Intelligent Automation & Soft Computing

Tech Science Press

DOI: 10.32604/iasc.2024.060390

ARTICLE

Internet of Things Software Engineering Model Validation Using Knowledge-Based Semantic Learning

Mahmood Alsaadi, Mohammed E. Seno* and Mohammed I. Khalaf

Department of Computer Sciences, College of Sciences, University of Al Maarif, Al Anbar, 31001, Iraq

*Corresponding Author: Mohammed E. Seno. Email: Mohammed.E.Seno@uoa.edu.iq

Received: 31 October 2024 Accepted: 12 December 2024 Published: 10 January 2025

ABSTRACT

The agility of Internet of Things (IoT) software engineering is benchmarked based on its systematic insights for wide application support infrastructure developments. Such developments are focused on reducing the interfacing complexity with heterogeneous devices through applications. To handle the interfacing complexity problem, this article introduces a Semantic Interfacing Obscuration Model (SIOM) for IoT software-engineered platforms. The interfacing obscuration between heterogeneous devices and application interfaces from the testing to real-time validations is accounted for in this model. Based on the level of obscuration between the infrastructure hardware to the end-user software, the modifications through device replacement, capacity amendments, or interface bug fixes are performed. These modifications are based on the level of semantic obscurations observed during the application service intervals. The obscuration level is determined using knowledge learning as a progression from hardware to software semantics. The results reported were computed using specific metrics obtained from these experimental evaluations: an 8.94% reduction in interfacing complexity and a 15.04% improvement in integration progression. The knowledge of obscurations maps the modifications appropriately to reinstate the agility testing of the hardware/software integrations. This modification-based semantics is verified using semantics error, modification time, and complexity.

KEYWORDS

Interfacing complexity; IoT; semantics assessment; software engineering

1 Introduction

In the Internet of Things (IoT) software engineering, creating robust models is crucial for navigating the intricacies inherent in IoT infrastructure. These models act as guiding frameworks, aiding the seamless integration and management of diverse devices and application interfaces [1,2]. They streamline development and boost system flexibility by offering systematic insights and methodologies. Key focuses involve tackling interfacing hurdles, optimizing communication protocols, and ensuring scalability across various IoT ecosystems [3]. Through continual refinement and adaptation, these models enable ongoing enhancement and innovation in IoT software engineering practices [4]. Evaluation criteria like performance, scalability, and interoperability serve as yardsticks for gauging the efficacy of these models. Ultimately, their successful implementation propels the progress and evolution of IoT technologies, ushering in a new era of interconnected and intelligent systems [5,6].

In the intricate domain of IoT software engineering, application interfaces are crucial as conduits, enabling seamless interaction among various heterogeneous devices and end-users. These are essential for orchestrating intuitive control, monitoring, and data exchange within expansive IoT ecosystems [7,8]. Through meticulous design and optimization, application interfaces cater to diverse user preferences and device functionalities, enhancing the overall user experience [9]. Adaptability is paramount, with interfaces continuously evolving to meet the dynamic demands of the ever-expanding IoT landscape and its myriad use cases [10]. Ensuring robust security measures within application interfaces is crucial, safeguarding sensitive data and preventing potential cyber threats. As the IoT ecosystem grows, the significance of user-friendly and secure application interfaces becomes increasingly evident, driving ongoing innovation within software engineering practices [11,12].

In IoT software engineering, optimization methods are pivotal for enhancing system performance and efficiency. These encompass a range of techniques targeting resource utilization, communication protocols, and data processing algorithms [13]. Common approaches include algorithmic optimization, compression algorithms, and parallel processing to improve efficiency [14]. Energy-efficient design strategies are also employed to extend device battery life. Real-time optimization algorithms allow dynamic adjustments based on changing conditions. Continuous monitoring and feedback mechanisms ensure ongoing optimization to adapt to evolving requirements [15]. This provides the resilience and effectiveness of IoT solutions in a rapidly changing technological landscape. Furthermore, optimization methods often involve fine-tuning parameters and configurations to optimize system performance. Machine learning algorithms are increasingly important in this optimization process, allowing systems to adapt and improve based on historical data and real-time feedback [16,17]. As a continuity of the above discussion, this article discusses the integration issues and complexity-causing factors through the Semantic Interfacing Obscuration (SIO) model. Thus, a short introduction to the article's contribution is presented below:

This article introduces a semantic interfacing obscuration model to aid testing and validation support for IoT-based hardware and software. The proposed model uses a knowledge-learning paradigm to identify the progression and complexities in different service intervals. It is evaluated using an experimental analysis presenting the metrics as a comparative analysis.

In the study, "obscuration" refers to the semantic mismatch or inconsistency arising in interactions between heterogeneous IoT devices, applications, and interfaces. It involves errors, delays, and inefficiencies brought about by differences in protocols, data formats, or device capabilities that prevent seamless integration.

The term "obscuration" has been a deliberate choice because of its relevance to such challenges, extending beyond the mechanical connotation that would be implied by terms such as "interfacing complexity" or "integration challenge". While those other options significantly emphasize structural or technical hurdles, "obscuration" greatly emphasizes semantic and operational barriers and thus would imply adaptive resolution mechanisms to restore the state of transparency in interactions.

2 Related Works

Cao et al. [18] developed a software-based remote attestation scheme for Class-1 IoT devices. The scheme includes mechanisms like delayed observation and memory filling to enhance efficiency.

The aim is to optimize attestation processes and withstand proxy attacks. Testing on a UNO-R3 development board demonstrated its practicality and effectiveness.

Herrera et al. [19] proposed enhancing response time in a software-defined network (SDN)-Fog environment for time-sensitive IoT applications. The approach identified optimal deployments for distributed applications, improving the Quality of Service. Utilizing the DADO framework, optimization of computational elements and SDN controller placements was achieved. Scalable deployments reduced response times by up to 37.89% compared to alternatives and up to 15.42% compared to benchmarks.

Zhu et al. [20] introduced a deep reinforcement learning-based algorithm for edge computing offloading in software-defined IoT. The goal was to optimize task offloading from IoT devices to edge servers, achieving global optimization. This is implemented as a software-defined edge computing architecture and an edge computing offloading algorithm based on deep reinforcement learning. The algorithm sped up tasks, saved energy, and improved balancing and finishing tasks.

Alulema et al. [21] developed SI4IoT, a method for integrating IoT systems. SI4IoT simplifies IoT system development using Model-Driven Engineering, aiding developers in navigating the complex IoT environment. The method enables users to access home information on television through tailored REST services for IoT nodes, enhancing user engagement. SI4IoT combines models and services to offer developers a versatile tool for managing diverse IoT environments efficiently.

Zhou et al. [22] introduced a smart library architecture that merges IoT and SDN. The method utilizes SDN to cut costs and enhance network management. Passive Radio Frequency Identification (RFID) tags are used for efficient book and library property management. The method's performance is assessed through both real-world deployment and computer simulations.

Zafar et al. [23] introduced PBCLR to reduce the control-plane load in a software-defined IoT network. Dynamic switch migration is proposed to handle the load, addressing inefficiencies during controller overload. The method proactively migrates switches with anticipated high traffic to an alternative control. Learning techniques and time-series analysis predict future workload based on historical data to inform migration decisions.

Moin et al. [24] created ML-Quadrat, a model-driven approach for IoT machine learning and software modelling. SE models plan architecture at different levels, dealing with concerns throughout software development. AI models enable smart capabilities such as prediction and decision-making, particularly in Machine Learning (ML). ML-Quadrat is implemented based on ThingML and validated through a case study from the IoT domain and empirical user evaluation.

Bou Ghantous et al. [25] introduced evaluating the DevOps reference architecture (DRA) for multi-cloud IoT applications. Originating from agile development, DevOps targets continuous software deployment in small releases. The DRA enables architects to design complex models for automated software development. Software engineers can deploy intricate IoT applications in multi-cloud environments using DRA models within existing organizational parameters.

Chen et al. [26] devised user integration using the evaluation grid method in two IoT sustainable services. The method offers a systematic framework to identify user desires from diverse stakeholders, guiding technological development. Through in-depth interviews, the evaluation grid method explores users' desires, visualizing them as a hierarchical evaluation map of attraction. An IoT prototype is constructed to gather insightful feedback, adhering to minimum viable product design principles.

Gaglianese et al. [27] suggested assessing and improving a Cloud-IoT monitoring service across federated testbeds. The evaluation spanned 20 to 40 nodes across two Fed4Fire+ federation testbeds.

Following the assessment, FogMon was upgraded to FogMon2, achieving TRL5 and improving monitoring accuracy and fault resiliency. The findings revealed that FogMon2 handles infrastructure failures well, with just a 10% measurement error and minimal impact on hardware and network resources.

Ali et al. [28] introduced a controller placement method for SD-IoT utilizing the Analytical Network Process (ANP). Their approach exceeded the performance of the standard k-means method. The technique effectively reduced delays and communication overhead. It also ensured fair switch allocation and minimized controller-to-controller delays.

Zhang et al. [29] developed human body IoT systems utilizing the triboelectrification effect. Their method relies on improved TENG designs and materials for flexibility and stability. TENG-based systems are proposed for monitoring human motion and physical status, utilizing wearable and implantable sensors. The method offers promising avenues for health monitoring and wearable technology.

Işıkdağ et al. [30] introduced IoT architecture to integrate geoinformation efficiently. Their approach prioritizes integration to manage and transfer Geoinformation in diverse IoT environments. The method tries various technologies and integration methods using an IoT Integration Testbed Architecture with inexpensive hardware, graph databases, and standard IoT protocols. Results show that handling multi-source Geoinformation in IoT environments is feasible, providing a basis for future development.

Deng et al. [31] introduced FogBus2, a lightweight and distributed container-based framework for integrating IoT-enabled systems. FogBus2 schedules heterogeneous IoT applications with various policies and uses an optimized genetic algorithm for fast convergence. The method ensures scalability for efficient responsiveness as the number of IoT devices increases. FogBus2 enhances IoT app response time by 53% and cuts queuing waiting time by up to 48%.

Villegas-Ch et al. [32] proposed integrating IoT and Blockchain in university campus processes to enhance sustainability. This concept improves decision-making and ensures the security and efficiency of processes and data. Blockchain technology is analyzed to create a new layer within university architecture, prioritizing data privacy. Integration verifies processes and enhances security within the university environment.

The SIO model is designed to improve the software engineering model validation of hardware devices based on application and user interfaces. The application interfaces are obtained from the IoT platform (i.e., the level of obscuration observed in random time intervals for accurate software validation). This proposed modelling aims to reduce the problem of interfacing complexity in identifying capacity changes. The challenging task is device replacement and bug fixes based on the obscuration level with the previous software engineering model.

Table 1 focuses on some critical recent advances concerning IoT, manufacturing, and software engineering, comparing innovative frameworks and algorithms, techniques achieved, and limitations to pave the way for future research.

Table 1: Comparison of proposed frameworks and techniques across IoT, manufacturing, and software engineering domains

Author name	Proposed name	Technique used	Results	Limitations
Lee et al. (2023) [33]	SEIF: Semantic-enabled IoT service framework	Utilizes semantic knowledge graphs and data ontologies to enable interoperable data and knowledge retrieval for IoT systems.	Improved IoT service interoperability with precise semantic mappings; reduced data retrieval errors.	High computational requirements for semantic reasoning; limited scalability for enormous datasets.
Su et al. (2024) [34]	Knowledge-based digital twin system	A knowledge-driven approach combining digital twins with manufacturing process modelling for dynamic and intelligent updates.	Enhanced manufacturing process modelling accuracy; faster dynamic reconfigurations and optimizations.	Implementation complexity in diverse manufacturing environments requires robust and precise data input.
Pandit et al. (2022) [35]	DePaaS: AI-based software defect prediction framework	Uses AI algorithms like deep learning and statistical analysis for global defect prediction across software systems.	Achieved high defect prediction accuracy across various datasets; improved software reliability.	Limited performance on smaller or highly imbalanced datasets requires extensive labelled data.
Kukkar et al. (2023) [36]	ProRE: ACO-based programmer recommendation model	Applies ant colony optimization (ACO) for recommending programmers to resolve software bugs effectively.	Increased bug resolution efficiency; reduced allocation mismatches in software debugging processes.	It relies on historical data, which can limit adaptability to new programming paradigms or

Current Models and Their Limitations

Interfacing complexity arises in IoT systems because of diverse communication protocols, hardware specifications, and software APIs on heterogeneous devices. Real-world examples include protocol mismatches, such as Zigbee-based thermostats and Wi-Fi-enabled security cameras, which result in delay and compatibility issues. Besides, error propagation due to interface bugs, scalability challenges when new or replacement devices are introduced, and high maintenance overheads further worsen the problem. Traditional models usually cannot dynamically cope with these based on static integration logic, limited semantic awareness, and manual validation processes, which are inefficient for real-time adjustments or ensuring smooth scalability.

The Semantic Interfacing Obscuration Model, presented in this paper, tries to overcome these shortcomings given the following:

The proposed SIOM has used knowledge-based learning to predict and resolve such semantic inconsistencies in real-time. Hardware-software mappings are adjusted to enable seamless device replacements, capacity amendments, and bug fixes with minimal downtime. Progressive adjustments consider only the most critical obscuration points, while automated semantic assessments address interdependencies and minimize computational overheads. SIOM focuses on low-resource adaptive operations while efficiently ensuring real-time validation and integration, overcoming the limitation of static models, and providing scalable and flexible IoT ecosystems.

3 Semantic Interfacing Obscuration Model (SIOM) for Software Engineering Model Validation

The proposed model aims to reduce the obscuration between heterogeneous devices and application interfaces by reducing interfacing complexity in different service intervals. The hardware/software integrations are controlled using knowledge learning for appropriate device replacement and bug fixes that are suppressed to reduce complexity.

To deal with the interfacing complexity of IoT, a fully integrated heterogeneous device and software component has evolved. Current methods have numerous deficiencies regarding semantic inconsistencies, device adaptability, and real-time validation in a dynamic environment. The Semantic Interfacing Obscuration model is proposed to reduce semantic errors, enhance the rate of improvement of the integration process, and facilitate adaptive modifications on various IoT platforms by proposing a new approach to tending a knowledge-learning Semantic assessment model for IoT systems' ability to interface challenges.

3.1 IoT Setup with SIOM

The proposed model can test and validate all software/hardware in all IoT devices. The level of obscuration between the infrastructure hardware and the end-user software is addressed to perform accurate modifications through device replacement, capacity amendments, or interface bug fixes. This modification through IoT prevents complexity problems from interfacing to improve the performance of IoT devices. Fig. 1 presents a schematic illustration of the SIOM in IoT.

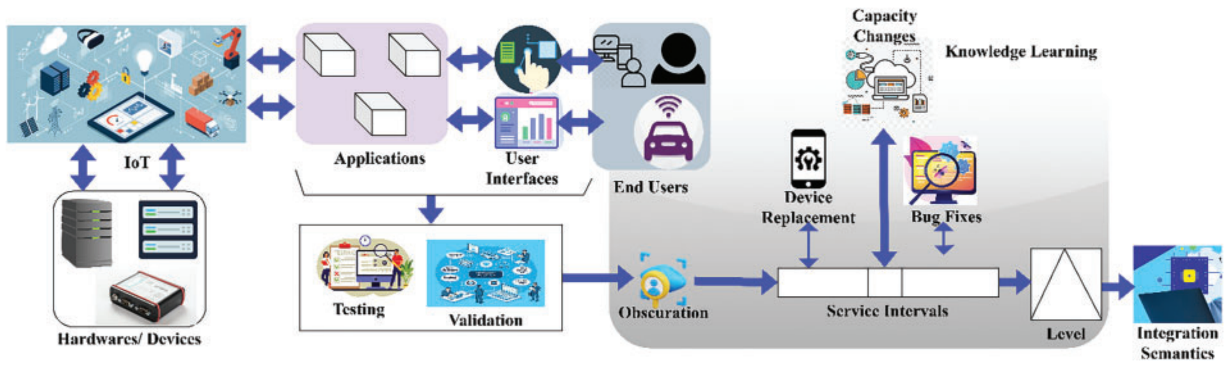


Figure 1: Schematic illustration of SIOM functions in IoT

The IoT technology consists of two types of terminals, such as applications and user interfaces. The application terminal collects input data, whereas the user interface terminal is responsible for sequential data processing and other interfacing complexity problems. Both the applications and user interfaces are used for performing accurate test validation processes (Fig. 1). The IoT software and hardware devices communicate with a set of applications $A(N) = \{1, 2, \ldots, A(n)\}$ and set of user interfaces $UI(N) = \{1, 2, \ldots, UI(n)\}$; these IoT-associated devices can identify interfacing complexity problems at the time of testing and validation process. The application processing is performed for different types of user information in random time intervals $T = \{1, 2, \ldots, i\}$. Let us assume Θ is the number of interfacing complexity occurrences in the IoT network. Based on the instance, the number of testing and validation processes performed per unit of time is ϕ, and from which the interfacing complexity (η_c) is estimated as

$$\eta_c = \{A\,(N) + UI(N) \times \phi \times i \forall A\,(N) + UI(N) :: T, \forall \Theta = 0 \; Obsc_\phi \times \frac{A\,(n) + UI(n) - \Theta}{A\,(N) + UI(N)} \times i \forall\,(A\,(N) + UI\,(N), \Theta) :: T, \forall \Theta \neq 0 \tag{1}$$

$$A\,(N) + UI(N) :: T = \sum\nolimits_{i=1}^{UI(N)} \phi_T \tag{2a}$$

$$(A\,(N) + UI\,(N), \Theta) :: T = \sum\nolimits_{i=1}^{UI(n)} \phi_T - Obsc_\phi \sum\nolimits_{i=1}^{\Theta} \phi_i \tag{2b}$$

where,

$$Obsc_\phi = \frac{\exists_r}{\exists_r + T} \tag{3}$$

In the above equations, the variables $Obsc_\phi$ and $\exists_r$ signify the obscuration occurrence rate and testing validation error rate in random time intervals. In the above equation $A\,(N) + UI(N) :: T$ and $(A\,(N) + UI\,(N), \Theta) :: T$ symbolizes the mapping of user interfaces and interfacing complexity occurring applications in different time intervals T. The obscuration in the IoT network is identified in two levels: testing and validation without integration loss. In the testing and validation process, the gathered data sequence and η_c are the added-up metrics for ensuring that each service satisfies the obscuration occurrence for the mapped time interval. For the collected information, the testing and validation process provides device replacement and bug fixes for each service. The obscuration estimation is described in Algorithm 1.

Algorithm 1: Obscuration estimation description

```
Input: $A(N)$, $T$
$Step\ 1: Initialize\ A\,(N)\ and\ UI\ for\ T$
$Step\ 2: Perform\ mapping\ as :: \pi = \sum_{i=1}^{UI(N)} \phi_T$
$Step\ 3: if\ maping\ is\ not\ done,\ then$
$Step\ 4: No\ integration\ is\ made; [A\,(N) - UI\,(N)] = \Theta$
$Step\ 5: Udpate\ T = T - 1$
$Step\ 6: \eta_c = \frac{A\,(n) + UI\,(n) - \Theta}{A\,(N) + UI(N)}$
$Step\ 7: else\ if\ mapping\ is\ done,\ then$
```

(Continued)

Algorithm 1 (continued)

```
Step 8: $\pi = [A(N) + UI(N)] - \phi$ is the update
Step 9: Compute $obsc_\phi = \frac{\exists_r}{\exists_r + T}$
Step 10: Repeat from Step 3 $\forall\ \Theta = 0$ is achieved
Step 11: End
Output: $Obsc_\phi$
```

The device replacement between $A(n) + UI(n) \in A(N) + UI(N)$ and Θ are processed based on identifying capacity changes and time mapping. In above Eq. (1), the condition of $\Theta > A(n) + UI(n)$ generates integrity loss from the hardware devices. The time-mapping for all the software/hardware devices and the sequential η_c based on $(A(n) + UI(n) \times \phi)$ are the testing and validation process conditions for precise obscuration identification

$$\Delta_{map} = \sum_{i=1}^{UI(n)} \frac{\complement_{asmt}}{T_i} \tag{4}$$

and,

$$\urcorner\eta_c = \frac{\eta_c}{(UI(n) - \Theta)} - (\complement_{asmt} - \exists_r) \tag{5}$$

In the above equation, Δ_{map} and $\urcorner\eta_c$ is the time mapping and routine test validation processing instance for IoT software-engineered platforms. The variable $\complement_{asmt}$ signifies conditional assessment. In Table 2, the conditional assessment-based Δ_{map} is presented with the different combinations of $A(n) + UI(n)$.

Table 2: Δ_{map} assessment based on different conditional combinations

Condition	$A(N)$	$UI(N)$	$::T$	$Obsc_\phi$	Δ_{map}
$\boldsymbol{\Theta = 0}$	0	0	Low	0.25	0.6
	0	1	Low	0.28	0.74
	1	0	High	0.08	0.98
	1	1	High	0.095	0.99
$\boldsymbol{\Theta \neq 0}$	0	0	Low	0.15	0.72
	0	1	Low	0.21	0.78
	1	0	Low	0.15	0.81
	1	1	High	0.098	0.978

In Table 2 above, the $\Theta = 0$ and $\Theta \neq 0$ are the conditions analyzed for the integration components: $A(N)$ and $UI(N)$. If $A(N) = 1$ and $UI(N) = 1$, then the application and its interface merge to ensure ϕ for different T. Based on the mapping defined in Algorithm 1, the T is defined using $-\phi$ instances for reducing $\exists_r$. The $obsc_\phi$ is defined for the unmapped π such that $\Theta = 0$ is achieved at any testing instances. Therefore as $obsc_\phi$ is low, the mapping between interface and application is achieved to maximize the integration semantics. The interfacing complexity/obscuration between application interfaces and heterogeneous devices from the testing to real-time validations is detected in this proposed model. From Eqs. (1) to (5), the accurate and appropriate test validation of the hardware

$\alpha(Test_V)$ is estimated for each service in different time intervals. This validation is performed to identify the conditions of $\Theta \neq 0$ and $\Theta = 0$ based on the level of obscurations in all the hardware/devices through device replacement and bug fixes. The level of obscurations between infrastructure hardware and the end-user software modifications for capacity amendments, device replacement, or interface bug fixes. The service intervals are dependent on validation sequences of Δ_{map} and $\urcorner\eta_c$ from which $\alpha(Test_V)$ is determined in all the mediate layer outputs (M_O) for integrity loss detection. The linear output of $\urcorner\eta_c$ in Δ_{map} is the validating instance for maximizing $(A\,(n) + UI(n) \times \phi)$ condition. The M_O and knowledge learning output (KL_O) is difficult to determine $\alpha(Test_V)$. The input validation of η_c for both $A\,(N) + UI(N) :: T$ and $(A\,(N) + UI\,(N), \Theta) :: T$ mappings are to identify obscurations in any hardware/devices. The different test validation-based complexity assessment is presented in Table 3 with the $Obsc_\varnothing$ rate.

Table 3: Complexity assessment for different test validations

Test	$\boldsymbol{Obsc_\phi}$ rate	$\boldsymbol{\alpha\,(Test_V)}$	$\complement_{asmt}$ (%)	$\boldsymbol{\eta_c \forall \Theta = 0}$	$\boldsymbol{\eta_c \forall \Theta \neq 0}$
Hardware	0.1	1	39.28	0.0282	0.131
	0.2	0	41.21	0.0321	0.127
	0.3	0	65.1	0.041	0.085
Software	0.1	1	42.56	0.0298	0.095
	0.2	1	52.36	0.0321	0.087
	0.3	0	74.1	0.0451	0.071
Interface	0.1	1	0.51.25	0.051	0.103
	0.2	1	58.69	0.032	0.098
	0.3	0	65.21	0.041	0.085
Bug fixes	0.1	1	55.36	0.048	0.112
	0.2	0	62.21	0.044	0.089
	0.3	0	71.24	0.054	0.074
Communication	0.1	1	59.36	0.0325	0.116
	0.2	0	66.321	0.0489	0.102
	0.3	0	74.21	0.0531	0.089

The testing is considered for hardware (device, functioning), Software (installation and usage), interface (access and response), bugfixes (errors), and communication (connectivity). Each test is unique and is validated by a $obsc_\phi$ rate ranging between 0.1 and 0.3. The $\alpha\,(Test_V) = 1\,(high)$ for low $Obsc_\phi$ rate that influences C_{asmt}. If the test is low (0), then conditional assessment increases for the chances of $\Theta = 0$ are less than $\Theta \neq 0$. Therefore, the number of audit chances for validation is high to ensure a low η_c. Thus, both cases are utilized eventually to maximize the assessments over test cases (Table 3).

3.2 Knowledge Learning Process

The knowledge learning implemented for both the applications and user interfaces is different for the conditions $\Theta \neq 0$, $\urcorner\eta_c = (A\,(n) + UI(n) - \Theta)\eta_c$ and $\complement_{asmt}$. If the hardware devices available in the mapped timing are accounted for integrity level verification, it is output as 1 or 0. The M_O in the first mapping of $A\,(N) + UI\,(N) :: T$ provides a linear finite outcome, whereas $(A\,(N) + UI\,(N), \Theta) :: T$

extracts output of $A(n) + UI(n)$ from $A(N) + UI(N)$ with $\Theta \neq 0$. In Eqs. (6) and (7), the two outputs for $A(N) + UI(N) :: T$ is estimated. The test validation is performed for the identification of $Obsc_{\phi}$ and $\exists_r$ and the conditional assessment of $\complement_{asmt} = 0$ or $\complement_{asmt} = 1$ in different time intervals. Therefore, the output is obtained from all the hardware/devices in validation time interval T. In the above test validation process, Θ serves as an input to the knowledge learning after the identification of $Obsc_{\phi}$ in $A(N) + UI(N) :: T$ condition

$$M_{O_1} = \neg\eta_{c_1} * i_1 + \phi_1\complement_{asmt_1} M_{O_2} = \neg\eta_{c_2} * i_2 - \exists_{r_1} + \phi_2\complement_{asmt_2} M_{O_3} = \neg\eta_{c_3} * i_3 - \exists_{r_2} + \phi_3\complement_{asmt_3} \vdots M_{O_T} = \neg\eta_{c_T} * i_T - \exists_{r_{T-1}} + \phi_T\complement_{asmt_T}\} \quad (6)$$

$$KL_{O_1} = M_{O_1} - \phi_1\, KL_{O_2} = M_{O_2} - Obsc_{\phi_1}\phi_2\, KL_{O_3} = M_{O_3} - Obsc_{\phi_2}\phi_3 \vdots KL_{O_T} = M_{O_T} - Obsc_{\phi_T}\phi_{T-1}| \; KL_{O_1} = \neg\eta_{c_1} * i_1 + \complement_{asmt_1} - Obsc_{\phi_1}\, KL_{O_2} = \neg\eta_{c_2} * i_2 - \exists_{r_1} + \phi_1\complement_{asmt_2} - Obsc_{\phi_2}\phi_1\, KL_{O_3} = \neg\eta_{c_3} * i_3 - \exists_{r_2} + \phi_2\complement_{asmt_3} - Obsc_{\phi_3}\phi_2 \vdots KL_{O_T} = \neg\eta_{c_T} * i_{T-1} - \exists_{r_T} + \phi_T\complement_{asmt_{T-1}} - Obsc_{\phi_{T-1}}\phi_T\} \quad (7)$$

In the above equations, the linear output of the testing and validation process is expressed as $KL_{O_T} = \neg\eta_{c_T} * i_{T-1} - \exists_{r_T} + \phi_T\complement_{asmt_{T-1}} - Obsc_{\phi_{T-1}}\phi_T$ with $\Theta = 0$, $\complement_{asmt} = 1$ and $\neg\eta_{c_T} = A(N) + UI$. Hence, the optimal output with $\alpha(Test_V) = 1$ for identifying integration loss. Therefore, such software and hardware testing is validated as 1 to identify knowledge-based semantic obscurations. The learning representation for $\alpha(Test_V) = 1$ is illustrated in Fig. 2.

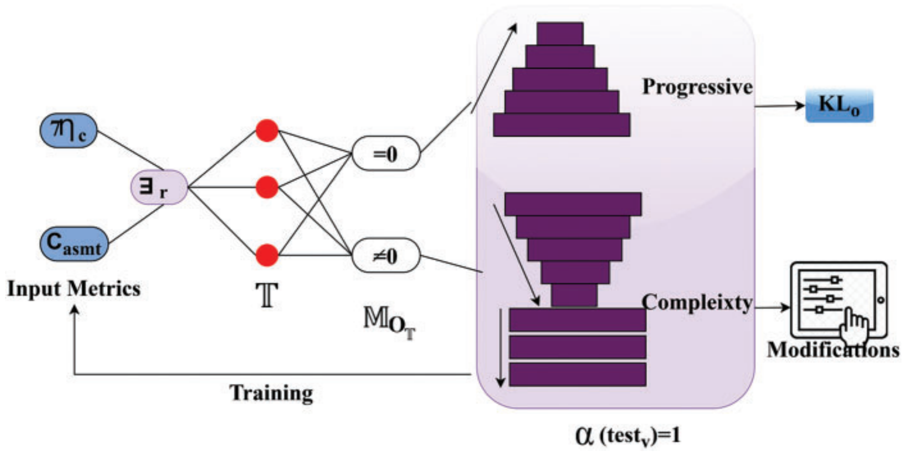


Figure 2: Learning model representation for $\boldsymbol{\alpha}(\boldsymbol{Test_V}) = \mathbf{1}$

The learning model considers $\neg\eta_c$, $\complement_{asmt}$, and $\exists_r$ inputs for π intervals for M_{O_T} intermediates. The intermediates are segregated as $= 0$ or $\neq 0$ based on Θ and its corresponding Δ_{map} factor. If $M_{O_T} = 0$ progressive knowledge is experienced failing, which requires modifications. Based on the recommended modifications, the complexity of the previous M_{O_T} instances are computed. As the complexity is high, the modifications for $A(N)$ and $UI(N)$ are initialized. Thus both $\alpha(Test_V) = 1$ outcomes are used to train the input metrics with updates (Fig. 2). The IoT stores $(\alpha(Test_V), UI(N))$ for each T, the service interval determines the testing for wide application support infrastructure developments. Instead, $(A(N) + UI(N), \Theta) :: T$ based on M_O and KL_O are estimated as in Eqs. (8)

and (9), respectively.

$$M_{O_1} = \eta_{c_1} M_{O_2} = \eta_{c_2} + \complement_{asmt} * Obsc_{\phi_1} - \exists_{r_1}\phi M_{O_3} = \eta_{c_3} + \complement_{asmt} * Obsc_{\phi_2} - \exists_{r_2}\phi \vdots$$
$$M_{O_T} = \eta_{c_T} + \complement_{asmt} * Obsc_{\phi_{T-1}} - \exists_{r_T}\phi\} \tag{8}$$

$$KL_{O_1} = M_{O_1} = \eta_{c_1} KL_{O_2} = M_{O_2} + \Delta_{map_1} - \urcorner\eta_{c_1} = \eta_{c_2} - \complement_{asmt} * Obsc_{\phi_1} - \exists_{r_1}\phi_1 + \Delta_{map_1}$$
$$KL_{O_3} = M_{O_3} + \Delta_{map_2} - \urcorner\eta_{c_2} = \eta_{c_3} - \complement_{asmt} * Obsc_{\phi_1} - \exists_{r_2}\phi_2 + \Delta_{map_2} \vdots KL_{O_T} = M_{O_T} + \Delta_{map_T}$$
$$- \urcorner\eta_{c_T} = \eta_{c_T} - \complement_{asmt} * Obsc_{\phi_T} - \exists_{r_T}\phi_T + \Delta_{map_{T-1}}\} \tag{9}$$

As per the Eqs. (8) and (9), the level of obscurations is obtained by testing validation conditions of $\urcorner\eta_c = (A(n) + UI(n) - \Theta)$ and $\complement_{asmt} = 1$ or $\complement_{asmt} = 0$ in by one manner for identifying modifications. If $\complement_{asmt} = 0$ then $KL_O = M_{O_T} + \Delta_{map_T} - \urcorner\eta_{c_T} = \eta_{c_T}$ is verified at the end of all services, and if $\complement_{asmt} = 1$ then $\exists_r = 0$, and hence, the last output is $KL_O = \alpha(Test_V - \eta_c$. Therefore, the condition of $(A(N) + UI(N), \Theta) :: T$ is the precise output for identifying integrity levels and thereby finding out modifications for this output, $\alpha(Test_V) = \left(\frac{\complement_{asmt} - \exists_r \times Obsc_\phi}{A(n)+UI(n)}\right)$ the semantic assessment output is modified with all the mediate layer outputs and knowledge learning outputs as in above Eqs. (8) and (9). This condition is not applicable for the first semantic assessment as in Eqs. (6) and (7). Because the above condition relies on all mapped applications and user interfaces at different time intervals. Therefore, the $\alpha(Test_V)$ along with Δ_{map} and $\urcorner\eta_c$ is validated by the IoT. The $Obsc_\varnothing$ level variations for the $\alpha(Test_V) = 0$ is described in Algorithm 2.

Algorithm 2: $Obsc_\varnothing$ **level variations for** $\alpha(Test_V) = 0$

```
Input: $M_{O_T}, \phi$
Step 1: Initialize $(T, I, A(N))$
Step 2: Compute $M_O$ for all the $T$ Intervals
Step 3: if $I$ is active for all $A(N)$ under $\Theta = 0$ then
Step 4: $\eta_c = 0; \complement_{asmt} = 0;$ increment $A(N)$ with $\Delta_{map}$
Step 5: Update $\alpha$ $(Test_V) = 1 \ \forall \ I$ and $A(N)$
Step 6: else if $I$ is active for all $A(N)$ under $\Theta \neq 0$ then
Step 7: compute $\exists_r \frac{Obsc_\phi - \Theta}{T}$ for the remaining integrations
Step 8: $KL_O = M_O + \Delta_{map} - \urcorner\eta_c - \complement_{asmt} \forall Obsc_\phi \forall \complement_{asmt} \neq 0$
Step 9: $M_O = \eta_c + \complement_{asmt} - \frac{\exists_r.\phi}{T}$
Step 10: End if
Step 11: Repeat from Step 3 until $A(N) + I = UI(N) \ \forall \ \Theta = 0$
Step 12: $\exists_r\phi = M_O - \frac{\eta_c}{T}$ is the final update
Step 13: End
Output: $\exists_{r_T}\phi_T$
```

Hence, the semantic interfacing obscuration remains unchanged. From the instance, the following sequence of testing and validation process, $\alpha(Test_V)$ on its previous time interval determines the level of obscuration between acquiring application interfaces. If the sequence is required from the condition $\Theta > A(n) + UI(n)$, then the software/hardware testing validation is halted to prevent obscuration. The proposed model focuses on reducing the interfacing complexity problem in IoT software-engineered platforms to ensure appropriate real-time validations to address the obscuration. The application support in the IoT software engineering model relies on semantic assessment. This prevents interfacing complexity problems and testing errors by modifying service intervals for device replacement or bug fixes. Hence, the integrity loss is high. The controlled semantic interfacing obscuration ensures less integration loss within the IoT platform. However, the chances for application interface modification in the IoT platform are high.

3.3 Semantic Integration Assessment

In the semantic assessment, the modification based on the level of obscuration at the time of application service intervals follows the complexity of application and user interfaces. The application interfaces from the testing relay on $(\alpha(Test_V), A(N), UI(N))$ for identifying changes in capacity. Through obscuration level-based software engineering, model validation is administered based on $\alpha(Test_V)$, modifying levels through infrastructure hardware and end-user software is still vulnerable in real-time validations. This concerns semantic assessment for application interfaces in an end-to-end manner to mitigate complex interfacing problems. This semantic integration is administered concurrently based on the device replacement and bug fixes. In this condition, the second level of the integration process is administered to detect the level of obscuration between the infrastructure hardware and the end-user software. In this semantic assessment, the obscuration level in testing is exchanged between the application interfaces.

The agility of IoT software engineering is observed through knowledge learning. The observations are classified as testing and validation. Device replacement and bug fixes reduce the chance of interfacing complexity by causing testing or validation errors. The testing errors are observed as instances of misdetection of capacity changes. The proposed model focuses on such testing errors through knowledge learning and mediate layer output. Initial proposed Modeling of application in Let $SAI(T)$ denote the sequence of software engineering models observed in different time intervals such that the semantic integration $N(I)$ is given as

$$N(I) = SAI(T) - \exists_r \times Obsc_\phi \text{ such that } arg \sum \exists_r(I) \forall SAI(T)\} \tag{10}$$

In Eq. (10), the testing and validation error and the objective of minimizing semantic interfacing obscurations for all $SAI(T) \in N(I)$ is defined as addressing the level of obscuration. The $N(I)$ feasibility and its related complexity based on two conditions for different tests are presented in Table 4.

Table 4: ***N*** **(*I*)** feasibility and complexity analysis

Conditions	M_{O_T}	Hardware		Software		Interface		Bug fixes		Communication	
		Feasibility	$N(I)$ (%)	Feasibility	$N(I)$ (%)	Feasibility	$N(I)$ (%)	Feasibility	$N(I)$ (%)	Feasibility	$N(I)$ (%)
$\boldsymbol{\Theta = 0}$	0.2	0.65	65.2	0.74	75.21	0.79	0.77	0.8	0.75	0.78	0.74
	0.4	0.58	62.36	0.71	73.52	0.75	0.75	0.59	0.74	0.75	0.65
	0.6	0.55	64.21	0.52	70.12	0.62	0.65	0.58	0.73	0.65	0.68
	0.8	0.52	62.3	0.65	65.21	0.52	0.58	0.45	0.67	0.55	0.59
	1	0.48	61.2	0.62	65.2	0.48	0.45	0.42	0.65	0.45	0.55
$\boldsymbol{\Theta \neq 0}$	0.2	0.45	65.14	0.52	45.21	0.65	0.52	0.52	0.37	0.52	0.52
	0.4	0.52	66.34	0.55	52.14	0.71	0.58	0.65	0.45	0.47	0.63
	0.6	0.52	74.1	0.63	48.25	0.85	0.62	0.74	0.56	0.51	0.52

(Continued)

Table 4 (continued)

Conditions	M_{O_T}	Hardware		Software		Interface		Bug fixes		Communication	
		Feasibility	$N(I)$ (%)	Feasibility	$N(I)$ (%)	Feasibility	$N(I)$ (%)	Feasibility	$N(I)$ (%)	Feasibility	$N(I)$ (%)
	0.8	0.46	72.3	0.52	52.1	0.65	0.71	0.71	0.61	0.48	0.58
	1	0.52	74.3	0.45	58.2	0.55	0.65	0.75	0.48	0.52	0.47

Table 4 data presents the feasibility and $N(I)$ values for $\Theta = 0$ and $\Theta \neq 0$ under M_{O_T} variants (0.2, 0.4, 0.6, 0.8, and 1.0). This is not unanimous for different test cases and validations as the progressive varies differently for $M_o\forall$ test cases. Therefore, the knowledge of progression/complexity is estimated across multiple T. This is obvious for the $\Theta = 0$ where $\eta_c \in \complement_{asmt} = 0$ and $\eta_c \in \complement_{asmt} \neq 0$ until maximum feasibility is achieved. For the $\Theta \neq 0$ condition, η_c is the observed complexity under either of the below case such that $\alpha\,(test_v) = 0$ is the failed output. Depending on the $A\,(N) + UI\,(N)\,\forall\Delta_{map}$, the successive validations are made, provided the $\exists_r$ is confined. Assume the condition $Cap_{changes}(x)$ for application interface and $Cap_{changes}(y)$ for user interface and the changes in $SAI(T)$ in random time interval is obtained and $\exists_r$ is identified in all application interfaces such that

$$Cap_{changes}\,(x) = A\,(n) + UI\,(n)\,x\colon\, SAI\,(T), \forall\exists_r = 0 \tag{11}$$

and,

$$Cap_{changes}(y) = \frac{\exists_r \times Obsc_\phi}{A\,(n) + UI\,(n)}y\colon\, \Theta * SAI\,(T), \forall\exists_r \neq 0 \tag{12}$$

As presented in Eqs. (11) and (12), the obscuration level is identified in the $A\,(n) + UI\,(n)\,x$ and $\frac{\exists_r \times Obsc_\phi}{A\,(n) + UI\,(n)}y$ are mapped with $SAI\,(T)$ for reducing complexity. Now, based on the level as in Eqs. (11) and (12), Eq. (10) is re-written as

$$N\,(I) = \{Cap_{changes}\,(x) = A\,(n) + UI\,(n)\,x\colon\, SAI\,(T), if\ \exists_r = 0\ Cap_{changes}\,(x) - Cap_{changes}(y)$$

$$= \frac{Obsc_\phi}{A\,(n) + UI\,(n)}y\colon\, SAI\,(T) - \frac{\exists_r}{A\,(n) + UI\,(n)}y\colon\, SAI\,(T), \forall\exists_r \neq 0 \tag{13}$$

This is computed to identify obscuration levels based on integration semantics using knowledge learning. The correlating service interval through the available dataset is performed using knowledge learning. Based on the knowledge of obscurations maps used to appropriately modify the agility testing of the hardware/software integrations for reducing semantic errors. The consecutive hardware and software semantic modification helps to identify the error in testing between hardware/software integrations. Semantic obscurations are observed at different application service intervals. The $Cap_{changes}$ for the x and y is analyzed in Fig. 3.

The $Cap_{changes}$ for x and y for M_{O_T} ranges are analyzed in the above Fig. 3. The proposed SIO model identifies $Obsc_\phi$ in various $[A\,(N) + UI\,(N) :: T]$ and $[A\,(N) + UI\,(N), \Theta]$ mapping instances. The knowledge learning classifies $\alpha\,(Test_V) = 1$ and $\alpha\,(Test_V) = 0$ for progression and complexity-based modifications. In this case, two M_o are identified from Eqs. (6) and (8), respectively, for $::\ T$ and Θ. Such identification eases the x and y $Cap_{changes}$ indifferent intervals (service). Based on the x and y analysis, the $N(I)$ progression is analyzed in Fig. 4.

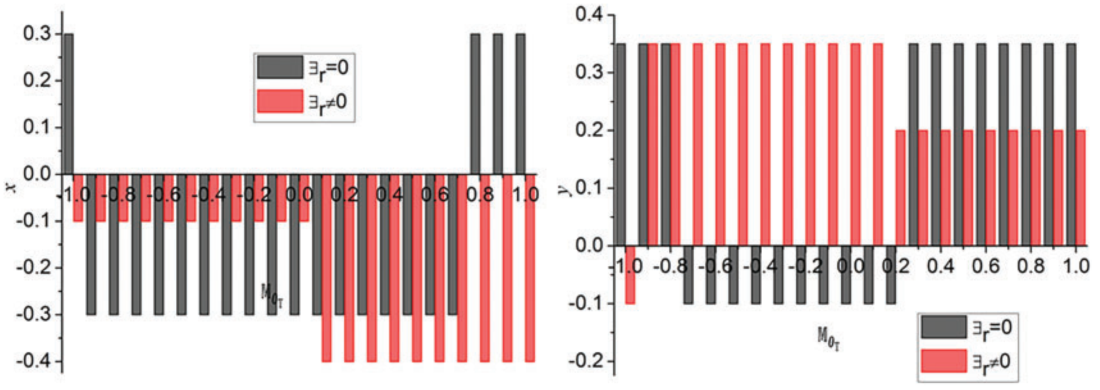
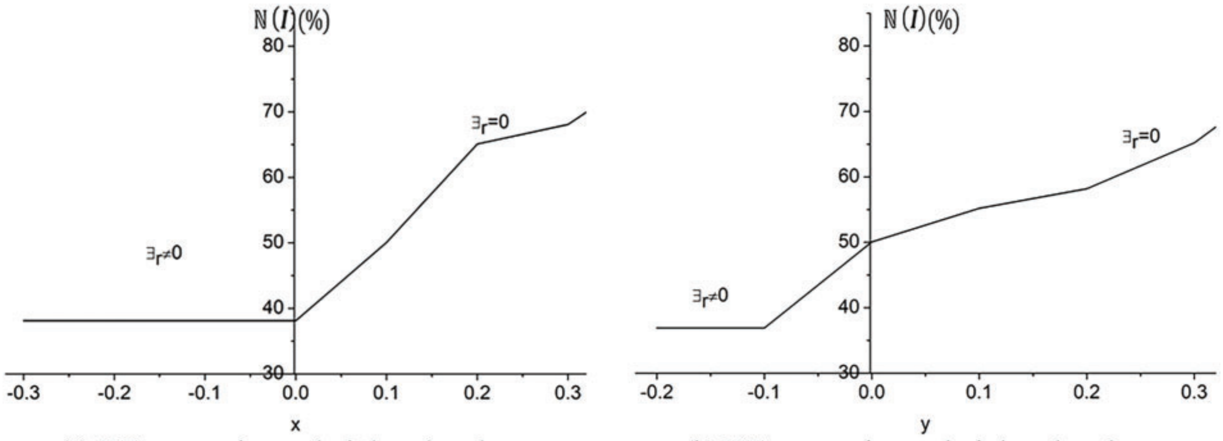


Figure 3: $Cap_{changes}$ for x and y

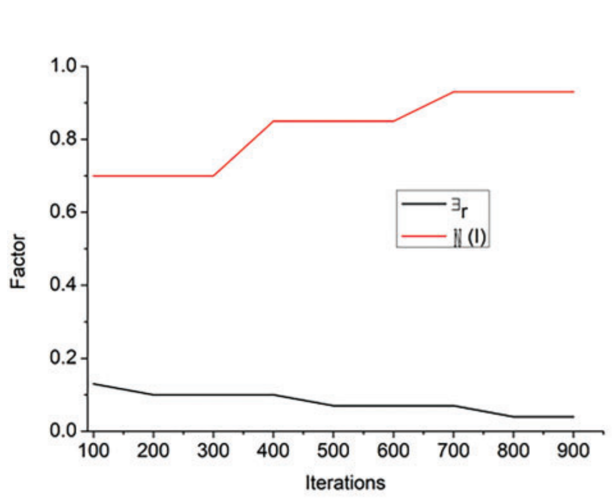
The $N(I)$ progression is estimated for $\exists_r \neq 0$ and $\exists_r = 0$ for various x and y ranges. Depending on the $\alpha\ (Test_V) = 1$ or $\alpha\ (Test_V) = 0$ case differentiations, the KL_o is validated to ensure maximum integration feasibility. Based on $A(N)$ mapping for $UI(N)$, the $\complement_{asmt}$ is utilized for multiple $\urcorner\eta_c$ for maximum semantics validation. Such a verification process leverages the modifications required in handling device integrations (Fig. 4). This reduces $\exists_r$ for different training iterations, maximizing semantics. The analysis of the same is presented in Fig. 5.



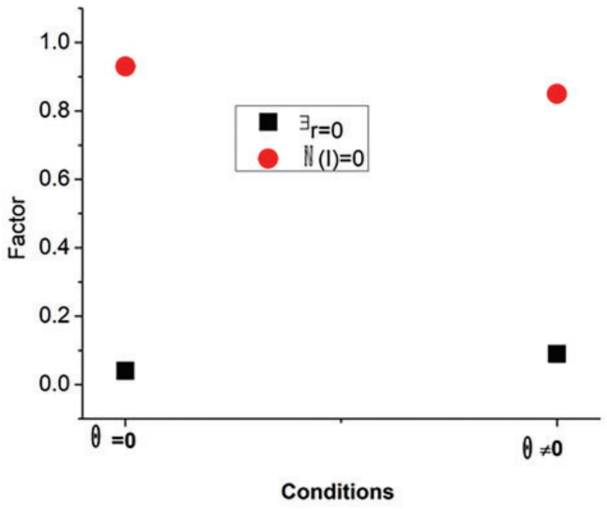
(a) *N(I)* progression analysis based on the *x* (b) *N(I)* **p**rogression **a**nalysis based on the *y*

Figure 4: $N(I)$ progression analyses

For the varying iterations and conditions, the $N\ (I)$ and $\exists_r$ the analysis is presented in Fig. 5. As the iterations increase, the progression and complexity for $M_{O_T} = 0$ and $\neq 0$ are analyzed recurrently to update the knowledge. The knowledge update is pursued for $\Theta = 0$ and $\Theta \neq 0$ conditions to ensure maximum $A\ (N) + I = UI(N)$ interfaces with KL_O or modifications. Therefore, the $\Theta = 0$ case is used to train the $N\ (I)$ factor in maximizing the semantics under fewer η_c.



(a) ∃γ Analysis



(b) *N(I)* Analysis

Figure 5: $\exists_r$ and $N(I)$ analysis

The SIO model generally deals with interfacing complexity issues in IoT systems, relying on the power of knowledge-based learning. It considers necessary steps, such as a) detecting obscuration levels across service intervals, b) adapting changes due to device replacement or bug fixing, and c) assessing using semantics for reducing errors and enhancing integration progress. These elements will orchestrate coherently to ensure smooth interaction among heterogeneous devices and applications, thereby enhancing the agility and reliability of IoT platforms.

4 Results and Discussion

The results and discussion are obtained from a simulation experiment using the Contiki Cooja simulator [37]. This tool uses IoT device interconnection to deploy 8 user interfaces for keying, voice, photo capture, etc. This simulation is modelled to encourage application-specific device support and complexity analysis. The complexity is estimated as the maximum wait time observed after the request generation. The experimental setup is presented in Table 5.

Table 5: Experimental setup

Setting	Value
Devices	240
Service intervals	14
Interval time	30 s minimum and 420 s maximum
Servers	4
Communication bandwidth	1 gbps
Connecting devices	11
Application size	200 to 400 mb

(Continued)

Table 5 (continued)

Setting	Value
Application requests	5/min
Service timeout	15 s

The metrics semantics error, modification time, interfacing complexity, integration progression, and integration semantics are comparatively analyzed using this simulation setup. The integration progression is analyzed using maximum device assimilations to support a single application for a prolonged service interval. The semantics factor is estimated based on the maximum sharing interval provided to a single user. With this information, the existing FogBus2 [31], ECO-SDIoT [20], and FogMon [27] methods discussed in the related works section are used in this comparative analysis.

The word *testing* implies applying a series of planned activities, usually on hardware or software, to determine the correct implementation of specified requirements. Testing detects possible faults or errors in system behaviour. *Validation* proves the system works in real-world operating conditions, fulfilling its purpose dependably and consistently to meet users' needs. *Bug Fixes* refer to actions taken to correct defects or errors found in the system while it is under test or in actual operation.

4.1 Semantics Error

The application and user interface-based information from the IoT devices is obtained and processed through software engineering models. The validation for improving the integration progression achieves fewer semantics errors (as in Fig. 6). In the hardware/devices obscuration identification, the interfacing complexity problem takes place due to application interfaces and heterogeneous devices at different time intervals for performing better test validation. This article detects the variation in capacity measures due to integration loss between the infrastructure hardware using knowledge learning. The capacity changes are addressed based on obscuration level using the conditions of $A(n) + UI(n) \in A(N) + UI(N)$ and Θ to reduce interface bugs. The high or low-capacity changes are detected through semantic learning to satisfy all the modifications through device replacement and bug fixes. The testing and validation process jointly provides the output of high, medium, or low integrity levels at the least possible level of obscuration to improve device capacity with less integration loss. Therefore, the optimal condition here is the IoT software-engineered platform with less semantic error.

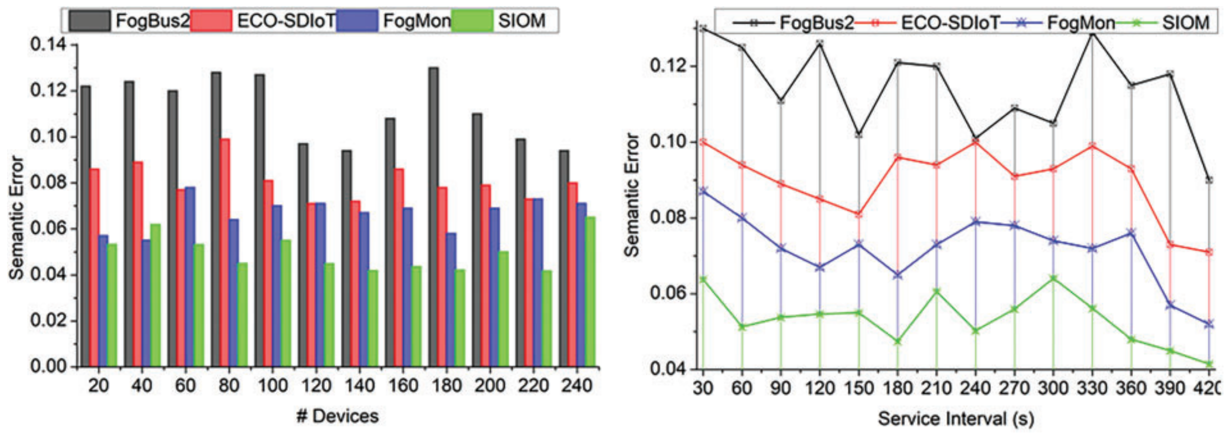


Figure 6: Discussion on semantics error

4.2 Modification Time

The interfacing obscuration includes selective application interfaces from the testing to real-time validations. The complexity problem is addressed to reduce modification time. Hence, device replacement, capacity, and bug fixes are pursued based on the level of obscuration and identification, which is constant at the time of modifications. In this case, the capacity change from the testing to validation is identified to achieve less modification time. The SIOM is designed to achieve high integration progression and semantics for device replacement and bug fixes in an IoT software-engineered platform. Hence, high integration semantics is achievable. Based on the level of obscuration, accurate modification is made using knowledge learning. From the instance, the available possibilities for capacity changes are identified to satisfy the sequential η_c based on $(A\ (n) + UI(n) \times \phi)$ for reducing sintegration loss and semantic errors. The obscuration in the IoT network is identified from the testing to validation without integration loss. In this scenario, the gathered data from IoT hardware/devices are analyzed to ensure the obscuration occurrence identification in the mapped time interval at each service. The identification of capacity changes and time mapping with less modification time and error is represented in Fig. 7.

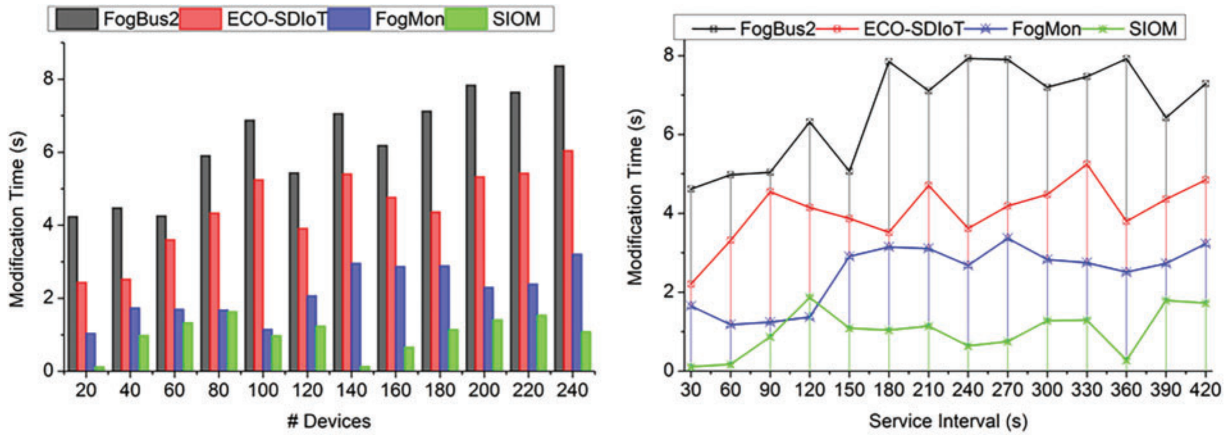


Figure 7: Discussion on modification time

4.3 Interfacing Complexity

The SIOM achieves less interfacing complexity for different service intervals through knowledge learning. The learning eases the identification of the obscuration level in hardware/devices. The interfacing obscuration is identified to merge all the service intervals observed from the testing process for accurate device replacement and capacity amendments. However, the obscuration levels are initially filtered to mitigate interfacing complexity through knowledge learning. Therefore, additional validations are performed to address interfacing complexity problems in the hardware/devices to increase integration progression. To prevent semantic errors, the learning is implemented to identify obscurations in each service to stop interfacing obscuration between the infrastructure hardware and the end-user software. The least integration loss possibilities are detected from the recurrent test validation process through the proposed model and learning. Therefore, the high changes identified service intervals are independently segregated with obscuration levels, leading to less interfacing complexity, as illustrated in Fig. 8.

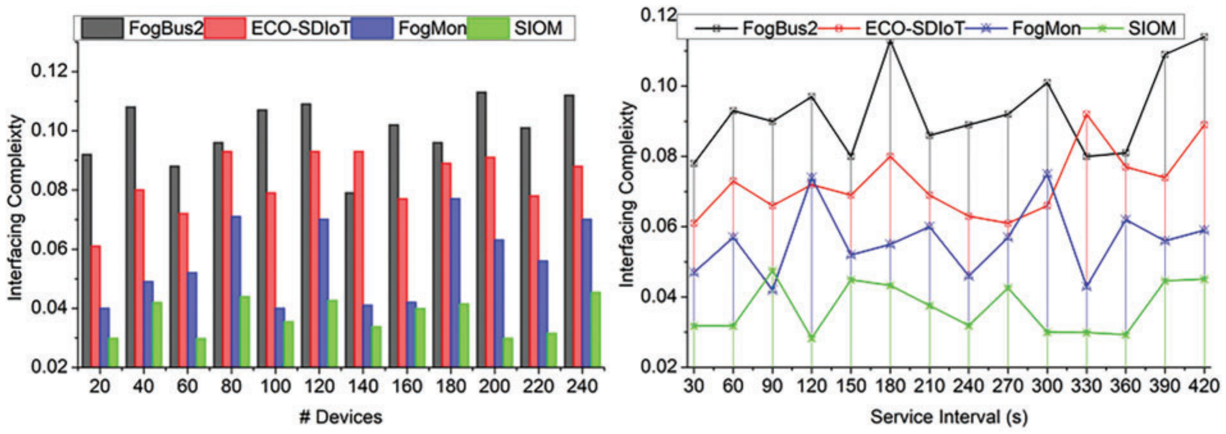


Figure 8: Discussion on interfacing complexity

4.4 Integration Progression

The proposed SIOM improves integration progression for the testing validation process between heterogeneous device and application interfaces (Refer to Fig. 9). The accurate obscuration level identification with fewer semantic errors and modification time is the optimal condition. The condition assessment uses trained inputs and service intervals to provide wide application support with fewer semantic errors. This proposed model identifies the interfacing complexity problem for obscuration levels through knowledge learning. The interfacing complexity/obscuration between application interfaces and heterogeneous devices is identified from the testing to real-time validations for each service to prevent interfacing complexity. Detecting which is constant in a particular service interval is identified for wide application support. The application interfaces observed from the hardware/devices are analyzed at different intervals to prevent interfacing complexity. The interfacing obscurations are identified with previous information using knowledge learning to reduce modification time. Hence, high integration progression is achieved.

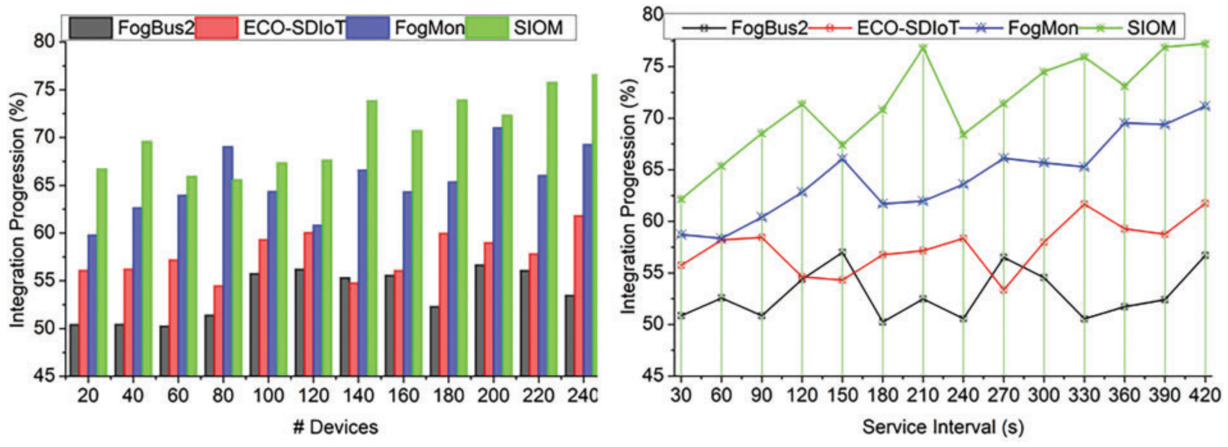


Figure 9: Discussion on integration progression

4.5 Integration Semantics

The application interfaces and heterogeneous device integrations are analyzed for obscurations from the testing to validation based on systematic insights to reduce semantic errors. The high integration semantics are achieved through device replacement, interface bug fixes, or capacity amendments based on capacity changes through mediate layer output and knowledge learning output. The interfacing complexity and semantic errors present in IoT software engineering model validation are reduced using the condition $A(N) + UI(N) :: T$ and $(A(N) + UI(N), \Theta) :: T$. In this article, the knowledge learning process initially halted the changes identified capacity hardware and thereby reduced semantic error and modification time. The capacity changes are identified and segregated from the instance through knowledge learning output with less integration loss. The acquired knowledge from the testing to the validation process is taken to perform accurate modifications using the proposed model, resulting in high integration semantics with less modification time. Hence, the condition-succeeding features are recurrently analyzed using filtering conditions to satisfy high precision (Fig. 10). From the above discussion, and the summary is presented below:

- The proposed model reduced the semantics error by 8.33%, modification time by 13.59%, and interfacing complexity by 8.94%. This model improved the integration progression by 15.04% and integration semantics by 14.36%. This is under the maximum device count.
- The proposed model reduced the semantics error by 9.83%, modification time by 11.06%, and interfacing complexity by 8.45%. This model improved the integration progression by 14.03% and integration semantics by 11.46%. This is under the maximum service interval time.

4.6 Threats to Validity

Threats to validity were thoroughly discussed to enhance the reliability of the research. The internal validity threats were reduced by standardizing the experimental settings, and the external validity was considered by planning real-world validations. Construct validity was ensured by clearly defining metrics based on prior research. Finally, using multiple performance metrics and cross-verifying results mitigated risks to conclusion validity, ensuring data-driven findings. This holistic approach also increases the confidence in the results of this research and follows best practice guidelines for experimentation in software engineering.

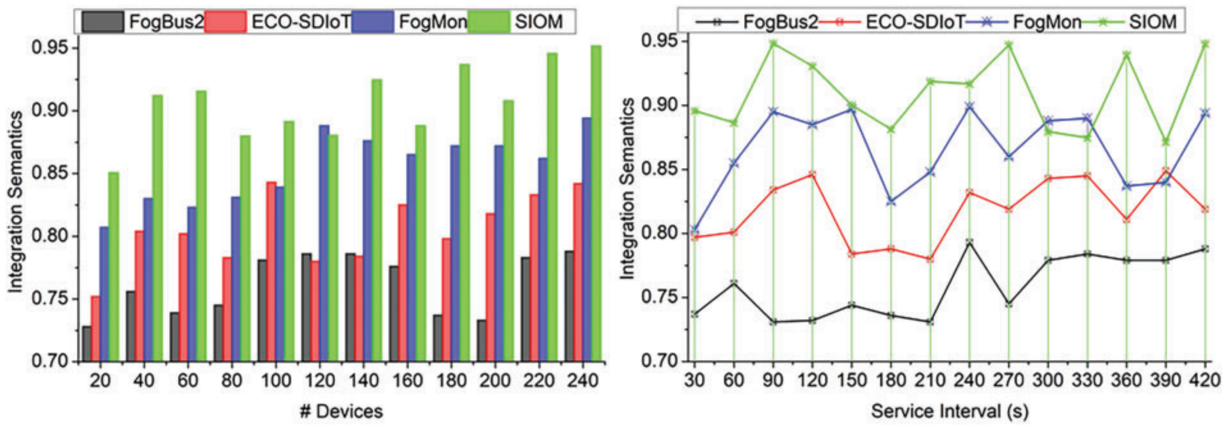


Figure 10: Discussion on integration semantics

Table 6 systematically outlines critical threats to validity and mitigation strategies to ensure rigour in the research. The risks to internal validity were minimized using standardized sets, while external validity was catered for by planning the validations. The construct validity was enhanced by basing the metrics on established literature, and their definitions were outlined. Finally, in conclusion, validity risks were mitigated through multiple metrics and cross-verification of results. This structured approach gives assurance of credible, generalizable, and analytically robust findings for the SIO model.

Table 6: Threats to validity in the SIO model research

Threat type	Description	Mitigation strategies
Internal validity	Potential bias in defining experimental parameters and configurations.	Standardized setups with documented procedures ensured consistency across all simulation environments.
External validity	Limited generalizability due to the controlled nature of simulation environments.	Future validation in real-world IoT deployments to ensure applicability across diverse, unpredictable conditions.
Construct validity	Ambiguity in metrics like interfacing complexity, semantic errors, and integration progression.	Metrics were selected based on established literature and explicitly defined in Section 4 to ensure clarity.
Conclusion validity	Risk of unsupported conclusions due to statistical or analytical errors.	Multiple metrics were used for comparisons with existing methods, with results cross-verified for robustness.

4.7 Summary of This Section

The results reported were computed using specific metrics obtained from these experimental evaluations: an 8.94% reduction in interfacing complexity and a 15.04% improvement in integration progression. To this purpose, interfacing complexity has been quantified by the frequency and severity of semantic errors, obscuration levels, and integration delays in testing and validation processes. The integration progress was measured regarding the number of successful interactions between devices and applications, as well as the consistency of service delivery across different time intervals and device loads.

Experimental Setup:

The study conducted experiments using the Contiki Cooja simulator, which is used for IoT device intercommunication. The simulation environment used was based on 240 devices, with service intervals of 14 random time intervals ranging from 30 to 420 s and a controlled bandwidth of 1 Gbps. The application requests, including voice, photo capture, and so on, were processed to evaluate the obscuration level and the modification efficiency across hardware-software interfaces. Key performance indicators monitored include semantic error rates, modification times, and integration semantics.

Comparative Analysis:

SIOM was compared to established methods, such as FogBus2, ECO-SDIoT, and FogMon, mentioned in related works. These models focus on resource optimization and integration but lack SIOM's knowledge-based semantic learning and adaptive adjustments. The results proved that SIOM consistently outperformed those approaches with higher integration progressions and lower semantic errors, underlining its efficiency in reducing interfacing complexity and enhancing real-time IoT integration.

5 Conclusions

This article proposes the semantic interfacing obscuration model to verify and validate the integration progression of IoT hardware and software. This proposed model addressed the integration complexity problem through testing and validation for real-time application support. The obscuration level and its impact on hardware and device integration are computed using knowledge learning. This is monotonous for different service intervals requiring different device integrations. The proposed model is validated using hardware, software, interface, and communication and bug fixes testing that demands irregular device and interface integrations. Based on the knowledge of the previous integration intervals, the obscuration level is identified for suitable testing and progression. This identification uses maximum integration semantics for different devices and service intervals. The complexity is reduced by recommending device replacements, bug fixes, and capacity changes for real-time scenarios. The proposed model reduced the interfacing complexity by 8.94% and improved the integration progression by 15.04% for the maximum device count.

Further, SIOM can be developed in its ability to adapt and become intelligent for problem-solving issues that may arise. The study can include enabling dynamic device reconfiguration according to the dynamic change in situations, supporting heterogeneous networks through cross-protocol interoperability, and adapting edge computing using lightweight knowledge-based learning optimization for real-time decisions over resource-constrained edges. Integrating AI-based techniques, such as reinforcement learning, could predict obscurations and automatically make the necessary changes in the system. Further, embedding secure interfacing protocols and exploring federated learning would address cybersecurity and privacy concerns, thus ensuring collaborative knowledge updates while safeguarding sensitive data in increasingly complex IoT environments.

Acknowledgement: This work was supported by University of Al Maarif.

Funding Statement: The authors received no specific funding for this study.

Author Contributions: The authors confirm their contributions to the paper: study conception and design: Mohammed E. Seno; data collection: Mahmood Alsaadi; analysis and interpretation of results: Mohammed I. Khalaf; draft manuscript preparation: Mohammed E. Seno. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: The data that support the findings of this study are available from the corresponding author, Mohammed E. Seno, upon reasonable request.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest to report regarding the present study.

References

[1] W. Rafique, A. S. Hafid, and S. Cherkaoui, "Complementing IoT services using software-defined information centric networks: A comprehensive survey," *IEEE Internet Things J.*, vol. 9, no. 23, pp. 23545–23569, Dec. 2022. doi: 10.1109/JIOT.2022.3206146.

[2] A. Tibazarwa, "Strategic integration for hardware and software convergence complexity," *IEEE Eng. Manag. Rev.*, vol. 49, no. 3, pp. 92–102, Sep. 2021. doi: 10.1109/EMR.2021.3089475.

[3] R. C. Motta, K. M. de Oliveira, and G. H. Travassos, "An evidence-based roadmap for IoT software systems engineering," *J. Syst. Softw.*, vol. 201, no. 8, 2023, Art. no. 111680. doi: 10.1016/j.jss.2023.111680.

[4] N. Gavrilović and A. Mishra, "Software architecture of the internet of things (IoT) for smart city, healthcare and agriculture: Analysis and improvement directions," *J. Ambient Intell. Humaniz. Comput.*, vol. 12, no. 1, pp. 1315–1336, Jun. 2020. doi: 10.1007/s12652-020-02197-3.

[5] B. M. M. El-Basioni and S. M. Abd El-Kader, "Designing and modeling an IoT-based software system for land suitability assessment use case," *Environ. Monit. Assess.*, vol. 196, no. 4, Mar. 2024, Art. no. 380. doi: 10.1007/s10661-024-12483-8.

[6] I. Ungurean and N. C. Gaitan, "A software architecture for the industrial internet of things—A conceptual model," *Sensors*, vol. 20, no. 19, Sep. 2020, Art. no. 5603. doi: 10.3390/s20195603.

[7] Z. G. Imran, A. Alshahrani, M. Fayaz, A. M. Alghamdi, and J. Gwak, "A topical review on machine learning, software defined networking, internet of things applications: Research limitations and challenges," *Electronics*, vol. 10, no. 8, Apr. 2021, Art. no. 880. doi: 10.3390/electronics10080880.

[8] M. Qasaimeh, R. S. Al-Qassas, and M. Ababneh, "Software design and experimental evaluation of a reduced AES for IoT applications," *Future Internet*, vol. 13, no. 11, Oct. 2021, Art. no. 273. doi: 10.3390/fi13110273.

[9] E. Ahmad, "Model-based system engineering of the internet of things: A bibliometric literature analysis," *IEEE Access*, vol. 11, pp. 50642–50658, 2023. doi: 10.1109/ACCESS.2023.3277429.

[10] S. Zhu, S. Yang, X. Gou, Y. Xu, T. Zhang and Y. Wan, "Survey of testing methods and testbed development concerning internet of things," *Wirel. Pers. Commun.*, vol. 123, no. 1, pp. 165–194, Sep. 2021. doi: 10.1007/s11277-021-09124-5.

[11] S. Shadroo and A. M. Rahmani, "Systematic survey of big data and data mining in internet of things," *Comput. Netw.*, vol. 139, no. 15, pp. 19–47, Jul. 2018. doi: 10.1016/j.comnet.2018.04.001.

[12] A. A. Helal, R. S. Villaça, C. A. S. Santos, and R. Colistete, "An integrated solution of software and hardware for environmental monitoring," *Internet Things*, vol. 19, no. 2, Aug. 2022, Art. no. 100518. doi: 10.1016/j.iot.2022.100518.

[13] H. Poveda, K. Navarro, F. Merchan, E. Ramos, and D. G. González, "A software defined radio-based prototype for wireless metrics studies in IoT applications," *Wirel. Pers. Commun.*, vol. 120, no. 3, pp. 2291–2306, Feb. 2021. doi: 10.1007/s11277-021-08281-x.

[14] A. Amjad, F. Azam, M. W. Anwar, and W. H. Butt, "A systematic review on the data interoperability of application layer protocols in industrial IoT," *IEEE Access*, vol. 9, pp. 96528–96545, 2021. doi: 10.1109/ACCESS.2021.3094763.

[15] A. Brunete, E. Gambao, M. Hernando, and R. Cedazo, "Smart assistive architecture for the integration of IoT devices, robotic systems, and multimodal interfaces in healthcare environments," *Sensors*, vol. 21, no. 6, Mar. 2021, Art. no. 2212. doi: 10.3390/s21062212.

[16] A. Sevin and A. A. O. Mohammed, "A survey on software implementation of lightweight block ciphers for IoT devices," *J. Ambient Intell. Humaniz. Comput.*, vol. 14, no. 3, pp. 1801–1815, Jul. 2021. doi: 10.1007/s12652-021-03395-3.

[17] R. Seiger, R. Kühn, M. Korzetz, and U. Aßmann, "HoloFlows: Modelling of processes for the internet of things in mixed reality," *Softw. Syst. Model.*, vol. 20, no. 5, pp. 1465–1489, Jan. 2021. doi: 10.1007/s10270-020-00859-6.

[18] J. Cao, T. Zhu, R. Ma, Z. Guo, Y. Zhang and H. Li, "A software-based remote attestation scheme for internet of things devices," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 2, pp. 1422–1434, Mar. 2023. doi: 10.1109/TDSC.2022.3154887.

[19] J. L. Herrera, J. Galán-Jiménez, J. Berrocal, and J. M. Murillo, "Optimizing the response time in SDN-fog environments for time-strict IoT applications," *IEEE Internet Things J.*, vol. 8, no. 23, pp. 17172–17185, Dec. 2021. doi: 10.1109/JIOT.2021.3077992.

[20] X. Zhu, T. Zhang, J. Zhang, B. Zhao, S. Zhang and C. Wu, "Deep reinforcement learning-based edge computing offloading algorithm for software-defined IoT," *Comput. Netw.*, vol. 235, no. 2, Nov. 2023, Art. no. 110006. doi: 10.1016/j.comnet.2023.110006.

[21] D. Alulema, J. Criado, L. Iribarne, A. J. Fernández-García, and R. Ayala, "SI4IoT: A methodology based on models and services for the integration of IoT systems," *Future Gener. Comput. Syst.*, vol. 143, no. 1, pp. 132–151, Jun. 2023. doi: 10.1016/j.future.2023.01.023.

[22] Q. Zhou, "Smart library architecture based on internet of things (IoT) and software defined networking (SDN)," *Heliyon*, vol. 10, no. 3, Feb. 2024, Art. no. e25375. doi: 10.1016/j.heliyon.2024.e25375.

[23] S. Zafar, B. Zafar, X. Hu, N. H. Zaydi, M. Ibrar and A. Erbad, "PBCLR: Prediction-based control-plane load reduction in a software-defined IoT network," *Internet Things*, vol. 24, no. 11, Dec. 2023, Art. no. 100934. doi: 10.1016/j.iot.2023.100934.

[24] A. Moin, M. Challenger, A. Badii, and S. Günnemann, "A model-driven approach to machine learning and software modeling for the IoT," *Softw. Syst. Model.*, vol. 21, no. 3, pp. 987–1014, Jan. 2022. doi: 10.1007/s10270-021-00967-x.

[25] G. B. Ghantous and A. Q. Gill, "Evaluating the DevOps reference architecture for multi-cloud IoT-applications," *SN Comput. Sci.*, vol. 2, no. 2, Mar. 2021, Art. no. 123. doi: 10.1007/s42979-021-00519-6.

[26] J. -C. Chen, C. -C. Chen, C. -H. Shen, and H. -W. Chen, "User integration in two IoT sustainable services by evaluation grid method," *IEEE Internet Things J.*, vol. 9, no. 3, pp. 2242–2252, Feb. 2022. doi: 10.1109/JIOT.2021.3091688.

[27] M. Gaglianese, S. Forti, F. Paganelli, and A. Brogi, "Assessing and enhancing a cloud-IoT monitoring service over federated testbeds," *SSRN Electron. J.*, 2022. doi: 10.2139/ssrn.4241469.

[28] J. Ali and B. Roh, "An effective approach for controller placement in software-defined internet-of-things (SD-IoT)," *Sensors*, vol. 22, no. 8, Apr. 2022, Art. no. 2992. doi: 10.3390/s22082992.

[29] Q. Zhang *et al.*, "Human body IoT systems based on the triboelectrification effect: Energy harvesting, sensing, interfacing and communication," *Energy Environ. Sci.*, vol. 15, no. 9, pp. 3688–3721, 2022. doi: 10.1039/D2EE01590K.

[30] Ü. Işikdağ, "An IoT architecture for facilitating integration of geoinformation," *Int. J. Eng. Geosci.*, vol. 5, no. 1, pp. 15–25, Feb. 2020. doi: 10.26833/ijeg.587023.

[31] M. Mouine and M. A. Saied, "Event-driven approach for monitoring and orchestration of cloud and edge-enabled IoT systems," in *2022 IEEE 15th Int. Conf. Cloud Comput. (CLOUD)*, Jul. 2022, pp. 273–282. doi: 10.1109/cloud55607.2022.00049.

[32] W. Villegas-Ch, X. Palacios-Pacheco, and M. Román-Cañizares, "Integration of IoT and blockchain to in the processes of a university campus," *Sustainability*, vol. 12, no. 12, Jun. 2020, Art. no. 4970. doi: 10.3390/su12124970.

[33] J. Lee, K. Gilani, N. Khatoon, S. Jeong, and J. Song, "SEIF: A semantic-enabled IoT service framework for realizing interoperable data and knowledge retrieval," *IEIE Trans. Smart Process. Comput.*, vol. 12, no. 1, pp. 9–22, Feb. 2023. doi: 10.5573/IEIESPC.2023.12.1.9.

[34] C. Su, Y. Han, X. Tang, Q. Jiang, T. Wang and Q. He, "Knowledge-based digital twin system: Using a knowlege-driven approach for manufacturing process modeling," *Comput. Ind.*, vol. 159–160, no. 6, Aug. 2024, Art. no. 104101. doi: 10.1016/j.compind.2024.104101.

[35] M. Pandit *et al.*, "Towards design and feasibility analysis of DePaaS: AI based global unified software defect prediction framework," *Appl. Sci.*, vol. 12, no. 1, Jan. 2022, Art. no. 493. doi: 10.3390/app12010493.

[36] A. Kukkar *et al.*, "ProRE: An ACO-based programmer recommendation model to precisely manage software bugs," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 35, no. 1, pp. 483–498, Jan. 2023. doi: 10.1016/j.jksuci.2022.12.017.

[37] O. S. Contiki, "Cooja simulator IoT simulation," Network Simulation Tools, Jan. 14, 2023. Accessed: Dec. 13, 2024. [Online]. Available: https://networksimulationtools.com/contiki-os-cooja-simulator-iot-simulation.