S1** (stable conditions), both requirements are well satisfied, indicating robustness. In **S2**, increased variability leads to lower satisfaction, especially for comfort. Under **S3** (stressful conditions), satisfaction levels drop significantly, showing the system's limits in extreme scenarios.

This analysis reveals behavioral trends that cannot be captured by static Boolean verification. Designers can detect system limitations, improve parameter ranges, or tune satisfaction thresholds accordingly.

5.3 Discussion

The batch-based evaluation enhances SysML v2 with variability-aware analysis, allowing robustness assessment under uncertainty, sensitivity exploration to identify critical inputs, and flexible design-space exploration without modifying the core model. Although the scenario evaluation is performed externally, the transformation is lightweight and preserves model semantics. This hybrid architecture bridges the gap between structural modeling and dynamic runtime reasoning, offering a scalable and practical method for validating fuzzy requirements across diverse CPS environments.

6 Conclusion

This paper introduced a model-based verification approach for Cyber-Physical Systems (CPS) that supports specifying and evaluating vague or imprecise requirements such as comfort or energy efficiency using fuzzy logic directly within the standard SysML v2 language. Unlike existing approaches that rely on metamodel extensions or external reasoning tools, our method leverages native SysML v2 constructs, including calculation definitions, attributes, constraints, and requirements, to encode fuzzy semantics in a lightweight and compliant manner.

We demonstrated the feasibility of our approach through a case study of a smart building HVAC system, where fuzzy satisfaction degrees were evaluated using trapezoidal membership functions and standard constraint mechanisms. This allows for continuous, explainable, and traceable verification of soft requirements, moving beyond the limitations of Boolean logic. Our method thus supports early-stage analysis, enabling engineers to reason about partial compliance and explore trade-offs under uncertainty.

To complement the in-model evaluation, we proposed a batch-based scenario analysis that exports key model logic to Python. This external extension allows system evaluation under diverse operating conditions using randomized scenarios, offering insights into system robustness, variability sensitivity, and performance boundaries.

In future work, we plan to extend the method to support composite fuzzy constraints involving multiple interacting variables and apply the approach to larger, more complex CPS domains to evaluate its scalability and applicability across diverse engineering contexts.

REFERENCES

Baresi, L., Pasquale, L., and Spoletini, P. (2010). Fuzzy goals for requirements-driven adaptation. In *2010 18th IEEE international requirements engineering conference*, pages 125–134. IEEE.

Bubenko, J., Rolland, C., Loucopoulos, P., and DeAntonellis, V. (1994). Facilitating” fuzzy to formal” requirements modelling. In *Proceedings of IEEE International Conference on Requirements Engineering*, pages 154–157. IEEE.

Dagli, C. H., Singh, A., DAUBY, J. P., and Wang, R. (2009). Smart systems architecting: computational intelligence applied to trade space exploration and system design. In *Systems Research Forum*, volume 3, pages 101–119. World Scientific.

DionisioParaiba, J. and Martins, L. E. G. A proposal of requirements specification process for adaptive systems based on fuzzy logic and nfr-framework.

Egesoy, A. and Güzel, A. (2021). Fuzzy logic support for requirements engineering. *International Journal of Innovative Research in Computer Science & Technology*, 9(2):14–21.

Han, D., Yang, Q., and Xing, J. (2014). Extending uml for the modeling of fuzzy self-adaptive software systems. In *The 26th Chinese Control and Decision Conference (2014 CCDC)*, pages 2400–2406. IEEE.

Han, D., Yang, Q., Xing, J., Li, J., and Wang, H. (2016). Fame: A uml-based framework for modeling fuzzy self-adaptive software. *Information and Software Technology*, 76:118–134.

Liu, X. F. (1998). Fuzzy requirements. *IEEE Potentials*, 17(2):24–26.

Ma, Z. M., Yan, L., and Zhang, F. (2012). Modeling fuzzy information in uml class diagrams and object-oriented database models. *Fuzzy Sets and Systems*, 186(1):26–46.

Object Management Group (OMG) (2012). OMG Systems Modeling Language SysML. Technical report.

Object Management Group (OMG) (2024). Systems Modeling Language (SysML) v2 Beta 2 Specification: Language. www.omg.org/spec/SysML/2.0/Beta2/Language/PDF. Accessed Mars 2025.

Taratuknin, V. and Yadgarova, Y. (2015). A fuzzy logic approach for product configuration and requirements management. In *2015 Annual Conference of the North American Fuzzy Information Processing Society (NAFIPS) held jointly with 2015 5th World Conference on Soft Computing (WConSC)*, pages 1–5. IEEE.

Yoo, Y.-Y. and Lee, J.-C. (2019). The improvement of maintainability evaluation method at system level using system component information and fuzzy technique. *Journal of the Korea Academia-Industrial cooperation Society*, 20(3):100–109.

Zadeh, L. A. (1965). Fuzzy sets. *Information and control*, 8(3):338–353.