Enhancing End-to-End Communication Security in IoT Devices Using Model Driven Engineering

BY

Rimsha Zahid

(Registration No: MS-SE-20-330694)

Supervisor

Dr. Farooque Azam

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING,

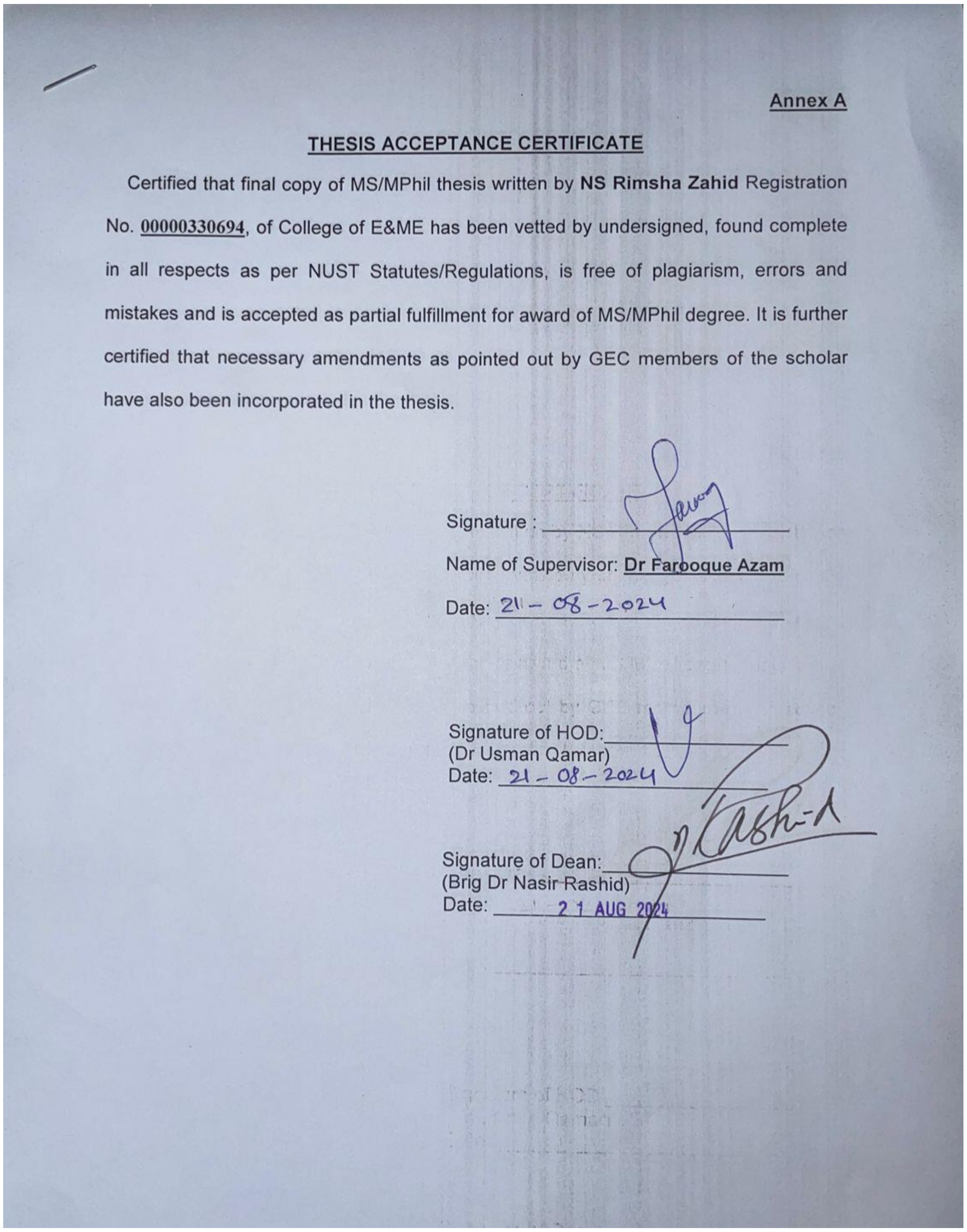
COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING,

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY,

ISLAMABAD

August 21, 2024

Thesis Acceptance Certificate



Annex A

THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS/MPhil thesis written by **NS Rimsha Zahid** Registration No. **00000330694**, of College of E&ME has been vetted by undersigned, found complete in all respects as per NUST Statutes/Regulations, is free of plagiarism, errors and mistakes and is accepted as partial fulfillment for award of MS/MPhil degree. It is further certified that necessary amendments as pointed out by GEC members of the scholar have also been incorporated in the thesis.

Signature : [illegible]

Name of Supervisor: **Dr Farooque Azam**

Date: 21-08-2024

Signature of HOD: [illegible]

(Dr Usman Qamar)

Date: 21-08-2024

Signature of Dean: [illegible]

(Brig Dr Nasir Rashid)

Date: 21 AUG 2024

Enhancing End-to-End Communication Security in IoT Devices Using Model Driven Engineering

BY

Rimsha Zahid

(Registration No:0000330694)

A thesis submitted to National University of Science and Technology Islamabad in partial fulfillment of the requirements for the degree of Master of Sciences in Software Engineering

<u>Supervisor:</u>

Dr. Farooque Azam

<u>Co Supervisor:</u>

Dr Wasi Haider Butt

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING,

COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING,

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY,

ISLAMABAD

August 21, 2024

Dedicated to my beloved Parents for their endless support and prayers, and to my Husband and Daughter who always motivated and inspired me with unwavering support.

ACKNOWLEDGEMENTS

First, I would like to praise and honor the Almighty Allah, the most beneficent and the most merciful, for granting me the ability, courage, knowledge, and skills required to undertake and accomplish this task. No doubt, He has facilitated my journey, and I am unable to achieve anything without His blessings.

I would also like to appreciate the sincere efforts of my supervisor **Dr. Farooque Azam** for guiding me throughout the journey of my MS thesis. Furthermore, I am also grateful to him for teaching us the courses of Model Driven Software Engineering (MDSE) and Software Development and Architecture (SDA). He professionally taught us both subjects in depth and from that; I developed my interest in continuing my research in the field of model-driven software engineering.

I would also like to express my gratitude to my Guidance Committee Members **Dr. Wasi Haider Butt, Dr. Arsalan Shaukat,** and **Dr. Mehwish Naseer** for providing guidance and assistance to further improve my work with their valuable recommendations. I would like to express my sincere gratitude to **Dr. Muhammad Waseem Anwar** for his assistance and cooperation throughout the journey of my thesis in achieving my research objectives. Without his unwavering support, the completion of this dissertation would not have been possible. I appreciate his patience and support throughout the whole thesis.

I am deeply grateful to my beloved parents for raising me and being available whenever I needed them and for their unwavering support throughout every aspect of my life.

Finally, I would also like to extend my heartfelt gratitude to my husband and my family for their steadfast support and cooperation through the research journey.

ABSTRACT

The Internet of Things (IoT) has transformed the way software and hardware interact, enabling users to control devices remotely. In environments like smart homes, offices, and industrial setups, diverse sensors generate large amounts of data, raising significant security concerns during transmission. Cloud-based systems enhance efficiency and cost-effectiveness but also introduce authentication and authorization vulnerabilities. While past studies addressed specific system issues, this research proposes a model-driven framework to tackle encryption challenges at the application layer, ensuring data integrity during device communication. The absence of a generic security-related solution that could be tailored to the specific needs of any IOT network configuration increased the overall effort of making the system secure since every time you need to devise a solution that solves the security concerns. Therefore, this research advocates for an End-to-End encryption metamodel for message communication through the application layer, safeguarding messages from unauthorized alterations by third parties. End-to-end encryption standards ensure that the data as well as the processed information travelling between different components of the network is not visible to any third party. With the introduction of metamodel for security and by creating profiles, we extend a generic model by adding domain-specific properties and constraints to tailor it for specialized security-related concerns. The proposed framework IoT developers to input data details and model the encryption-decryption process, ensuring data integrity during communication. This model can be used by the developers working with IoTs to simply put the details of the data and they can model an encrypted IOT network by ensuring data integrity while it is being communicated. By providing an atomic view of IoT network configuration, the proposed framework incorporates the robust security mechanism so that any network can be verified mapping with respect to the meta-model and thus executed in a live environment ensuring the data sensing, transmitting, and then transferring into pieces of information is safe throughout the way. Using Node-Red as a proof of concept, a case study was implemented to validate the efficacy of encoding and decoding processes.

Keywords: MDSE (Model Driven Software Engineering), IOT (Internet of Things), End-to-End Encryption, Communication, Security

Table of Contents

List of Figures

List of Tables

CHAPTER 1

INTRODUCTION

This chapter of the research is an explanation of the background of the topic, explaining the important terms in the title along with their importance. The chapter also encompasses the objectives, research problem, proposed methodology, and the structure of the thesis.

1.1. Background Study:

1.1.1. Internet of Things (IoT)

The Internet of Things (IoT) is a collective term to describe everyday objects connected to computing devices [1]. This entails connecting numerous physical yet smart objects through the internet so that they can work well in many areas of our lives for example smart transportation systems, building and home automation, wearable healthcare technologies, industrial process control, and infrastructure monitoring. When traditional devices are brought online with the help of IoT, they are first transformed into smart devices. There are many benefits associated with the integration of smart devices into our daily lives. They range from making things simple for humans, enabling them to make informed decisions, and sharing information among other benefits. Achieving the full potential of the Internet of Things (IoT) develops a pathway to gainful venture making sure responses are quick and reliable, economic use of available infrastructure as well as effortlessly combining different devices and technologies that work for it, scaling adequately to support growth in network breadth; also having strong systems in place that allow it to collect, exchange and process data more efficiently [2]. According to a study carried out in 2020, over 749 billion dollars was spent on these devices. Projections showed growth to 1.1 trillion in 2023 for yearly spending on the same things for all nations. In 2018, there were 22 billion connected IoT devices and twice that number will be surpassed, reaching 60 billion in 2030. [1] A milestone in technological advancement is marked by the coming of 5G which sees an increase in computing devices integrated into different daily life scenarios connected through the internet while a large amount of data and processed information travels around the globe.

IoT is considered one the most important inventions as it has reduced lot of human manual work while automating daily tasks and reduced labor cost. It is a network of devices over the web in which embedded systems are used to collect, send, and transform the data. They mostly include sensors, digital machines, and consumer objects along with mechanical. With the increase in ease and satisfaction, a lot of businesses have been using the IoT to operate efficiently and effectively.

When a network of interrelated things connects and exchanges data with other devices in the group or over a network and interacts with the cloud, the IoT network becomes vulnerable to third-party access.

1.1.2. End-to-End Encryption

It is a form of secure communication that hides messages from non-potential users and safely and securely sends the information to the one who needs it, whether it's another system or merely a device. It is a technique of securing useful data into unreadable format and then transporting it for communication purpose. The transformed data format is usually known as cipher text while turning it back to its original yet readable form requires a key that decrypts this cipher text into human-readable form. This data security process has been widely used to ensure the confidentiality of useful information while keeping its integrity intact.

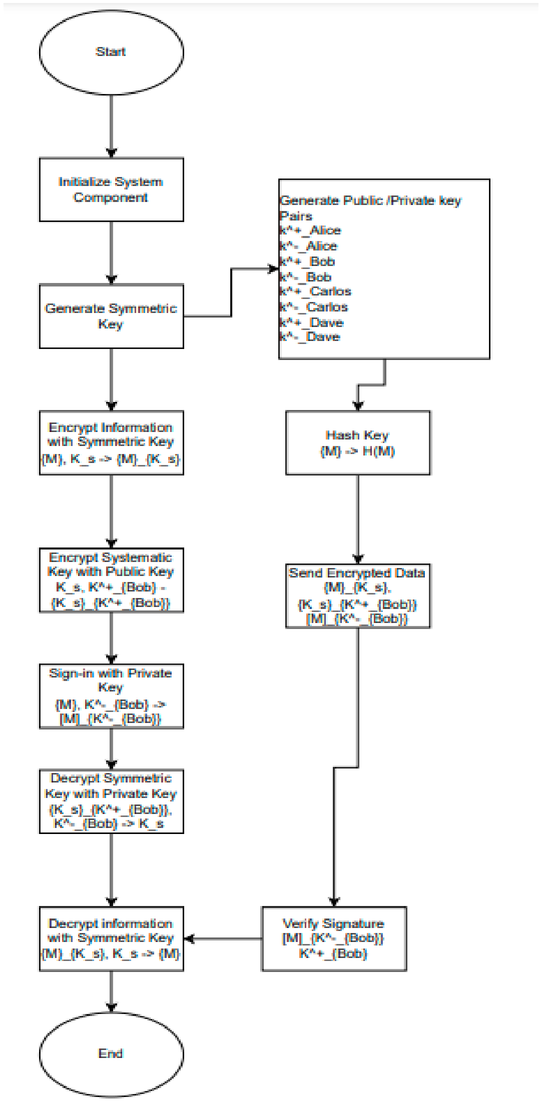

Start
Initialize System Component
Generate Symmetric Key
Generate Public /Private key Pairs
k^+_Alice
k^-_Alice
k^+_Bob
k^-_Bob
k^+_Carlos
k^-_Carlos
k^+_Dave
k^-_Dave
Encrypt Information with Symmetric Key {M}, K_s -> {M}_{K_s}
Hash Key {M} -> H(M)
Encrypt Systematic Key with Public Key K_s, K^+_{Bob} - {K_s}_{K^+_{Bob}}
Send Encrypted Data {M}_{K_s}, {K_s}_{K^+_{Bob}} [M]_{K^-_{Bob}}
Sign-in with Private Key {M}, K^-_{Bob} -> [M]_{K^-_{Bob}}
Decrypt Symmetric Key with Private Key {K_s}_{K^+_{Bob}}, K^-_{Bob} -> K_s
Decrypt information with Symmetric Key {M}_{K_s}, K_s -> {M}
Verify Signature [M]_{K^-_{Bob}} K^+_{Bob}
End

Figure 1: Data Encryption Mechanism

This technique of hiding the information while its way of transformation protects sensitive information whether it is stored on any wearable smart gadget, smartphone, or computer, just to name a few. Encryption guarantees data confidentiality in all types of communication whether it is being sent via email, message, shared over a call, or sent in a file. Finance takes great help from the ciphers as it gives a seamless yet reliable online banking experience to their customers whereas their credentials, transactions, or any other payment details are being protected from any kind of breach. Also, it ensures that the updates they are getting are sources from a legitimate source. In a nutshell, encryption is a fundamental part of the digital global world. It serves as a backbone for security where threats are avoided and gives users a reliable environment to operate within. [3] [4]

Data encryption and generation of public/private keys are done by clients. Different data models are allowed to be deployed to show that in a remote database, their instances can be serialized to enable encryption. Different label features are introduced that allow data scope in different groups. It is done to enhance trust and provide a trust system that authenticates the identity [5]. For example, see

Figure , First A sends a trust invitation to B. Then they establish trust and verify with the help of the other system which is known as a third-party mechanism. Then Alice generates a server request to identify the identity of the author.

In end-to-end encryption, the sender only knows what he has sent and only the receiver receives exactly what the sender has sent and no one can decrypt the message in the middle. It connects millions of devices and protects from vulnerabilities like private information that may be exposed when all IoT devices are connected. To ensure users' privacy and safety, it is important to enable secure endpoints in IoT. [6]. So, to achieve guaranteed security without compromising its other non-functional requirements such as performance, and scalability, this area is crucial to be researched. The main problem that is faced these days is that the primitive security in computations-constrained devices is costly for usage. [7] This protection of data through encryption is further divided into two types:

1. **Symmetric Encryption:** This method makes use of the same key for encrypting the data as well as decrypting it. Being the fast and efficient way of data security, this type of encryption is usually preferred to protect large amounts. Data Encryption Standard (DES) and Advanced Encryption Standard (AES) are commonly used symmetric encryption algorithms.
2. **Asymmetric Encryption:** Commonly referred to as the public-key encryption method, this type of encryption has a public key and private key which are used for encryption and decryption respectively. This combination of keys makes it a reliable way of protecting the data from unauthorized access. Elliptic Curve Cryptography (ECC) and Rivest-Shamir-Adleman (RSA) are examples of asymmetric encryption algorithms.

End-to-end Encryption is a special case of data encryption where the encoding and decoding of data are done at the sender and receiver's end respectively – no intermediate parties are involved in the encryption process whether it is any access granting body or service provider. End-to-end encryption works in a manner where a message sent by the user is transformed into cipher text with a key and this unreadable combination of alphanumeric characters is routed over the network to the receiver. Afterwards, the receiving node decrypts this data into a readable form using their private key ensuring even if someone gets access to the cipher text on its way to the receiver, there is no chance that they would be able to read it without the key.

This method of encryption protects from man-in-the-middle attacks, safeguards data against eavesdropping, and ensures data integrity as it is not being modified by anyone when it is in transit. A high level of privacy of data is maintained since communicating users are the only ones having access to the comprehendible form of the data. End-to-end encryption is the strongest form of encryption which is being used for communication widely in the global world where any type of data is being exchanged between different parties.

1.1.3. Security and its Significance in IOT

Security is defined as a collection of different measures to safeguard critical assets from being stolen or taken hold of for illegal purposes. It involves safeguarding any physical object or information that is stored digitally – the latter is common these days as we have entered into a world of digital assets. Protection of these assets from any kind of harm from unauthorized persons is the core of security. Unlike physical security measures which involve taking protective measures like surveillance systems, control locks, and guards, network security involves the protection of information in terms of its availability, integrity, and confidentially when it is being transmitted in shared networks. Data security lies under this category where encryption techniques, firewalls, and other algorithms are being employed to safeguard data from intrusions.

Confidentiality, Integrity, and Availability (CIA) is the core of security where confidentiality is the limitation on data accessibility to authorized uses, integrity is the completeness and maintenance of data and availability is to make sure that information remains accessible to the users when needed. When discussing IoT, this CIA matrix is critical because enormous data are being generated and transmitted from one node to another. For example, when tiny sensors in cities, industries, or home smart meters that send data must be periodically connected to communicate with a large number of groups, safekeeping this critical data is a resourceful task – failure of which might lead to compromising the reliability of the data [8]. A Robust technique is needed to ensure this data is protected such as the use of a secure communication path, implementing limitations on the access of data, and implementing robust algorithms.

With the progression of remote servers over the Internet such as the cloud, and the potential risk they pose - for example, limited resources and uncertainty in connectivity and a very large number of potential users – a combination of all these factors makes it difficult for group communication. To solve all the potential issues of data protection and to implement intermittent, low bandwidth, intermediate edge servers with low range connections in the context of the base station – there is a need to introduce an effective measure for meditating communication. [9]

IoT uses a mechanism that is widespread in indirect communication primitives, where IoT trusted servers are installed close to IoT nodes. During all this time the risk still stands that arises from untrusted servers as the network is being connected to multiple unknown, small yet smart devices to make their operations smooth and worthy for us.

1.1.4. Model Driven Software Engineering (MDSE)

MDSE helps in increasing the quality of software products, enhancing productivity, reducing the cost and time of software development, and making it more maintainable [10]. During the development process subset of software engineering, model-driven software is treated as first-class entities. Models in MDSE serve a specific purpose such as code generation, model integration manipulation, and construction. Model is used as a development methodology, and it is only specified for creating a model that is domain-specific and defines the specific problems of those conceptual models [9].

Many real-time applications are developed using components and this is crucial for developing many applications where timing is crucial. A metamodel is a model which is developed using models. Thus, the analysis, construction, frame development, and some theories are defined already. In General, Round-trip engineering is used as a concept for software design that is the best approach for model-driven software engineering [11] [12].

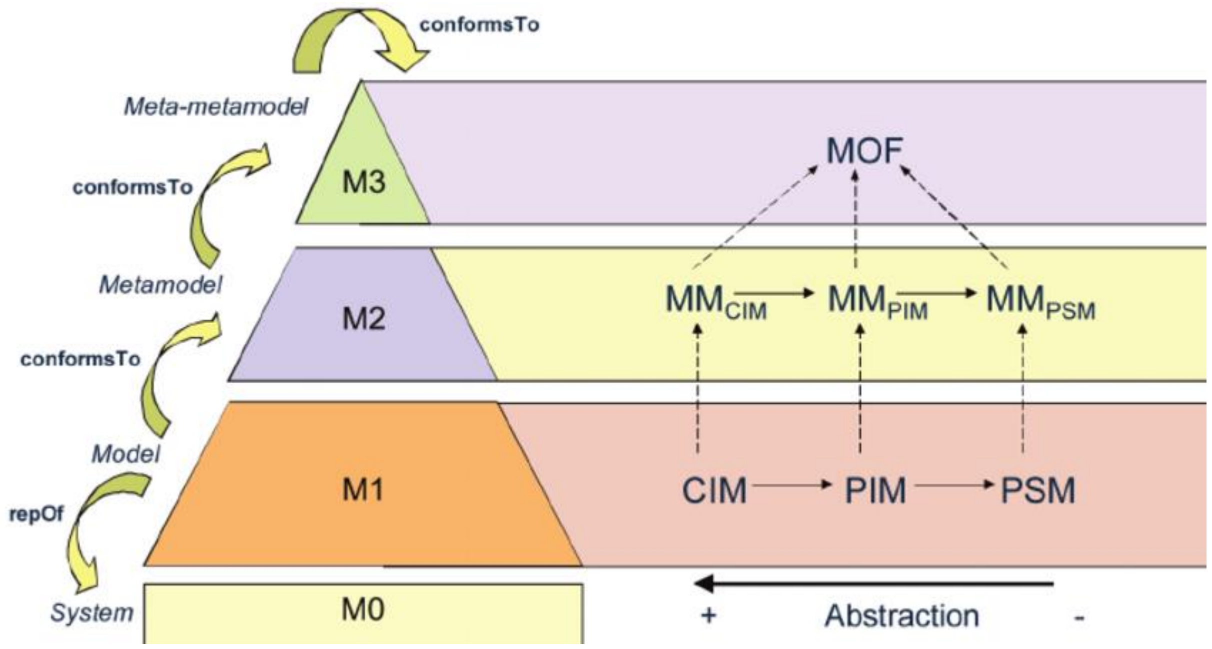


Figure 2: Overview of Model-Driven Development

Metamodel is a model that is the representation of the actual model. It looks exactly like a mathematical algorithm where you give the input, and it generates the output. Some of the abstract syntax is defined in a metamodel and they are specified for creating rules in them. [13]. Below are the fundamentals of meta-modeling:

- **Abstraction Level:** In this level, specific models are used that is used to create a schema for other models
- **Standardization:** It is about how models are created and how they are used for standardization to ensure consistency and interoperability
- **Hierarchical Structure:** For this, four layers are present in a hierarchy with M3 meta-model layers and M0 an instance-level

The main aim of model-driven techniques is to increase productivity as it also helps simplify the process of design. It promotes a culture of interaction among software development teams. Also, working with modeling approaches increases the compatibility between two systems as it helps to create the proper structure for the development of software products and results in the production of documentation, source code tests, and more generated algorithms for domain models [6].

Model engineering is a method that is effective for the implementation of a software system that is developed with the help of designers, developers, and managers that are specified for the development domain specific for model-driven engineering [11]. It is also used to automate the tasks that are specified for the analysis such as generation, analysis, and simulations [11].

Models are emphasized to use in the specific stage of development such as design, implementation, coding, testing, or automation. They are used to automate, reducing the effort of coding and helping in generation. It is used to separate one model from another model, and it is a widely used concept that allows stakeholders to focus on a specific problem [11].

1.2. Research Contribution

This research addresses a critical gap in IoT security by leveraging Model-Driven Engineering (MDE) to enhance end-to-end communication security. The growing complexity and ubiquity of IoT devices make traditional security approaches increasingly inadequate. MDE provides a structured methodology to model IoT systems at a high level of abstraction, enabling precise specification of security requirements and behaviours. By systematically transforming these models into secure and executable configurations, we ensure that IoT devices and their communications adhere to stringent security protocols. This approach addresses the inconsistencies and vulnerabilities inherent in manual security implementations, offering a robust and scalable solution to IoT security challenges.

One of the significant contributions of this research is the demonstration of how MDE can effectively bridge the gap between high-level security requirements and low-level implementation details. Traditional approaches often struggle to keep up with the dynamic and heterogeneous nature of IoT ecosystems, where devices vary widely in capability and function. By employing MDE, we can create adaptable models that capture the security needs of diverse IoT devices and their interactions. These models facilitate the automatic generation and validation of security policies, ensuring that all devices within an IoT network comply with the established security framework. This not only enhances the overall security posture of IoT systems but also simplifies the process of maintaining and updating security measures as the network evolves.

1.3. Thesis Organization

In this chapter, we have given a comprehensive analysis of the research by defining some important terms as well as the objectives the study would have had as well as the problem statement that led to this research. The literature review highlights significant studies carried out on IOT, data transmission and security in IoT, challenges like real-time processing, vulnerability to attacks, and finally, we figure out the gaps/problem statement in current security mechanisms.

In Chapter 4, we look at how Model Driven Development (MDD) can simplify the protection of data in IoT and introduce a generic model to improve security, and integrity in intricate IoT systems. The next chapter presents the implementation of our model-driven methodology with a focus on communication security for improving real-time data collection and transmission encryption needed to make IoT configurations secure after which this will contribute to increased system reliability and cybersecurity. In the validation part, we have discussed how our model integrates advanced security techniques into IoT systems and shows how it can secure and manage complex infrastructures with a case study, emphasizing its practical utility and adaptability.

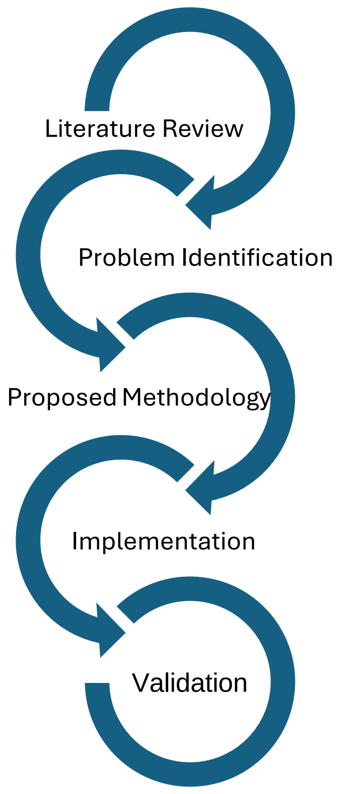


Figure 3: Flow of Research

CHAPTER 2

LITERATURE REVIEW

There is extensive work that is done in the field of IOT, specifically on its security, communication, and data. In this chapter, we extensively discussed the previous research studies carried out in this domain – majorly in terms of MDSE where solutions are simplified. We thoroughly explore the current techniques and methodologies and conclude this chapter with the identification of the gap leading to the motivation of carrying out this research i.e., in the form of the problem statement.

2.1. Security Challenges of IoT

To maintain hustle-free communication among devices, a dynamic and comprehensive system is required and providing fast data processing [14]. To achieve this, a system that is recognized as an Information Centric-Networking System (ICNS) is implemented in [15]. The method that is used is known as the naming method where a unique name is assigned to each node. The unique naming system is known as NOID. Only in-network communication is provided by them. A trust system is implemented with optimization algorithms. It is based on cluster head section but ultimately poses more threats [14]. It contains honest nodes that give positive answers to real nodes and negative ones to malicious code. It lacks the effectiveness of the model and is not able to detect the specific attacks of IoT networks thus making questionable performance in other networks. [14] They have limited or no information and capacities to handle the sophisticated exploits these days and zero attack vulnerability

Computational capabilities of IoT devices are limited and they are inherently vulnerable, and make the system difficult when implementing robust techniques for security. [16]The problems that are raised due to this are the weak authentication and prevalence of these vulnerabilities. The IoT devices that are susceptible to hijacking and denial of service (DoS) attacks [17]. For the protection of credentials of IoT Devices, a Mirai botnet was created which resulted in exploration with the help of Dos attack [18]. Cloud-based devices and network channels offer an environment that is quite scalable and provides the methods for securing IoT devices for the development of computational tasks that are quite intensive. They use the SGX methods that provide hardware-security-based technologies that run the enclaves of the creation of sensitive data. It provides the type of information that is secured from unauthorized access, even for the type of operating system that has a complex built [16].

2.1.1. Cyber Security and Communication

With the progression of the world, the IoT is introduced in farming as well to improve operations [11]. But with the rapid increase in IoT, they pose a lot of serious risks. They are not considered because they yield a very high cost. Potential risks result in perilous and unproductive farming [19]. An entire field of a crop can be destroyed from exploits and their threats are more highlighted that farmlands can be flooded through them, and pesticides sprayed over farms with the help of drones. The attacks are coordinated largely, the term that is used to refer to them is known as Cyber Agroterrorism [20] and as a result, the economy is destroyed. The other attack that is worth mentioning is "The Night Dragon" where the information large petrochemical product was stolen [21]. Another famous attack was at a German Steel Mill where the office network and the production system of the plan were accessed by attackers using Spear Phishing [22]. The shift in the usage of digital products has been widely seen and is difficult to catch the man in the middle [20].

For over two and more decades the constant persistent in the world of the internet is Distributed DoS (DDoS) attack. It causes issues like making it inaccessible to legitimate users and flooding the servers with huge traffic thus disrupting network services [19]. These types of factors play a role in the exploitation of the shared access of data to users and making it difficult for the understanding the difference and making it difficult to recognize who are actual users and who is forging it [23]. The major problems that arise in cyber security are to distinguish between the legitimate users and the attackers [24].In traditional DOS mitigations, famous methods are used such as reducing the traffic rate, IP whitelisting/whitelisting, and intrusion detection systems. They are defined as

- **Rate Limiting**: I6t reduces the amount of traffic to be reduced only a specific number of users can access it at a time and the chances of attacks reduce. Also, it limits the actual user to access the information and perform other functionalities it provides.
- **IP Blacklisting**: Some IP addresses are classified as a malicious IP address, while some people use lethal methods such as Botnet that changes the IP address of the device for access [24]. Authors in [25] have addressed the basic data security concerns and have listed down authenticity properties in Table I. The interest level of each requirement is proportional to the percentage of academic discussion carried on that specific requirement compared to the total number of studies available in that category.

ID	Security Requirement	Relative Interest	% within category
01	Multi-factor Authentication	Medium	9%
02	Transitive Authentication	Low	1%
03	Mutual Authentication	Medium	17%
04	Privacy-preserving Authentication	Medium	8%
05	Non-Repudiation	Low	4%
06	Attestation	Low	5%

Table 1. List of Security concerns discussed by academia in the field of IoT

2.1.2. Detection Techniques

Authors in [26] have used Deep Learning models that Enable Threat Intelligence (DTLI) to allow cloud edge networks to detect anomalies. Two main components are involved which help to identify threats. Deep Pattern Extractor (DPE) and Threat Type Identification (TTI). DPE uses generative deep-learning modules that are responsible for the automatic extraction of patterns [21]. They are used to capture the hidden patterns and transform the network traffic area that is not properly defined into meaningful and comprehensible representations with the class labels. They allow us to understand the behavior better [1]. SICON (Scalability, Control, and Isolation on Next-Generation Networks), poses the new techniques that help in addressing the key challenges that are faced these days. [27] The ISDs in them are used to organize the multiple autonomous systems and they have their routing pane. This system helps in the separation of the path and clears confusion and misconfiguration. They have their key which follows Public Key Infrastructure, that authenticates messages and prevents attacks. With this technology, they prevent DDoS attacks and by redesigning them they provide the proper structure and architecture that helps to achieve the aim of SICON. However, if used on a massive scale, the error might be reduced and the risk and potential failures it poses can be reduced [27].

Gated Recurrent Neural Network (GRNN) is used to identify the anomaly that the DPE model provides that makes the TIDD more powerful. It distinguishes the patterns that are to be identified as abnormal patterns with the help of the concurrent connections that are established and hidden multiple times across different steps [26]. With the combination of supervised and unsupervised behavior, it offers a heterogeneous network environment [26].Many Electric Circuit Units are equipped together in modern vehicles that allow communication among smart nodes connected in a network and a better user experience. The increase in electric units increases the chance of cyber-attacks and the susceptibility of modern vehicles [28]. Therefore, the feasibility of modern vehicles' automated systems emphasizes how security measures are necessary for the prevention of penetrations through loopholes [28]. Other vectors that play a role in the detection of the attack are the interfaces that provide the distance between the short-range and long-range devices [28].

Controller Area Networks (CAN) are more vulnerable to attacks since their design does not include these smart devices' specifications. The results show that CANS are more susceptible to DoS attacks and SQL injections as the system lacks authentication techniques [29]. This highlights that the CAN needs a security framework that protects IoT from attacks. Authors have also implied an Intrusion Detection System – it's a framework that uses two methods to Signature IDS network and an Anomaly-based IDS system. These detection techniques identify the common threats but if a new thread or an intrusion technique is introduced to the system, this gets a point of failure [29].

2.2. Security Enhancement Framework

Smart cars are one of the critical use cases of the IoT world as they are dependent on lots of sensors and actuators that are used to monitor vehicle functions. The optimal operation of the device is dependent on the collection of data including environmental factors, the performance of its engine, sensors, and the relevant actuators [17] The cooling system works by automated checking, if the system is too hot the system has actuators, so it adjusts the temperature accordingly. When all the small components are combined in the complex system, and they communicate with the help of the IoT that is when we get a fully functional smart system.

IoT requires real-time processing and transmission of data thus it a very difficult to control its latency to avoid human loss and financial loss. [12]The communication among devices in such systems is done with the help of the various mobiles, cellular data, satellites, and infrastructure that provide the framework for the running of the CAN. They allow communication among internal devices and external services ensuring their remote capabilities and providing real-time updates [21]. End-Devices Layer, Communication Layer, Services Layer, and Application Layer are the crucial layers in an IoT-enabled system.

IoT has become much more sophisticated that ensures to provide safe communication. The IoT Security Development Framework (ISDP) provides a sophisticated framework to ensure the safe transmission of data. ISDP supports the architecture of the system and mitigates malicious software [17]. From offering a top-notch authentication system to authorizing the users to use it the legitimate way, this framework keeps an audit of the usage profiles to maintain the integrity of the system.

2.2.1. MDD in IoT Security

The use of model-driven techniques for assisting developers in creating IoT applications through separation of concerns as well as abstraction was first discussed in 2015 for high-level application development for the Internet of Things [30]. The complexity of IoT app development was also discussed. This was about the lack of migration pathways, the absence of high-level abstractions, and life cycle phase challenges. The aim was to find a holistic solution whereby IoT applications could be broken down into distinct compulsions, and a model for application development could be created. This was made available through a methodology taking a paired set of languages employing conceptual modeling standards for specifying different development concerns; it also abstracts complexity due to size as well as variety." It also used the first time a model borrowing technique that combines task-mapping algorithms with code generators as well as linkers to make for more automatic programming [30].

The role of MDE in mission-critical IoT (MC-IoT) systems is significant. Heterogeneity management is a high-level abstraction provided by MDE, with separation of concerns for maintainability as well as automated adaptation to runtime adjustments, therefore reducing development effort through reusability. Together these advantages increase the reliability, safety, and security of MC-IoT systems. [31] The areas of security in terms of MDD are crucial and pose serious threats when IOT configurations are discussed - they must be analyzed and processed to fill the gap between the shortcomings to make the system more efficient user friendly and secure [19].

A research article introduced model-driven development (MDD) for a home security system, highlighting an organized method to create systems [32]. The design process is initiated by constructing high-level models depicted in Unified Modeling Language (UML) diagrams to capture vital entities like human detection modules or authentication processes. The sensor and doorbell signals interact with one another over Telegram along with a model for the access control mechanism based on how telegraphy expertise is utilized to control GPIO pins that lock or unlock the door from user commands [32]. The models are worked upon by iteration after which they are converted into executable Python code that enhances Telegram communication as well as GPIO management. In this Model Driven Development (MDD) approach there is an assurance that information obtained during the design phase will adequately inform further development. [32].

A model-driven approach was applied to improve Smart Street Lights [33]. Smart street lighting systems enable cars, bikes, and pedestrians to have their separate lighting zones through a system of smart lamp posts. MDE4IoT helped the developers to make models of behavioral capacities for software applications as well as hardware devices and their placements with the use of UML and ALF [33]. Several studies reference the use of these tools, quantifiably six have focused on textual languages implemented with the Xtext framework. For visual modeling languages with graphical syntax, two studies reported the use of Sirius, two studies used Eugenia, two studies employed GMF, and other studies noted the use of tools like Obeo Designer and MetaEdit+. [34]

Table 2. Distribution of papers by textual and graphical modeling language tools

	Textual Modeling Language Tools			**Graphical Modeling Language Tools**				
PID	**Framework**	**Xtext**	**Manticore**	**Sirius**	**Obeo Designer**	**GMF**	**MetaEdit+**	**Eugenia**
1	ML-QUADRATE (ML2)	Y	N	N	N	N	N	N
2	MONITOR-IOT	N	N	Y	N	N	N	N
3	SIMULATEIOT-FIWARE	N	N		N	N	N	Y
4	CYPRIOT-DSL	N	N	N	N	N	N	N
5	HEALMA	N	N	Y	N	N	N	N
6	CHESSIOT	N	N	N	N	N	N	N
7	SIMULATE iot	N	N	N	N	Y	N	Y
8	IoTSuite	Y	N	N	N	N	N	N
9	cypiot	Y	N	N	N	N	N	N
10	SiMoNa	N	N	N	N	N	Y	N
11	EL4IOT	Y	N	N	N	N	N	N
12	FRASAD	N	N	N	N	Y	N	N

Another article [35] explained how simulation could be achieved using model-driven technology. The focus was on the importance of models in the conception and regulation of simulation systems. Due to formal modeling languages, developers can detail system behavior, architecture, and communication. Apart from that, MDE contributes towards making simulation better by generating codes and enhancing interoperability thus guaranteeing replicability [35]. This method works by refining and transforming models, configuring physical components with deployment diagrams, and finally generating executable simulation models. As a whole, this method helps to make simulation software development easier and increases reliability in distributed systems [35].

2.2.2. Cloud Servers based Encryption:

As the client node is a resource-constrained device, and no matter the technology whether it is the web, mobile app, or computer Share lock works with the device regardless of whether the device is more powerful [36]. To manage clients, groups, and data through a public interface edge server supports the service layer and the database. The cloud server receives the data as the more trusted entity. The communication among groups takes place by untrusted edge servers. IoT, which serves another member of the group is a client from the perspective of group communication protocol. So, keeping given this analogy "servers" are the most trusted edge servers. As in the general structure, only one server is shared by multiple independent groups [37].

To rectify the problem where the third party can interact with the system and steal sensitive information about something very necessary to the party concerned, to rectify the problem a solution is required where third parties or some malicious users don't interact with the system and the information and data remains within the two parties. A model is needed that is present at the application layer and uses this layer to communicate with other devices we give input and it processes it [38].

The API abstracts the complexities of the communication messages and helps to focus on bigger things rather than small details. APIs provide the basic security methods that help to identify that there is no intruder They are interoperable means they can connect regardless of their device type, vendors, and manufacturers allowing a large number of devices to connect. We have used API endpoints that if the attacker even attempts to attack the APIs, the chances of penetration are lessened [39]:

- **Authentication:** Verifies the holder of the account of information is the same as the one accessing it.
- **Authorization:** Ensures the person's request to perform specific have permission to do that.
- **Encryption:** Converting message into some kind of Cipher text to ensure that no other person reads, accesses, or performs changes in it.

2.3. Research Gap

IoT security has progressed over the years – specifically with the introduction and implementation of Industry 4.0. However, gaps exist when we go through the previous research studies for securing IoT networks from threats in terms of MDD. The proceedings articles focused on specific security problems. Additionally, no suitable framework has been developed for comprehensive attack prevention. Therefore, there is a need for a complete methodology based on Model-Driven Architecture (MDA) to facilitate a driven Framework for the security of communicating devices and data in an IoT network. The proposed framework is intended to ensure the communication level security of an IoT-based network is simple, easy, and robust at the same time. This security framework is intended to devise a rigorous security mechanism in an IoT configuration and increase the reliability of the developers and ultimately users. The focus is to improve the integrity and efficacy of the sensor-actuator network by guaranteeing that the developed system designs satisfy the security requirements and will shield against vulnerabilities. The proposed framework efficiently handles the heterogeneous formats of data to be processed in real-time scenarios while protecting them from being preyed upon.

CHAPTER 3

PROPOSED METHODOLOGY

In the following chapter, we elaborate on the proposed technique for solving the complexities and security challenges that are common to communication integrity in IoT systems. Often conventional software development falls short of offering the requisite level of abstraction and visualization for efficient system management considering that systems are growingly complex and interconnected [40]. Model Driven Development (MDD) eliminates gaps between different levels for the developers to work with higher abstraction levels, making design, code, implementation, and testing simple [41].

This chapter explores different aspects of the Model-driven Technique, covering the utilization of differing modeling levels as well as the Sirius tool in terms of validation and event-driven models The main aim of integrating these tools and implementing a robust yet convenient way of making the sensor-based network secure, efficient and reliable irrespective of the area of installation – whether it is being configured in a smart home setting or a big industrial area, the MDE based frameworks focus on the abstract level things which are common in the most configuration yet ensuring the functional and non-functional concerns are also met concretely. [42]. This chapter focuses on providing exemplary answers to how traditional models of software development are changing to become more agile and flexible. How does real-time data collection as well as anomaly detection contribute towards system stability? Finally, the significant role that security for communication channels plays in guarding against exposure to sensitive information.

The Traditional Software Development Life Cycle (SDLC) involves the waterfall model [43], which is a sequential way of doing things in software development, which is a phased process where each stage has to be finished before moving into another thereby making it difficult for the project if a change occurs at any point after commencement [44]. However, Current methodologies (Modern) in the Software Development Life Cycle like Agile and DevOps have put more emphasis on gradual and repetitive development allowing delivery and integration throughout [45]. They stand out as more flexible to changes hence promoting collaboration and being alert all through while developing software [46].

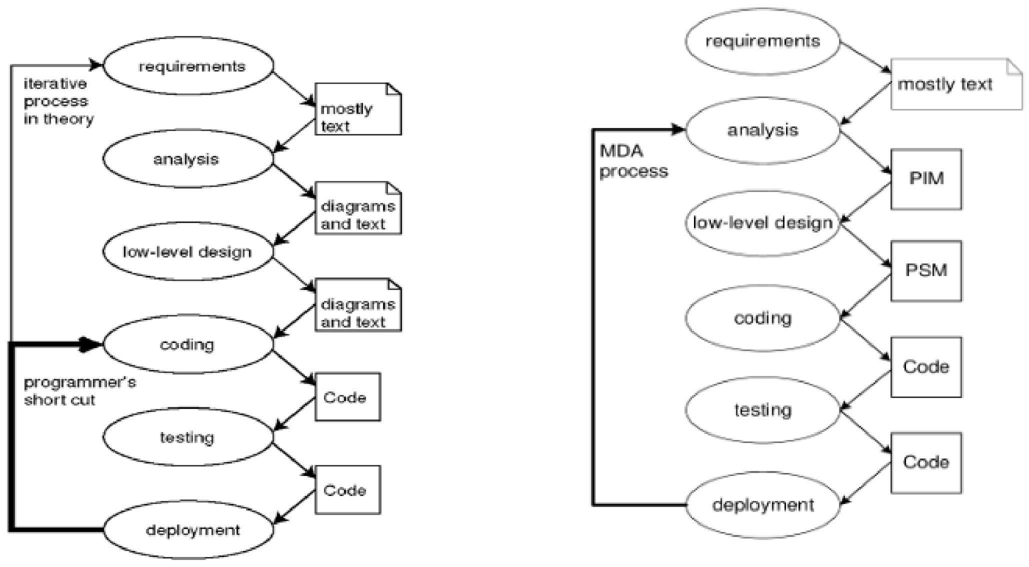


Figure 4: Traditional SDLC (left) Vs Modern SDLC (Right)

3.1. MDD based Security Framework

MDD is a standard change that helps to achieve higher degrees of abstraction and better visualization for complicated systems. MDD promotes better comprehension through the development of system models that operate at high levels of abstraction. These models and their transformations play crucial roles in automating the design, development, implementation, testing, maintenance, and other software engineering tasks. The key to the model-driven approach is its capacity to simplify difficult situations while delivering greater visualization and comprehension. To do this, one must first define and create a formal model, sometimes referred to as a domain model or a platform-independent model. Notably, the model-driven method has applicability across many fields, including the creation of information systems, small software companies, large-scale operational applications, and mixed interactive systems. Low-level design and coding are the focal points of the conventional software development life cycle, in this conventional method, as soon as coding begins, the importance of documentation and architectural/design diagrams decreases. As a result, only the code is altered as the system evolves, which widens the gap between the code and the accompanying documentation and diagrams. Whereas in the MDD life cycle, where modeling efforts are what guide the software development process. This method creates formal models that are understandable to computers as artifacts. The following are the main models used in model-driven architecture (MDA): There are three types of models: PIMs (Platform Independent Models), PSMs (Platform Specific Models), and Java. Lang. Code. MDA transforms the software development process by depending on this model-centric paradigm, providing improved maintenance, more agility, and higher flexibility to changing needs.

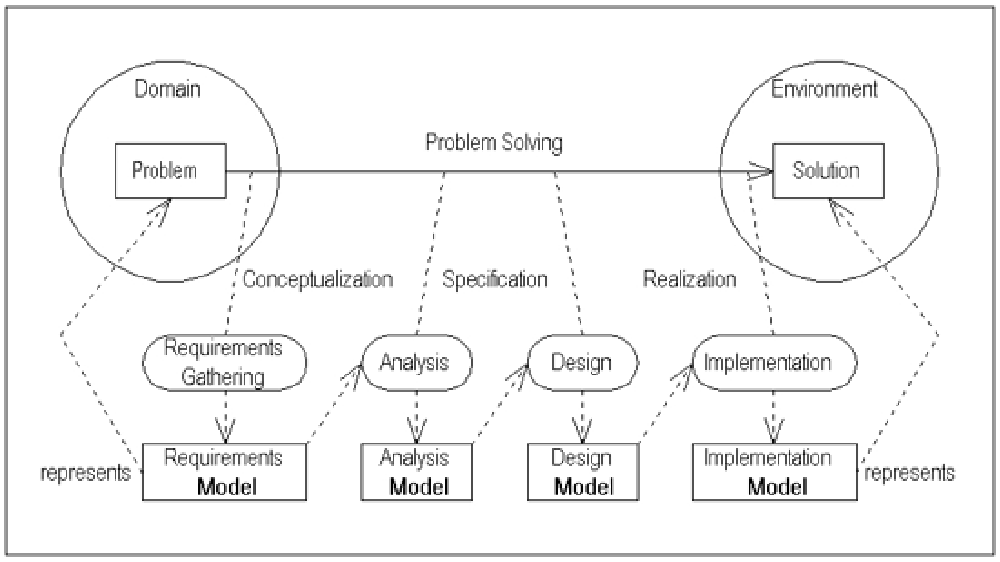


Figure 5: Overview of Model-Driven Development

The methodology uses Model Driven Engineering (MDE) to increase the level of abstraction at which developers work with complex systems. This way, it becomes easy for one to understand better what is underpinning their codes and those of other individuals [47]. When MDE is implemented, developers can come up with models of a system in its high form showing its structure, behavior, and interaction so that such models can subsequently transformed automatically into code that operates [48]. Less effort is exerted in coding manually by adopting this method, hence reducing errors as the visual models give an inclusive description of system purposes and changes [49]. Moreover, the approach exploits MDE for purposes of making the construction, testing, and verification of systems easier especially with tools such as Node-Red that enable the creation of IoT flows with the visual as well as support the integration of different components [50]. MDE increases the effectiveness, stability, and dependability of the process of creating software thereby making it possible to manage complex systems better and at the same time ensure they are secure and keep working as expected [47].

The methodology includes a systematic approach to developing and applying models using Sirius, Acceleo, and Eclipse. Using Eclipse, a metamodel at the M2 level is first created. This entails carefully identifying and describing the crucial concepts, things, and relationships that your system must represent. The characteristics, affiliations, inheritance, and constraints of the metamodel components are specified using the Eclipse Modelling Framework (EMF), guaranteeing a complete representation of the system's structure and semantics [52].

A case study is carried out using Sirius, an Eclipse-based tooling platform after the metamodel has been created. With Sirius, you may create specialized graphical modeling editors that are tailored to the needs and scope of your project. By defining perspectives, diagrams, and representations that are unique to your system, you may create a user interface that is both simple to use and well-suited for generating and modifying instances of the metamodel elements. The visual and interactive modeling experience made possible by the Sirius editor improves the efficiency and simplicity of model development. The next phase in your process is to use Acceleo to convert the models into text-based artifacts, especially to produce. JSON files when the case study in Sirius has been finished. Acceleo automates the transition of your models into text by acting as a potent model-to-text transformation language and generator. To do this, you create Acceleo templates and rules that get the necessary data from the models and produce the required. JSON files.

The process includes the step of importing the generated. JSON files into Node-RED for validation. A well-liked tool for modeling, simulating, and validating real-time systems is called Node-RED. You may use Node-RED strong encryption capabilities to validate the characteristics.

3.2. Proposed Metamodel

In our proposed framework a generic meta-model for End-to-End Encryption is shown in Figure 7. The proposed meta model provides a comprehensive framework for End-to-End Encryption (E2EE) which makes sure that information sent between parties is encrypted at the source and only decoded at the recipient's end, shielding it from any interceptors even when the communication channel is open. For applications to implement robust E2EE, this model captures a variety of entities and their interrelationships.

The proposed metamodel contains several entities, each playing a specific role in the encryption and decryption processes. The main entities include Encryption, Decryption, Message Payload, and Protocol. These entities are interconnected, forming the backbone of the encryption architecture. The Encryption entity handles the process of converting plaintext into cipher text using a specific Algorithm. On the other hand, the Decryption entity does the opposite, applying the same or a similar method to translate the cipher text back to plaintext. All of the top-level structure is defined within the Flow class. Encapsulating many classes including Sensor Input, Algorithm, Message Payload, Encryption, Protocol, and so on, it functions as a meta-entity. Explaining how data flows from inputs to outputs and how nodes are connected within the flow structure. It makes sure that all parts, whether they are used for encryption or other data processing duties, cooperate to produce the intended results.

The sensor input, which can come from virtual or actual sensors, is where data enters. Alphanumeric data indicating measurements or status updates from various scenarios are provided by these sources. They set off events within the flow, starting procedures like using algorithmic scripting to encrypt data. In applications like IoT or industrial automation, each input is uniquely identified by a sensor ID and classified by kind, which is essential for comprehending and reacting to real-world situations. The scripting or logic used for encryption procedures is referred to as the Algorithm class. It outlines the transformation and encryption processes used to protect the integrity of data. To meet security needs, we incorporated characteristics like key sizes and encryption standards in addition to using cryptographic techniques to secure sensitive data. The data being processed within the flow is represented by the message payload. It contains data that nodes alter, convert, or encrypt by the algorithm's definition. This class makes sure that the integrity of the data is preserved over its entire travel through the system. Payloads of messages are prepared by the specifications of the application, like XML or JSON, to enable easy integration and interoperability with other systems or services.

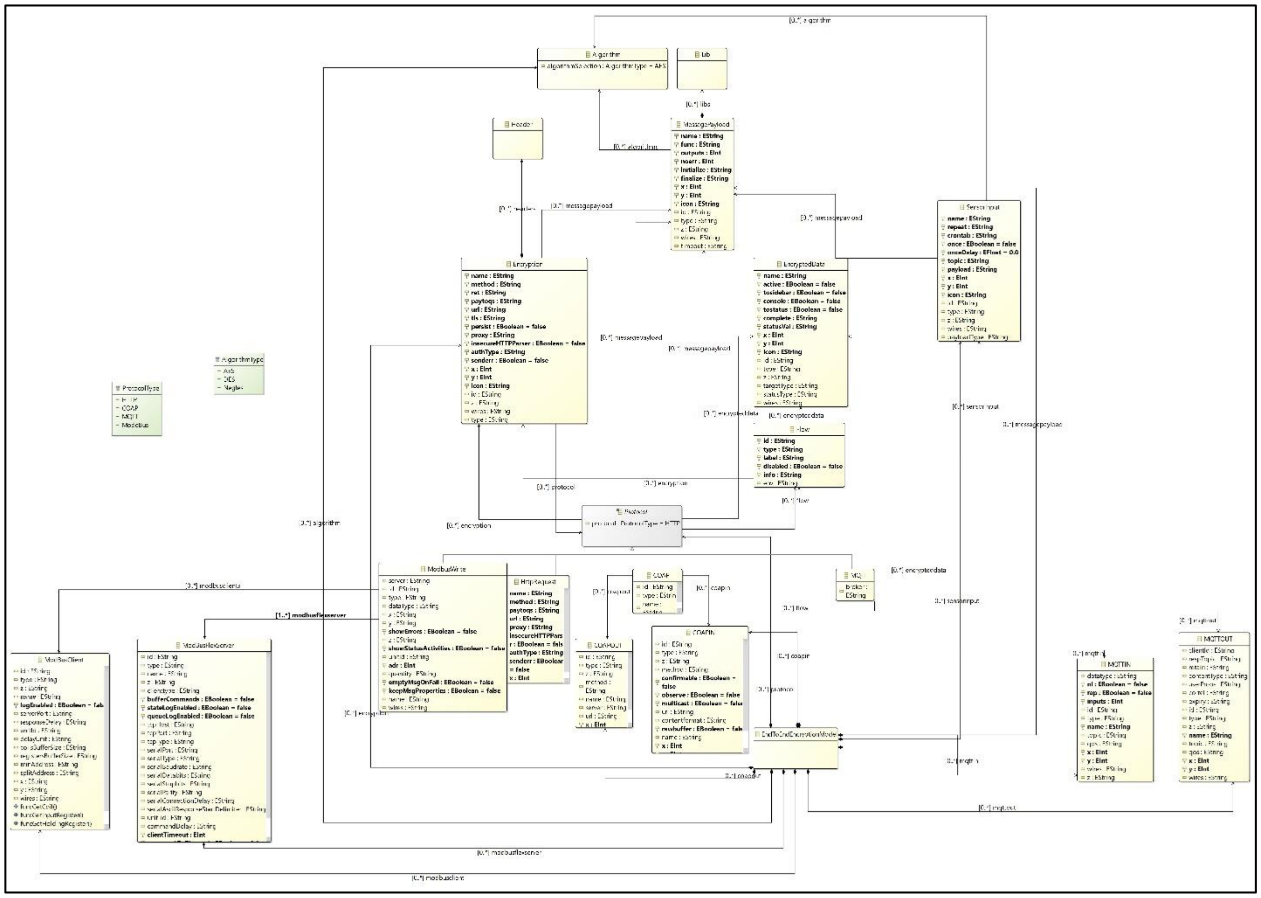


Figure 6: Proposed Meta-Model

Ensuring the confidentiality and protection of sensitive information from unauthorized access is an essential component of data processing. Data encryption before transmission is specified by the encryption techniques, which are implemented and configured by the Encryption class. To ensure safe communication and storage procedures, it incorporates encryption standards and key management procedures. The output of the encryption procedure is referred to as encrypted data. It symbolizes information that has been safely encrypted using the encryption techniques and algorithms that have been outlined in the Encryption Data class. Additionally, the flow connects with other protocols (Protocol), including HTTP, MQTT, CoAP, Modbus, and CoAP, to enable data interchange and communication with external systems.

The class that communicates with Modbus-based systems or devices is represented by a Modbus Client. Employing the Modbus protocol makes communication easier by reading and publishing data to Modbus servers (Modbus Server). Equipment and industrial automation systems can be seamlessly integrated thanks to the Modbus Client class, which offers options for data types, device addresses, and communication settings. Modbus Server uses the Modbus protocol to function as a server. It acts as a central repository to process variables, control parameters, and other data used in industrial automation and control applications by responding to requests from Modbus clients (Modbus Client) to read or write data. To provide dependable and effective data interchange with Modbus clients, the Modbus Server class controls communication protocols, security settings, and data access rights. Modbus Write is a meta-class that describes the operations in charge of writing data to Modbus-based systems or devices.

CoAP is a lightweight communication protocol that defines characteristics that make use of UDP or DTLS protocols to efficiently exchange data, such as endpoints, message formats, and security configurations. Views from meta-modeling shed light on the connections between CoAP and other classes, such as COAP IN and COAP OUT, and show how CoAP protocols affect interoperability and communication within the larger flow structure. COAP IN stands for input nodes that get CoAP messages from other services or devices. It specifies properties including payload handling, message parsing rules, and endpoint settings that allow CoAP-based communication to be easily integrated into applications. COAP OUT designates output nodes that are in charge of transmitting CoAP messages to other services or devices.

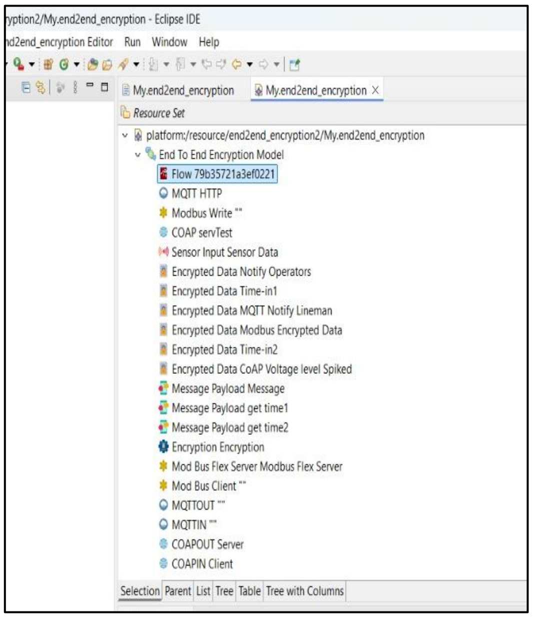
An HTTP request is in charge of sending HTTP requests to web servers or APIs. It allows integration with many HTTP-based systems and services by including configurations for URL endpoints, request methods (GET, POST, PUT, DELETE), headers, and login credentials.

A message broker or server that carries out the MQTT (Message Queuing Telemetry Transport) protocol is denoted as an MQTT Broker. It enables MQTT clients (MQTT IN and MQTT OUT) connected to the broker to communicate and exchange messages asynchronously. The MQTT Broker class ensures dependable and scalable MQTT communication within IoT applications by including configurations for the broker address, port number, security settings (TLS/SSL), and message persistence options. When input nodes subscribe to MQTT topics and receive messages from MQTT clients or devices linked to the MQTT broker, they are represented as MQTT IN nodes. Based on subscribed topics and message payloads, these nodes process incoming MQTT messages, initiating actions or processing inside the flow. The MQTT IN class facilitates the smooth integration of MQTT-based communication by providing configurable message parsing rules, topic subscriptions, and quality of service (QoS) settings. Output nodes that publish MQTT messages to MQTT topics housed on the MQTT broker are represented by MQTT OUT. Based on data processing within the flow, these nodes start MQTT messages, enabling bidirectional communication with MQTT clients or devices.

Together, these classes describe the functionality, structure, and integration capabilities of this encryption meta-model, allowing developers to efficiently design, implement, and oversee complex automation, data processing, and communication systems. Every class has a distinct function in overseeing data flow, protocol communication, security, and interoperability, facilitating an extensive array of applications spanning many industries including industrial automation and the Internet of Things.

3.2.1. TreeView of Metamodel

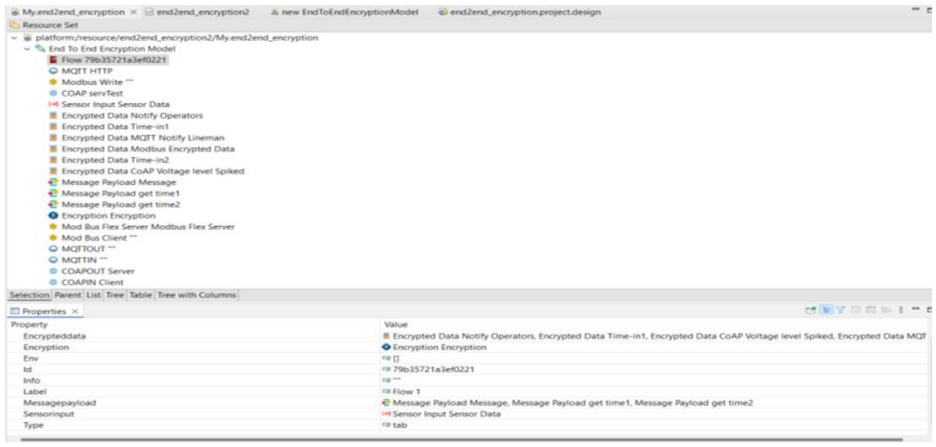
The TreeView feature in Obeo Eclipse offers a potent method for visualizing and interacting with a metamodel. Users may traverse, study, and change the elements and connections of the metamodel using TreeView, which displays its hierarchical structure. The TreeView in Obeo Eclipse shows a tree-like depiction of the metamodel's ideas and associations when the metamodel has been loaded. Each metamodel component is represented as a node in the tree, with its name and type visible for quick recognition. Users may browse around the metamodel's structure by expanding and collapsing nodes, which reveals the hierarchical organization. Different forms of interaction are supported by TreeView. Selecting a node allows users to access specific details about the associated metamodel elements, including their attributes, operations, and references. Additionally, they can operate on the elements by adding new instances, changing properties, or forming connections between them. Working with metamodels is made simple and comprehensive by TreeView in Obeo Eclipse. It offers visual cues and icons to denote the kind, visibility, and connections of metamodel elements, facilitating comprehension and interpretation of the structure and semantics. Users may better comprehend the metamodel's structure, construct instances that correspond to its stated principles, and develop relationships by utilizing TreeView. By instantiating relevant concepts from our proposed meta-model and setting relationships among instances accordingly, this M1-level model maps the requirements of a given case study. In Figure 8, a tree view editor is developed to make the hierarchy of your case study before implementing it in Sirius. The basic purpose of the tree view is to add or delete some attributes or classes in the meta-model according to your case study [53].


yption2/My.end2end_encryption - Eclipse IDE
d2end_encryption Editor Run Window Help
My.end2end_encryption
My.end2end_encryption
Resource Set
Selection Parent List Tree Table Tree with Columns

platform:/resource/end2end_encryption2/My.end2end_encryption
End To End Encryption Model
Flow 79b35721a3ef0221
MQTT HTTP
Modbus Write ""
COAP servTest
Sensor Input Sensor Data
Encrypted Data Notify Operators
Encrypted Data Time-in1
Encrypted Data MQTT Notify Lineman
Encrypted Data Modbus Encrypted Data
Encrypted Data Time-in2
Encrypted Data CoAP Voltage level Spiked
Message Payload Message
Message Payload get time1
Message Payload get time2
Encryption Encryption
Mod Bus Flex Server Modbus Flex Server
Mod Bus Client ""
MQTTOUT ""
MQTTIN ""
COAPOUT Server
COAPIN Client

Figure 7. Treeview of Metamodel

Clicking on the "Flow" node in Eclipse's TreeView opens an enlarged view that shows all the attributes and information related to the "Flow" concept as shown in Figure 9. TreeView makes it simple for users to explore and grasp the complexities of the "Flow" notion, providing a greater comprehension of its attributes and behavior inside the metamodel.

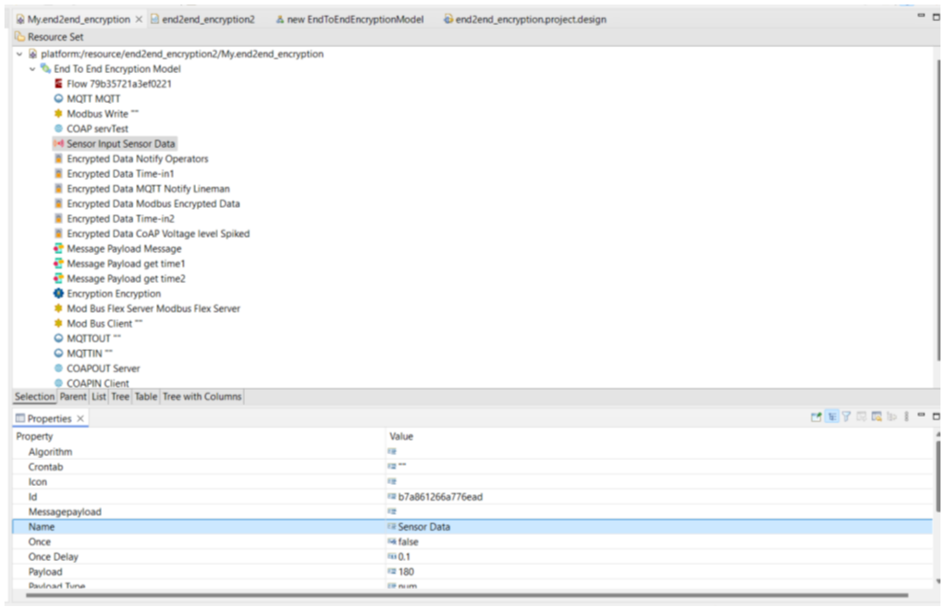


```
My.end2end_encryption ×   end2end_encryption2   new EndToEndEncryptionModel   end2end_encryption.project.design
Resource Set
⌄ platform:/resource/end2end_encryption2/My.end2end_encryption
  ⌄ End To End Encryption Model
      Flow 79b35721a3ef0221
      MQTT HTTP
      Modbus Write ""
      COAP servTest
      Sensor Input Sensor Data
      Encrypted Data Notify Operators
      Encrypted Data Time-in1
      Encrypted Data MQTT Notify Lineman
      Encrypted Data Modbus Encrypted Data
      Encrypted Data Time-in2
      Encrypted Data CoAP Voltage level Spiked
      Message Payload Message
      Message Payload get time1
      Message Payload get time2
      Encryption Encryption
      Mod Bus Flex Server Modbus Flex Server
      Mod Bus Client ""
      MQTTOUT ""
      MQTTIN ""
      COAPOUT Server
      COAPIN Client
Selection | Parent | List | Tree | Table | Tree with Columns
Properties ×
```

Property	Value
Encrypteddata	Encrypted Data Notify Operators, Encrypted Data Time-in1, Encrypted Data CoAP Voltage level Spiked, Encrypted Data MQT
Encryption	Encryption Encryption
Env	[]
Id	79b35721a3ef0221
Info	""
Label	Flow 1
Messagepayload	Message Payload Message, Message Payload get time1, Message Payload get time2
Sensorinput	Sensor Input Sensor Data
Type	tab

Figure 8. Properties of Flow Node.

Clicking on the "Sensor Data" node in Eclipse's TreeView opens an enlarged view that shows all the attributes and information related to the "Sensor Data " concept as shown in Figure 10. This provides details about the " Sensor Data " an element of the metamodel, such as attributes, operations, and references.

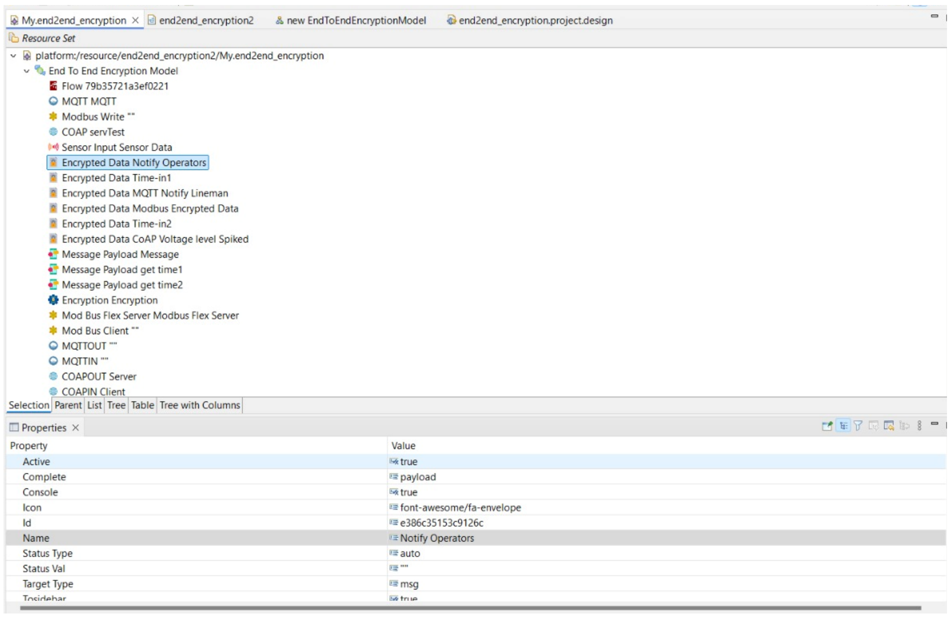


```
My.end2end_encryption ×   end2end_encryption2   new EndToEndEncryptionModel   end2end_encryption.project.design
Resource Set
⌄ platform:/resource/end2end_encryption2/My.end2end_encryption
  ⌄ End To End Encryption Model
      Flow 79b35721a3ef0221
      MQTT MQTT
      Modbus Write ""
      COAP servTest
      Sensor Input Sensor Data
      Encrypted Data Notify Operators
      Encrypted Data Time-in1
      Encrypted Data MQTT Notify Lineman
      Encrypted Data Modbus Encrypted Data
      Encrypted Data Time-in2
      Encrypted Data CoAP Voltage level Spiked
      Message Payload Message
      Message Payload get time1
      Message Payload get time2
      Encryption Encryption
      Mod Bus Flex Server Modbus Flex Server
      Mod Bus Client ""
      MQTTOUT ""
      MQTTIN ""
      COAPOUT Server
      COAPIN Client
Selection | Parent | List | Tree | Table | Tree with Columns
Properties ×
```

Property	Value
Algorithm	
Crontab	""
Icon	
Id	b7a861266a776ead
Messagepayload	
Name	Sensor Data
Once	false
Once Delay	0.1
Payload	180
Payload Type	num

Figure 9. Properties of Sensor Data

Figure 11 provides details about the " Encrypted Data " an element of the metamodel, such as attributes, operations, and references. TreeView makes it simple for users to explore and grasp the complexities of the " Encrypted Data " notion.



My.end2end_encryption × | end2end_encryption2 | new EndToEndEncryptionModel | end2end_encryption.project.design

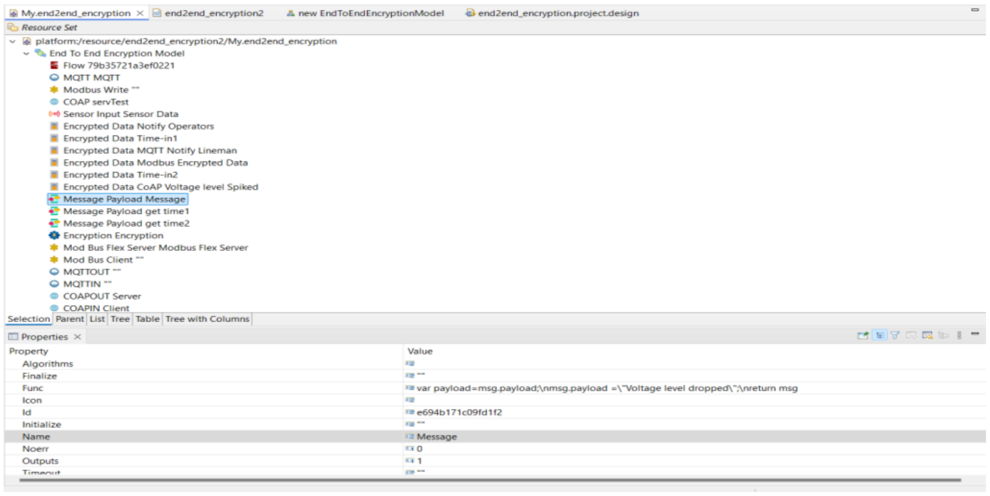
Resource Set

```
platform:/resource/end2end_encryption2/My.end2end_encryption
  End To End Encryption Model
    Flow 79b35721a3ef0221
    MQTT MQTT
    Modbus Write ""
    COAP servTest
    Sensor Input Sensor Data
    Encrypted Data Notify Operators
    Encrypted Data Time-in1
    Encrypted Data MQTT Notify Lineman
    Encrypted Data Modbus Encrypted Data
    Encrypted Data Time-in2
    Encrypted Data CoAP Voltage level Spiked
    Message Payload Message
    Message Payload get time1
    Message Payload get time2
    Encryption Encryption
    Mod Bus Flex Server Modbus Flex Server
    Mod Bus Client ""
    MQTTOUT ""
    MQTTIN ""
    COAPOUT Server
    COAPIN Client
```

Property	Value
Active	true
Complete	payload
Console	true
Icon	font-awesome/fa-envelope
Id	e386c35153c9126c
Name	Notify Operators
Status Type	auto
Status Val	""
Target Type	msg
Tosidebar	true

Figure 10. Properties of Encrypted Data

Clicking on the "Message Payload" node in Eclipse's TreeView opens an enlarged view that shows all the attributes and information related to the " Message Payload" concept as shown in Figure 12.



My.end2end_encryption × | end2end_encryption2 | new EndToEndEncryptionModel | end2end_encryption.project.design

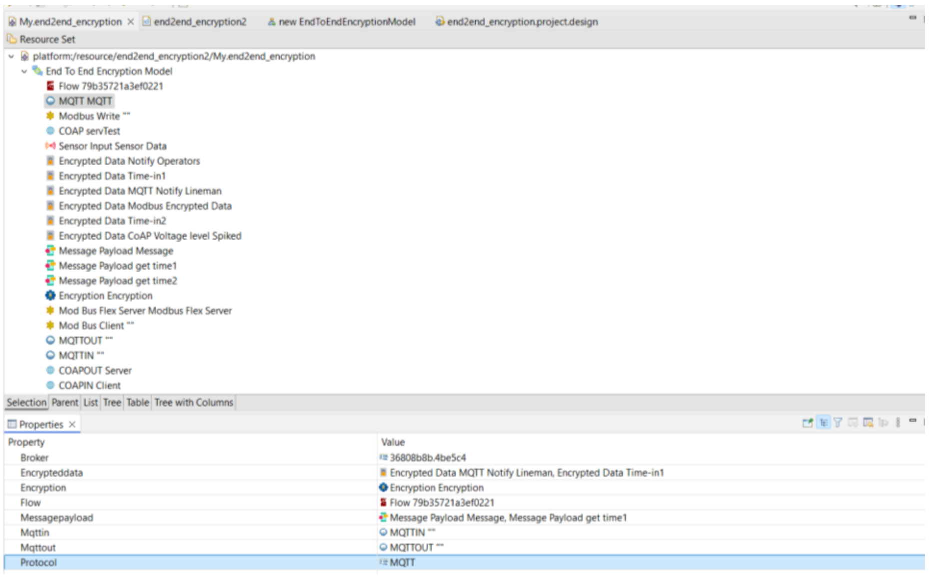
Resource Set

```
platform:/resource/end2end_encryption2/My.end2end_encryption
  End To End Encryption Model
    Flow 79b35721a3ef0221
    MQTT MQTT
    Modbus Write ""
    COAP servTest
    Sensor Input Sensor Data
    Encrypted Data Notify Operators
    Encrypted Data Time-in1
    Encrypted Data MQTT Notify Lineman
    Encrypted Data Modbus Encrypted Data
    Encrypted Data Time-in2
    Encrypted Data CoAP Voltage level Spiked
    Message Payload Message
    Message Payload get time1
    Message Payload get time2
    Encryption Encryption
    Mod Bus Flex Server Modbus Flex Server
    Mod Bus Client ""
    MQTTOUT ""
    MQTTIN ""
    COAPOUT Server
    COAPIN Client
```

Property	Value
Algorithms	
Finalize	""
Func	var payload=msg.payload;\nmsg.payload =\"Voltage level dropped\";\nreturn msg
Icon	
Id	e694b171c09fd1f2
Initialize	""
Name	Message
Noerr	0
Outputs	1
Timeout	""

Figure 11. Properties of Message Payload

Figure 13 shows all the attributes and information related to the "MQTT" concept. This provides details about the "MQTT" an element of the metamodel, such as attributes, operations, and references. TreeView makes it simple for users to explore and grasp the complexities of the "MQTT" notion, providing a greater comprehension of its attributes and behavior inside the metamodel.

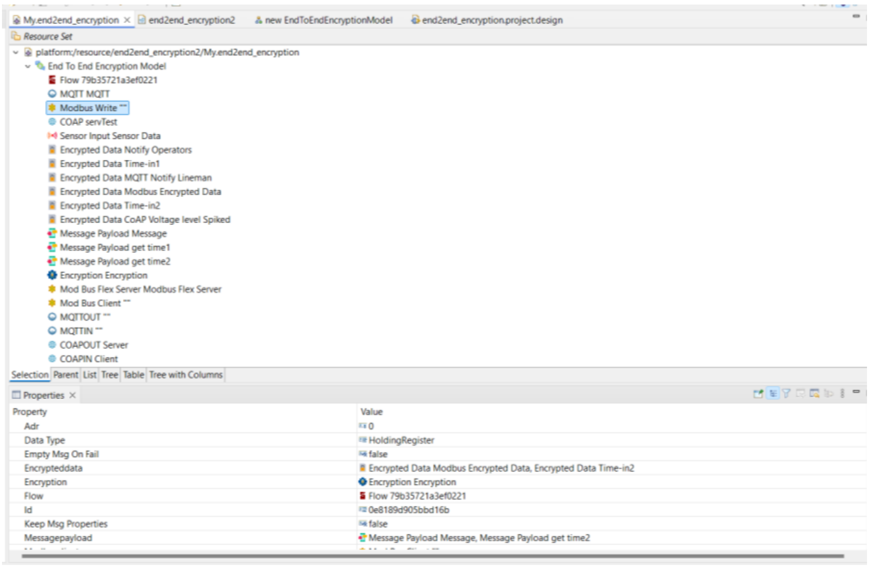


platform:/resource/end2end_encryption2/My.end2end_encryption
End To End Encryption Model
Flow 79b35721a3ef0221
MQTT MQTT
Modbus Write ""
COAP servTest
Sensor Input Sensor Data
Encrypted Data Notify Operators
Encrypted Data Time-in1
Encrypted Data MQTT Notify Lineman
Encrypted Data Modbus Encrypted Data
Encrypted Data Time-in2
Encrypted Data CoAP Voltage level Spiked
Message Payload Message
Message Payload get time1
Message Payload get time2
Encryption Encryption
Mod Bus Flex Server Modbus Flex Server
Mod Bus Client ""
MQTTOUT ""
MQTTIN ""
COAPOUT Server
COAPIN Client

Property	Value
Broker	36808b8b.4be5c4
Encrypteddata	Encrypted Data MQTT Notify Lineman, Encrypted Data Time-in1
Encryption	Encryption Encryption
Flow	Flow 79b35721a3ef0221
Messagepayload	Message Payload Message, Message Payload get time1
Mqttin	MQTTIN ""
Mqttout	MQTTOUT ""
Protocol	MQTT

Figure 12. Properties of MQTT

Figure 14 provides details about the " Modbus Write " an element of the metamodel, such as attributes, operations, and references.

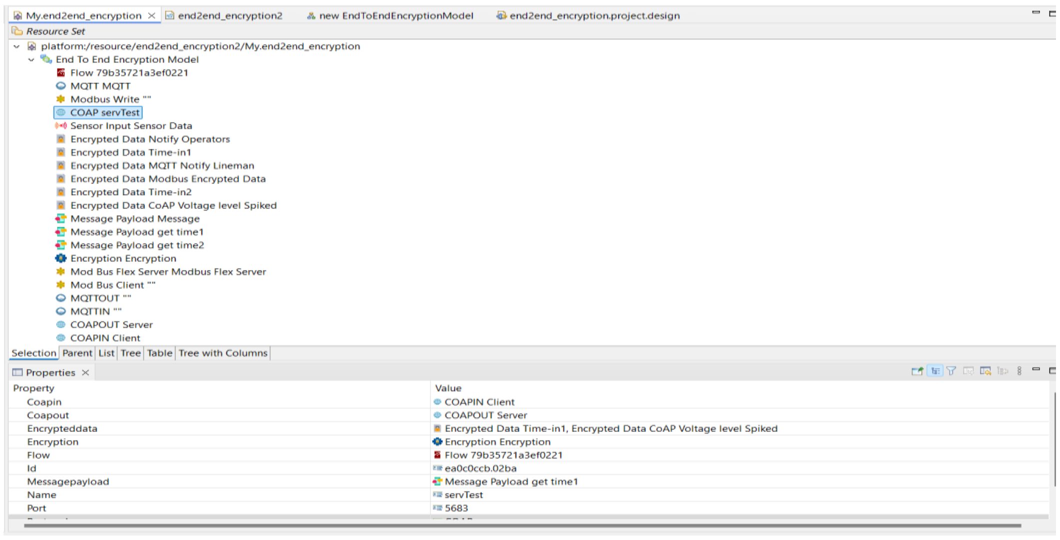


platform:/resource/end2end_encryption2/My.end2end_encryption
End To End Encryption Model
Flow 79b35721a3ef0221
MQTT MQTT
Modbus Write ""
COAP servTest
Sensor Input Sensor Data
Encrypted Data Notify Operators
Encrypted Data Time-in1
Encrypted Data MQTT Notify Lineman
Encrypted Data Modbus Encrypted Data
Encrypted Data Time-in2
Encrypted Data CoAP Voltage level Spiked
Message Payload Message
Message Payload get time1
Message Payload get time2
Encryption Encryption
Mod Bus Flex Server Modbus Flex Server
Mod Bus Client ""
MQTTOUT ""
MQTTIN ""
COAPOUT Server
COAPIN Client

Property	Value
Adr	0
Data Type	HoldingRegister
Empty Msg On Fail	false
Encrypteddata	Encrypted Data Modbus Encrypted Data, Encrypted Data Time-in2
Encryption	Encryption Encryption
Flow	Flow 79b35721a3ef0221
Id	0e8189d905bbd16b
Keep Msg Properties	false
Messagepayload	Message Payload Message, Message Payload get time2

Figure 13. Properties of Modbus Write

The "COAP" node in Eclipse's TreeView opens an enlarged view that shows all the attributes and information related to the "COAP " concept as shown in Figure 9 - makes it simple for users to explore and grasp the complexities, providing a greater comprehension of its attributes and behavior inside the metamodel.


My.end2end_encryption
end2end_encryption2
new EndToEndEncryptionModel
end2end_encryption.project.design
Resource Set

platform:/resource/end2end_encryption2/My.end2end_encryption
- End To End Encryption Model
  - Flow 79b35721a3ef0221
  - MQTT MQTT
  - Modbus Write ""
  - COAP servTest
  - Sensor Input Sensor Data
  - Encrypted Data Notify Operators
  - Encrypted Data Time-in1
  - Encrypted Data MQTT Notify Lineman
  - Encrypted Data Modbus Encrypted Data
  - Encrypted Data Time-in2
  - Encrypted Data CoAP Voltage level Spiked
  - Message Payload Message
  - Message Payload get time1
  - Message Payload get time2
  - Encryption Encryption
  - Mod Bus Flex Server Modbus Flex Server
  - Mod Bus Client ""
  - MQTTOUT ""
  - MQTTIN ""
  - COAPOUT Server
  - COAPIN Client

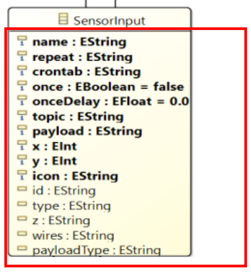
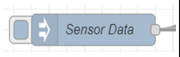
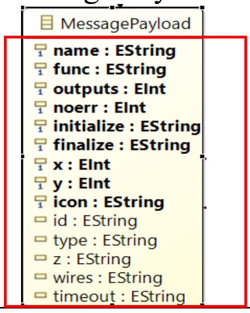
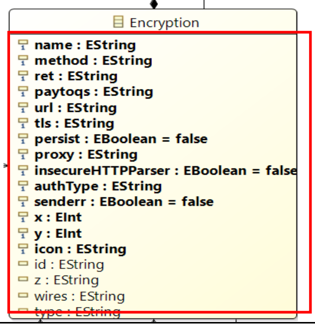
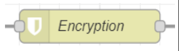
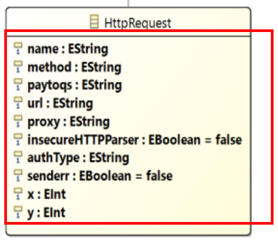
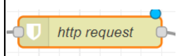
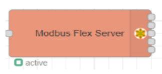
Property	Value
Coapin	COAPIN Client
Coapout	COAPOUT Server
Encrypteddata	Encrypted Data Time-in1, Encrypted Data CoAP Voltage level Spiked
Encryption	Encryption Encryption
Flow	Flow 79b35721a3ef0221
Id	ea0c0ccb.02ba
Messagepayload	Message Payload get time1
Name	servTest
Port	5683

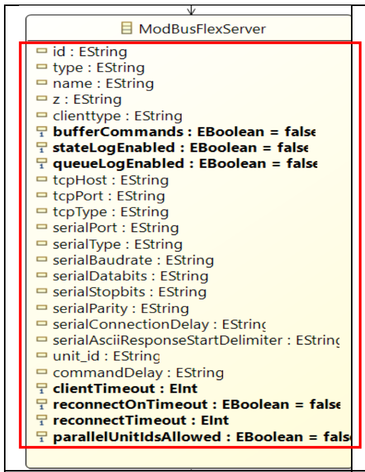
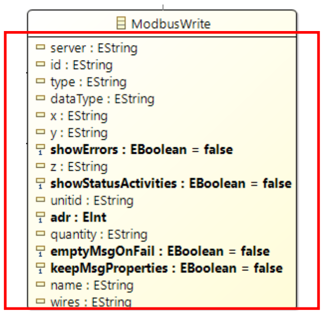
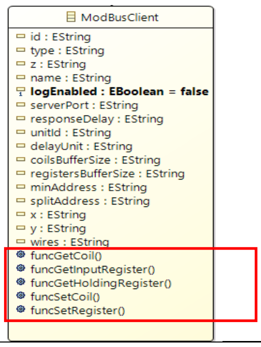
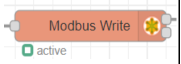
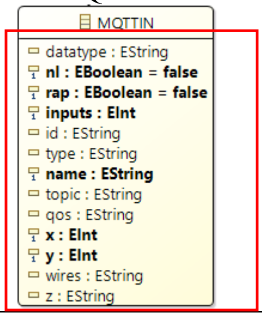
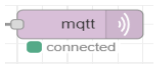
Figure 14. Properties of COAP

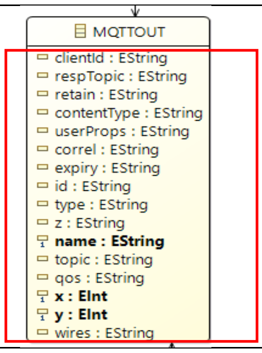
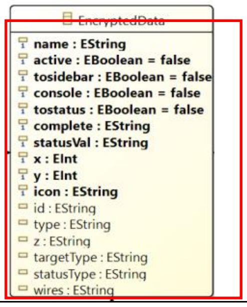
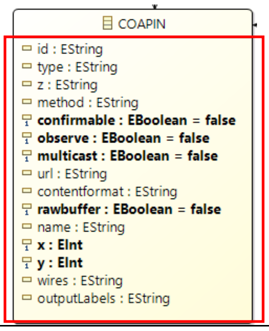
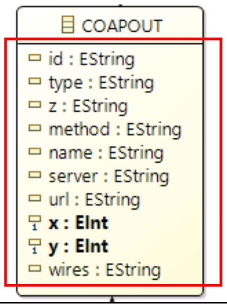
3.2.2. Transformation Rules

In model transformation procedures, transformation rules are essential. The logic and procedures necessary to transform input models into desired output models or textual artifacts are specified by these rules. The components, properties, and relationships in the input model are mapped and converted into equivalent elements, attributes, and relationships in the output model or text according to transformation rules. To explain transformation logic, transformation rules are often expressed using specialized transformation languages or tools like Acceleo, ATL, or QVT. Developers may automate and expedite the model transformation process by utilizing transformation rules, resulting in accurate and reliable conversions throughout the different phases of the software development lifecycle [54]. Our approach uses transformation rules to automatically change the M2 level metamodel, acting as the input, into Node-Red, the intended output. The elements, relationships, and constraints described in the metamodel are mapped to their appropriate representations in Node-Red according to these transformation rules, which create a systematic method for doing so. In this study, the M1 level models are transformed into a JSON file format, which displays the model in textual form. The MWS Framework uses the Acceleo transformation language to carry out the transformation process. While components in the metamodel are defined by classes, objects, and relationships, components in Node-Red are defined by templates, locations, and edges. The metamodel uses attributes and methods to specify properties, whereas Node-Red uses local and global variables to express them. In Node-Red, associations from the metamodel are converted to wires that connect the various components. The Node-Red process transforms constraints from the metamodel into guard conditions for transitions. In contrast to Node-Red, which allows automated verification through a runtime compilation mechanism, verification in the metamodel is often carried out manually through testing and debugging. Finally, Node-Red imports and exports the model as a JSON file suited for the simulation of the IOT network while the metamodel is stored as source code files in programming languages like Java, JS, C++, or Python [55].

Table 3. Transformation Rules

Model Artifacts	ICON	Mapping	Description
Sensor Input  SensorInput name : EString repeat : EString crontab : EString once : EBoolean = false onceDelay : EFloat = 0.0 topic : EString payload : EString x : EInt y : EInt icon : EString id : EString type : EString z : EString wires : EString payloadType : EString		Class→ Attributes→ Name	Fetching the name of attributes with their corresponding values by first targeting the corresponding class
Message Payload  MessagePayload name : EString func : EString outputs : EInt noerr : EInt initialize : EString finalize : EString x : EInt y : EInt icon : EString id : EString type : EString z : EString wires : EString timeout : EString		Class→ Attributes→ Name	Fetching the name of attributes with their corresponding values by first targeting the corresponding class
Encryption  Encryption name : EString method : EString ret : EString paytoqs : EString url : EString tls : EString persist : EBoolean = false proxy : EString insecureHTTPParser : EBoolean = false authType : EString senderr : EBoolean = false x : EInt y : EInt icon : EString id : EString z : EString wires : EString type : EString		Class→ Attributes→ Name	Fetching the name of attributes with their corresponding values by first targeting the corresponding class
HTTP request  HttpRequest name : EString method : EString paytoqs : EString url : EString proxy : EString insecureHTTPParser : EBoolean = false authType : EString senderr : EBoolean = false x : EInt y : EInt		Class→ Attributes→ Name	Fetching the name of attributes with their corresponding values by first targeting the corresponding class
Modbus Flex Server		Class→ Attributes→ Name	Fetching the name of attributes with their corresponding values by first

 ModBusFlexServer id : EString type : EString name : EString z : EString clienttype : EString **bufferCommands : EBoolean = false** **stateLogEnabled : EBoolean = false** **queueLogEnabled : EBoolean = false** tcpHost : EString tcpPort : EString tcpType : EString serialPort : EString serialType : EString serialBaudrate : EString serialDatabits : EString serialStopbits : EString serialParity : EString serialConnectionDelay : EString serialAsciiResponseStartDelimiter : EString unit_id : EString commandDelay : EString **clientTimeout : EInt** **reconnectOnTimeout : EBoolean = false** **reconnectTimeout : EInt** **parallelUnitIdsAllowed : EBoolean = fals**			targeting the corresponding class
Modbus Write  ModbusWrite server : EString id : EString type : EString dataType : EString x : EString y : EString **showErrors : EBoolean = false** z : EString **showStatusActivities : EBoolean = false** unitid : EString **adr : EInt** quantity : EString **emptyMsgOnFail : EBoolean = false** **keepMsgProperties : EBoolean = false** name : EString wires : EString & Modbus Client  ModBusClient id : EString type : EString z : EString name : EString **logEnabled : EBoolean = false** serverPort : EString responseDelay : EString unitId : EString delayUnit : EString coilsBufferSize : EString registersBufferSize : EString minAddress : EString splitAddress : EString x : EString y : EString wires : EString funcGetCoil() funcGetInputRegister() funcGetHoldingRegister() funcSetCoil() funcSetRegister()		Class →Attributes→ Name & Class →Operations→ Name	Fetching the name of attributes with their corresponding values by first targeting the corresponding class & Fetching the name of operations by first targeting the corresponding class
MQTT IN  MQTTIN datatype : EString **nl : EBoolean = false** **rap : EBoolean = false** **inputs : EInt** id : EString type : EString **name : EString** topic : EString qos : EString **x : EInt** **y : EInt** wires : EString z : EString		Class→ Attributes→ Name	Fetching the name of attributes with their corresponding values by first targeting the corresponding class
MQTT Out		Class→ Attributes→ Name	Fetching the name of attributes with their corresponding values by first

 MQTTOUT clientId : EString respTopic : EString retain : EString contentType : EString userProps : EString correl : EString expiry : EString id : EString type : EString z : EString name : EString topic : EString qos : EString x : EInt y : EInt wires : EString			targeting the corresponding class
Encrypted Data  EncryptedData name : EString active : EBoolean = false tosidebar : EBoolean = false console : EBoolean = false tostatus : EBoolean = false complete : EString statusVal : EString x : EInt y : EInt icon : EString id : EString type : EString z : EString targetType : EString statusType : EString wires : EString		Class→ Attributes→ Name	Fetching the name of attributes with their corresponding values by first targeting the corresponding class
COAP IN  COAPIN id : EString type : EString z : EString method : EString confirmable : EBoolean = false observe : EBoolean = false multicast : EBoolean = false url : EString contentformat : EString rawbuffer : EBoolean = false name : EString x : EInt y : EInt wires : EString outputLabels : EString		Class→ Attributes→ Name	Fetching the name of attributes with their corresponding values by first targeting the corresponding class
COAP OUT  COAPOUT id : EString type : EString z : EString method : EString name : EString server : EString url : EString x : EInt y : EInt wires : EString		Class→ Attributes→ Name	Fetching the name of attributes with their corresponding values by first targeting the corresponding class
<<Connections>>		ConnectedElement →Association Name ContainerClass→ Containments→ ContainingClass →name	Fetching the name of reference or containment to further fetch the corresponding attributes or operations

In short, the technique entails utilizing Eclipse to create a metamodel at the M2 level, using Sirius to perform a case study for customized graphical modeling, using Acceleo to convert the models into JSON files, and then verifying these files in Node-Red. This thorough and iterative process makes sure that system models are created accurately, that they are converted into textual representations, and that they are then validated utilizing the Node-Red compiler. We guarantee a smooth and automated process of building IoT models by precisely specifying these rules, which capture the sensors' data and semantics inherent in the M2-level metamodel. To analyze and validate the system modeled at a higher degree of abstraction and to close the gap between the abstract metamodel and the concrete Node-Red representation, we may leverage the rigorous verification capabilities of the tool. To make the transition from the metamodel to the Node-Red process easier, transformation rules are required.

CHAPTER 4

IMPLEMENTATION

In this chapter, a thorough analysis of Sirius and its application, as well as an in-depth examination of Acceleo transformation will be discussed. We will delve into Sirius' complexities to learn how it permits the construction of domain-specific languages (DSLs) and the building of customized graphical modeling editors. We will examine Sirius' capabilities and characteristics that enable simple and attractive visual representations of complex structures. I will also focus on Acceleo's transformation features and how they help with model-to-text transformations. This knowledge will empower readers to confidently employ these tools to produce expressive visual models and generate customized textual artifacts that cater to their specific needs.

4.1. Case Study 1

Smart Grid Station

Our case study is based on cybersecurity threats in critical infrastructures like grid stations in a smart city. [51]. Ensuring safety against threads is crucial in these infrastructures to guarantee smooth continuity of operations. These power grids provide electricity supply for residential zones, commercial sectors, and industrial areas. Deploying sensors in a grid assists in the smooth continuity of operations with the real-time data collected from all the components of the grid thus taking time accordingly. Any sort of security attack on this crucial data may cause disruption of services which worsens espionage for larger geopolitical damage. Safeguarding power grids from cyber-attacks ensures economic stability and public safety. In a power grid infrastructure, IoT sensors are deployed to capture voltage levels and assess the temperature of key components such as circuit breakers, power lines, and transformers. Sensors are also strategically installed to measure the stability metrics of power stations such as phase, frequency, and power factors to check the health status of the station. Overall line conditions are also checked with the help of automated IoT devices which help in identifying potential issues in a line such as congestion or faults.

Real-time data collected from the grid is collected and transmitted to the control center. This data is then monitored by a centralized system which identifies incoming data and checks trends to identify anomalies and hence initiates respective responses required to ensure the smooth operations of the grid. This data also helps the operators to make informed decisions and take timely actions to maintain the stability of the grid thus enhancing the overall efficiency of the system.

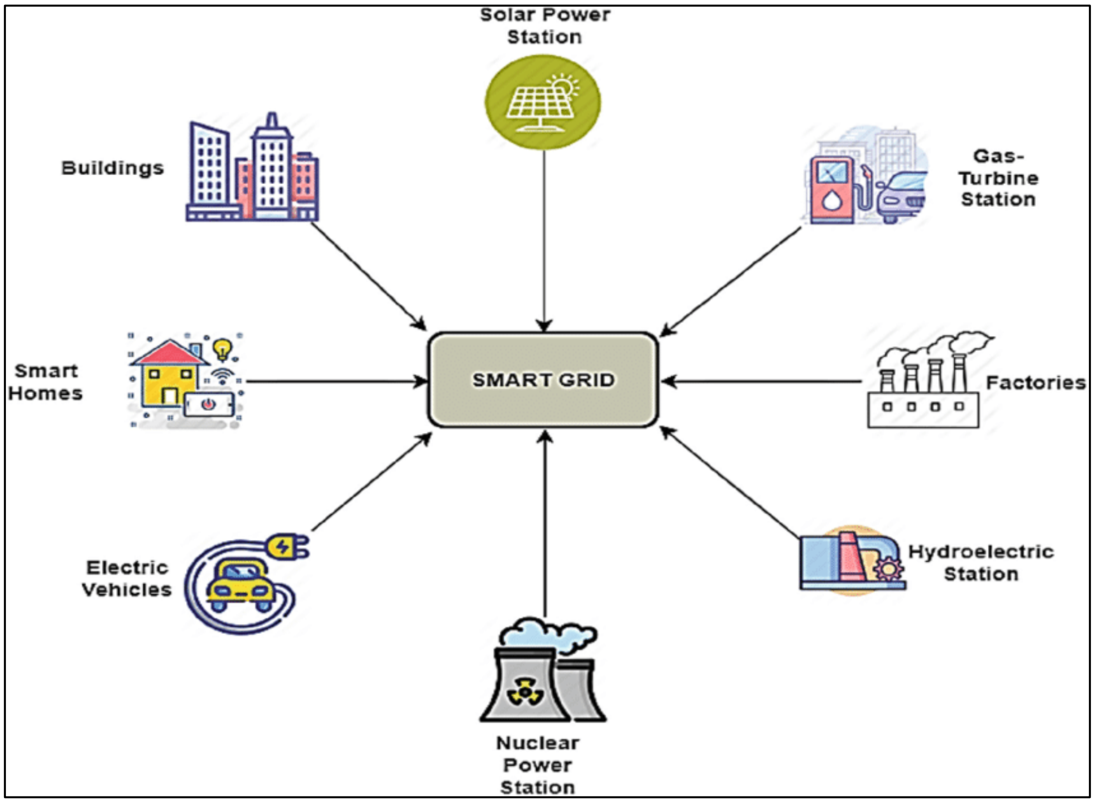


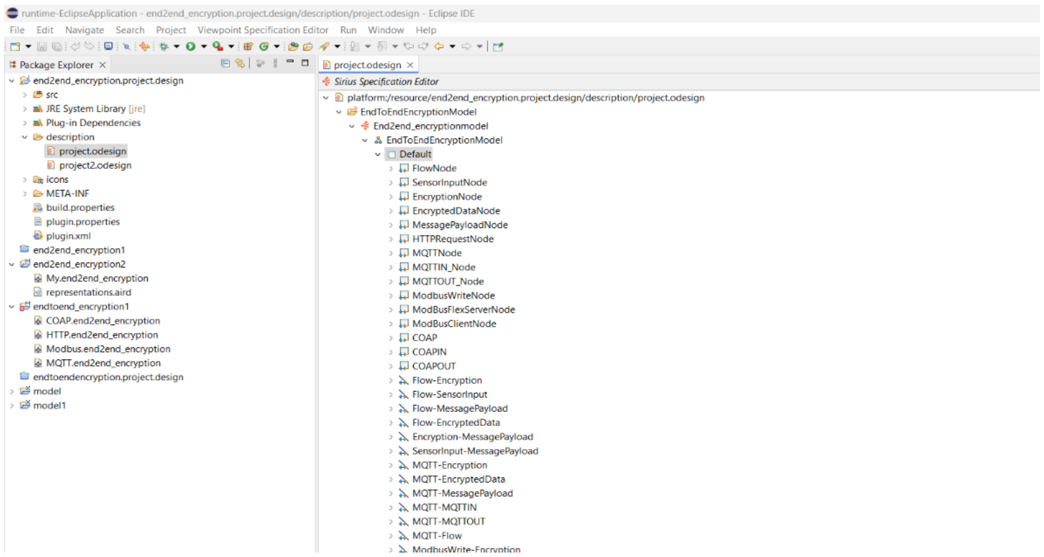
Figure 15. Smart Grid Station [52]

We will be implementing our adaptable framework for safeguarding the power grid from cyber-attacks. This robust encryption algorithm will be deployed to ensure that the data collected from servers reaches the servers with all its authenticity. The process involves gathering the data, processing it, and finally analyzing it to evaluate a finalized decree so that timely decisions can be made on whether the power grid needs any form of operator assistance or maintenance or not. In case any component of the grid doesn't need any upkeep, then transmission of false data could also be prohibited so that forged sensor values are not communicated to relevant control centers. The model safeguards the lines of the control system so that no loopholes are left in the communication pathway.

When validating the encryption made using the proposed framework, we have made use of AES as it is an enhanced version of DES and utilized key shift rounds. In our case study, we have used a block algorithm, a subtype of symmetric encryption. It operates on a group of bites having a fixed length. A secret key is used for decryption whereas block completion is awaited to complete the encryption or decryption process.

4.1.1. Sirius:

The open-source modeling tool Eclipse Sirius enables the development of unique graphical modeling workbenches in the Eclipse environment. Because of its novel methodology, users may create graphical editors and domain-specific languages (DSLs) that are customized to meet their unique requirements. There are generally multiple phases involved in implementing a case study with Eclipse Sirius. To identify the specific problem domain and the modeling ideas to be represented, the case study's requirements and scope are first specified. Next, Sirius' easy graphical modeling capabilities are used to develop the DSL for the case study. This entails establishing the metamodel, creating graphical representations, and defining the layout and behavior of elements. The case study's model instances are generated and modified using the custom graphical editor when the DSL is specified.


runtime-EclipseApplication - end2end_encryption.project.design/description/project.odesign - Eclipse IDE
File
Edit
Navigate
Search
Project
Viewpoint Specification Editor
Run
Window
Help
Package Explorer
project.odesign
Sirius Specification Editor

```
end2end_encryption.project.design
    src
    JRE System Library [jre]
    Plug-in Dependencies
    description
        project.odesign
        project2.odesign
    icons
    META-INF
    build.properties
    plugin.properties
    plugin.xml
end2end_encryption1
end2end_encryption2
    My.end2end_encryption
    representations.aird
endtoend_encryption1
    COAP.end2end_encryption
    HTTP.end2end_encryption
    Modbus.end2end_encryption
    MQTT.end2end_encryption
endtoendencryption.project.design
model
model1
```

```
platform:/resource/end2end_encryption.project.design/description/project.odesign
  EndToEndEncryptionModel
    End2end_encryptionmodel
      EndToEndEncryptionModel
        Default
          FlowNode
          SensorInputNode
          EncryptionNode
          EncryptedDataNode
          MessagePayloadNode
          HTTPRequestNode
          MQTTNode
          MQTTIN_Node
          MQTTOUT_Node
          ModbusWriteNode
          ModBusFlexServerNode
          ModBusClientNode
          COAP
          COAPIN
          COAPOUT
          Flow-Encryption
          Flow-SensorInput
          Flow-MessagePayload
          Flow-EncryptedData
          Encryption-MessagePayload
          SensorInput-MessagePayload
          MQTT-Encryption
          MQTT-EncryptedData
          MQTT-MessagePayload
          MQTT-MQTTIN
          MQTT-MQTTOUT
          MQTT-Flow
          ModbusWrite-Encryption
```

Figure 16. Treeview of Sirus

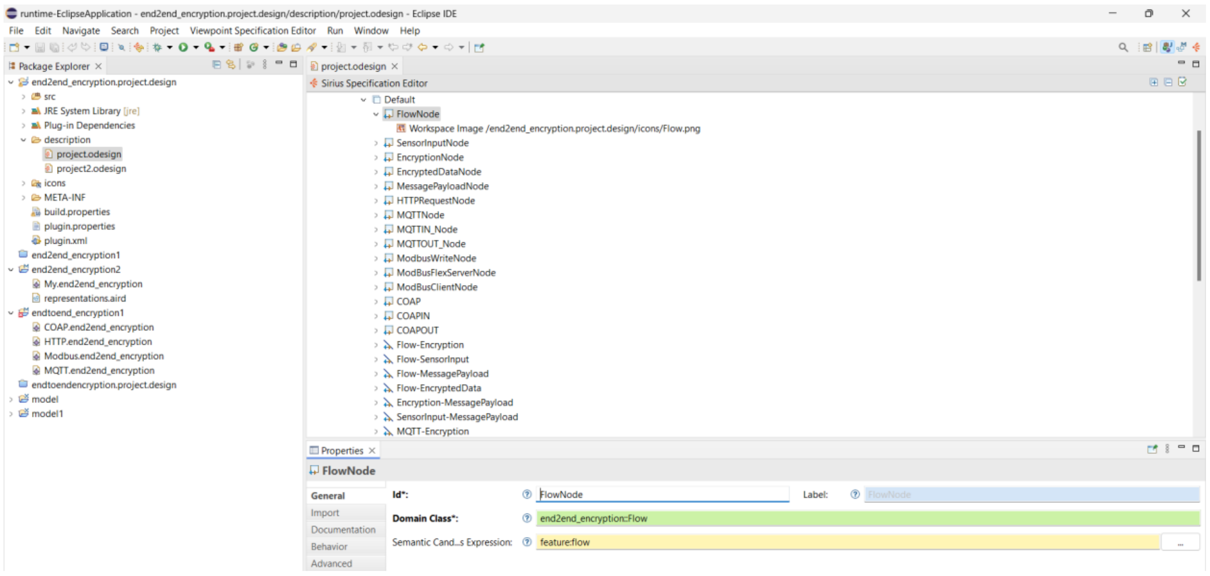
In this Figure, a network of nodes and their connections are graphically displayed to show how the system's interconnected components relate to one another. The graphic is purposefully made to explain the underlying architecture simply and understandably. Each node in the Sirius diagram has specific properties that are accessible by clicking on the node. For instance, when you click on the Flow node, a properties panel with important features is displayed, allowing you to efficiently customize the node's depiction [57].

The following fields are present in the Flow node's properties panel:

1. ID: To distinguish the Flow node from other nodes in the diagram and to assist in identifying it, you may give it a special identity using this field.
2. Domain Class: The domain class linked to the Flow node can be specified here. The domain class establishes the node's fundamental data structure and behaviour, guaranteeing the node's proper interaction with other system elements.
3. Semantic Candidates Expression: You can specify the possible semantic candidates for the

4. Flow node using the robust expression language offered by this field. The data components or entities that are allowed to be depicted in the diagram as Flow nodes can be managed by specifying this expression.

You may guarantee that the final representation is coherent and functionally related by precisely filling out these properties for each node as shown in Figures. To ensure that the final representation appropriately depicts the underlying architecture and data exchanges, each node develops into a well-defined and useful component of the system.


runtime-EclipseApplication - end2end_encryption.project.design/description/project.odesign - Eclipse IDE
File Edit Navigate Search Project Viewpoint Specification Editor Run Window Help
Package Explorer
project.odesign
Sirius Specification Editor
Properties
General
Import
Documentation
Behavior
Advanced

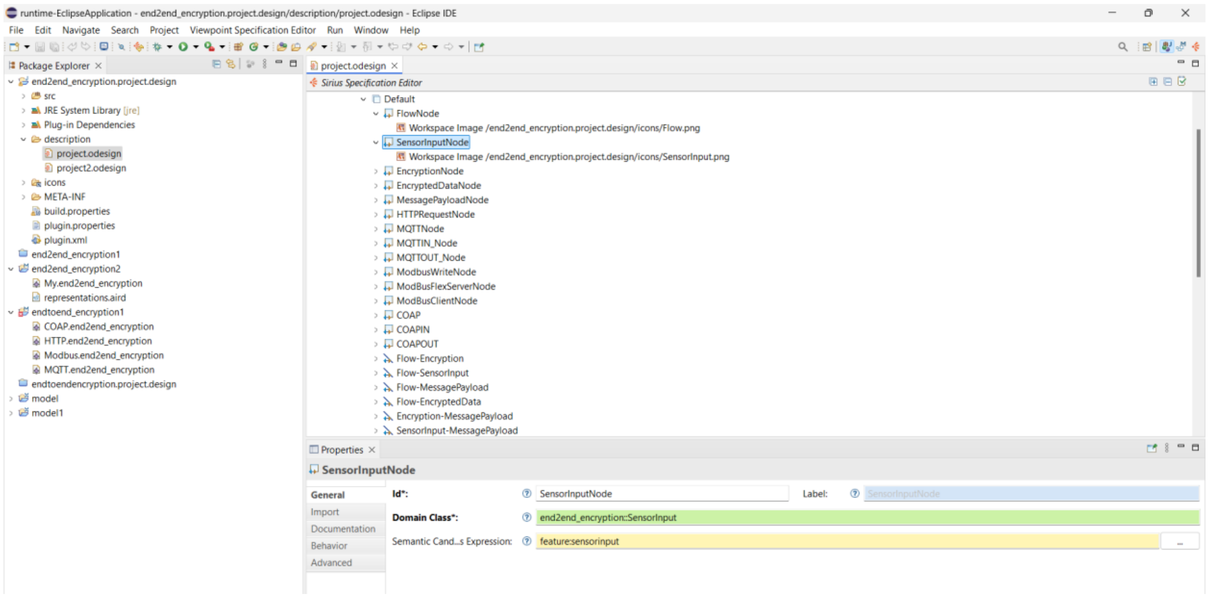
Package Explorer:
- end2end_encryption.project.design
  - src
  - JRE System Library [jre]
  - Plug-in Dependencies
  - description
    - project.odesign
    - project2.odesign
  - icons
  - META-INF
  - build.properties
  - plugin.properties
  - plugin.xml
- end2end_encryption1
- end2end_encryption2
  - My.end2end_encryption
  - representations.aird
- endtoend_encryption1
  - COAP.end2end_encryption
  - HTTP.end2end_encryption
  - Modbus.end2end_encryption
  - MQTT.end2end_encryption
- endtoendencryption.project.design
- model
- model1

Sirius Specification Editor:
- Default
  - FlowNode
    - Workspace Image /end2end_encryption.project.design/icons/Flow.png
  - SensorInputNode
  - EncryptionNode
  - EncryptedDataNode
  - MessagePayloadNode
  - HTTPRequestNode
  - MQTTNode
  - MQTTIN_Node
  - MQTTOUT_Node
  - ModbusWriteNode
  - ModBusFlexServerNode
  - ModBusClientNode
  - COAP
  - COAPIN
  - COAPOUT
  - Flow-Encryption
  - Flow-SensorInput
  - Flow-MessagePayload
  - Flow-EncryptedData
  - Encryption-MessagePayload
  - SensorInput-MessagePayload
  - MQTT-Encryption

Properties: FlowNode

Id*: FlowNode Label: FlowNode
Domain Class*: end2end_encryption::Flow
Semantic Cand...s Expression: feature:flow

Figure 17. Flow Node Sirus Architecture.


runtime-EclipseApplication - end2end_encryption.project.design/description/project.odesign - Eclipse IDE
File Edit Navigate Search Project Viewpoint Specification Editor Run Window Help
Package Explorer
project.odesign
Sirius Specification Editor
Properties
General
Import
Documentation
Behavior
Advanced

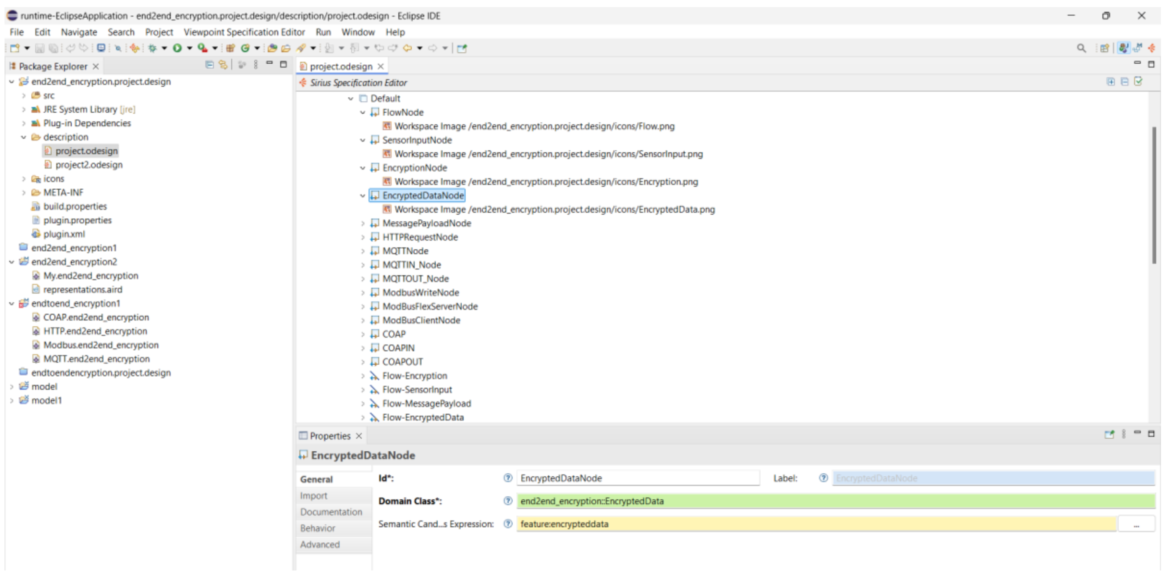
Package Explorer:
- end2end_encryption.project.design
  - src
  - JRE System Library [jre]
  - Plug-in Dependencies
  - description
    - project.odesign
    - project2.odesign
  - icons
  - META-INF
  - build.properties
  - plugin.properties
  - plugin.xml
- end2end_encryption1
- end2end_encryption2
  - My.end2end_encryption
  - representations.aird
- endtoend_encryption1
  - COAP.end2end_encryption
  - HTTP.end2end_encryption
  - Modbus.end2end_encryption
  - MQTT.end2end_encryption
- endtoendencryption.project.design
- model
- model1

Sirius Specification Editor:
- Default
  - FlowNode
    - Workspace Image /end2end_encryption.project.design/icons/Flow.png
  - SensorInputNode
    - Workspace Image /end2end_encryption.project.design/icons/SensorInput.png
  - EncryptionNode
  - EncryptedDataNode
  - MessagePayloadNode
  - HTTPRequestNode
  - MQTTNode
  - MQTTIN_Node
  - MQTTOUT_Node
  - ModbusWriteNode
  - ModBusFlexServerNode
  - ModBusClientNode
  - COAP
  - COAPIN
  - COAPOUT
  - Flow-Encryption
  - Flow-SensorInput
  - Flow-MessagePayload
  - Flow-EncryptedData
  - Encryption-MessagePayload
  - SensorInput-MessagePayload

Properties: SensorInputNode

Id*: SensorInputNode Label: SensorInputNode
Domain Class*: end2end_encryption::SensorInput
Semantic Cand...s Expression: feature:sensorinput

Figure 18. Sensor Input Node Sirus Architecture.


runtime-EclipseApplication - end2end_encryption.project.design/description/project.odesign - Eclipse IDE

File Edit Navigate Search Project Viewpoint Specification Editor Run Window Help
Package Explorer
project.odesign
Sirius Specification Editor
Properties

```
end2end_encryption.project.design
  src
  JRE System Library [jre]
  Plug-in Dependencies
  description
    project.odesign
    project2.odesign
  icons
  META-INF
  build.properties
  plugin.properties
  plugin.xml
end2end_encryption1
end2end_encryption2
  My.end2end_encryption
  representations.aird
endtoend_encryption1
  COAP.end2end_encryption
  HTTP.end2end_encryption
  Modbus.end2end_encryption
  MQTT.end2end_encryption
endtoendencryption.project.design
model
model1
```

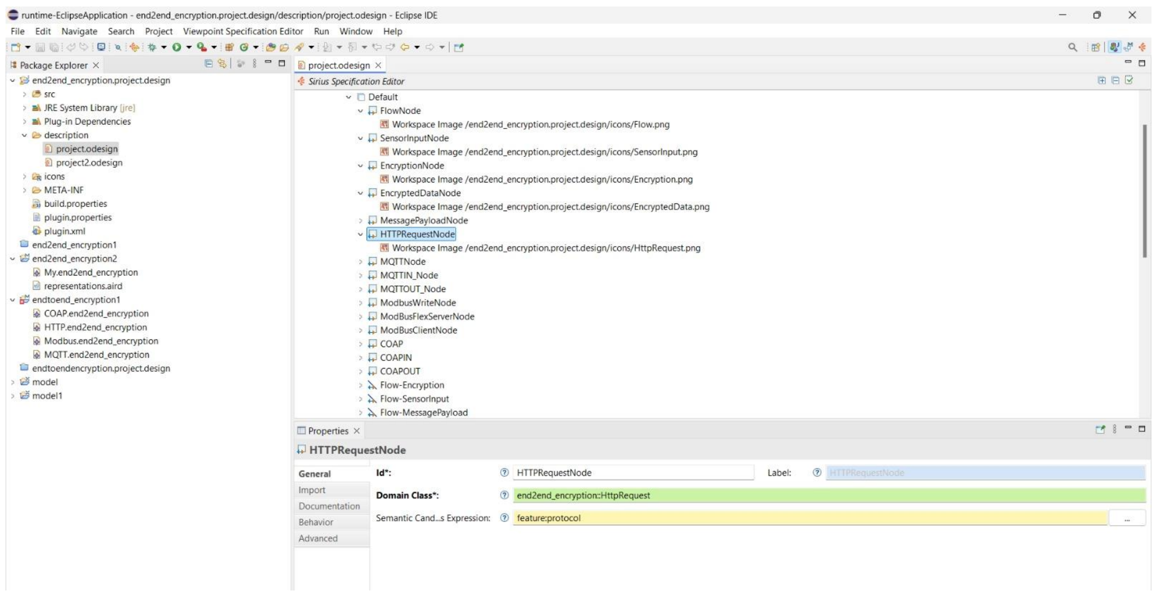
```
Default
  FlowNode
    Workspace Image /end2end_encryption.project.design/icons/Flow.png
  SensorInputNode
    Workspace Image /end2end_encryption.project.design/icons/SensorInput.png
  EncryptionNode
    Workspace Image /end2end_encryption.project.design/icons/Encryption.png
  EncryptedDataNode
    Workspace Image /end2end_encryption.project.design/icons/EncryptedData.png
  MessagePayloadNode
  HTTPRequestNode
  MQTTNode
  MQTTIN_Node
  MQTTOUT_Node
  ModbusWriteNode
  ModBusFlexServerNode
  ModBusClientNode
  COAP
  COAPIN
  COAPOUT
  Flow-Encryption
  Flow-SensorInput
  Flow-MessagePayload
  Flow-EncryptedData
```

EncryptedDataNode

General | Import | Documentation | Behavior | Advanced

Id*: EncryptedDataNode Label: EncryptedDataNode
Domain Class*: end2end_encryption::EncryptedData
Semantic Cand...s Expression: feature:encrypteddata

Figure 19. Encrypted Data Node in Sirius Architectures.


runtime-EclipseApplication - end2end_encryption.project.design/description/project.odesign - Eclipse IDE

File Edit Navigate Search Project Viewpoint Specification Editor Run Window Help
Package Explorer
project.odesign
Sirius Specification Editor
Properties

```
end2end_encryption.project.design
  src
  JRE System Library [jre]
  Plug-in Dependencies
  description
    project.odesign
    project2.odesign
  icons
  META-INF
  build.properties
  plugin.properties
  plugin.xml
end2end_encryption1
end2end_encryption2
  My.end2end_encryption
  representations.aird
endtoend_encryption1
  COAP.end2end_encryption
  HTTP.end2end_encryption
  Modbus.end2end_encryption
  MQTT.end2end_encryption
endtoendencryption.project.design
model
model1
```

```
Default
  FlowNode
    Workspace Image /end2end_encryption.project.design/icons/Flow.png
  SensorInputNode
    Workspace Image /end2end_encryption.project.design/icons/SensorInput.png
  EncryptionNode
    Workspace Image /end2end_encryption.project.design/icons/Encryption.png
  EncryptedDataNode
    Workspace Image /end2end_encryption.project.design/icons/EncryptedData.png
  MessagePayloadNode
  HTTPRequestNode
    Workspace Image /end2end_encryption.project.design/icons/HttpRequest.png
  MQTTNode
  MQTTIN_Node
  MQTTOUT_Node
  ModbusWriteNode
  ModBusFlexServerNode
  ModBusClientNode
  COAP
  COAPIN
  COAPOUT
  Flow-Encryption
  Flow-SensorInput
  Flow-MessagePayload
```

HTTPRequestNode

General | Import | Documentation | Behavior | Advanced

Id*: HTTPRequestNode Label: HTTPRequestNode
Domain Class*: end2end_encryption::HttpRequest
Semantic Cand...s Expression: feature:protocol

Figure 20. HTTP Request Node in Sirius Architectures.

The process of representation in Sirius entails developing unique visual representations of models within a customized modeling workbench. Designers create a user-friendly and engaging environment using perspectives, diagrams, node and edge mappings, visual styles, tool extensions, layouts, and expressive queries. Diagrams provide components of the model in accordance with predefined rules, whereas viewpoints specify certain perspectives or elements of the model. For users to build, edit, and visualize domain-specific models, Sirius's modeling environment offers a simple and effective platform. The Diagram Area, Palette, and Attribute Assignment Area work together to give users the ability to quickly instantiate ideas, define attributes, and create relationships, resulting in a thorough and visually engaging representation of the modeled system. A graphical representation of case study 1 is shown in Figure 16.

4.2. Case Study 2

Smart Voice Pathology Monitoring System (SVPMS)

Voice pitch and frequency is everyone's unique feature. However, the natural phenomena of generating the voice are sometimes disturbed when individuals continuously speak for longer periods or habitual to communicate in a louder voice. Musicians, teachers, and sometimes the kids who shout very often face some distortion in their voice, called voice pathology. Other than poor vocal practices, smoking, dehydration, and laryngeal infection can also deteriorate voice quality causing fatigue and strain in vocal cords. [53] Monitoring of voice pathology is an important factor that most patients ignore when they get a temporary relief after medication. Whereas this issue leads to severe health problems if not monitored.

We will be creating a Smart Voice Pathology Monitoring System (SVPMS) which would be monitoring the patient's health attributes to identify their current condition. This automated system would be handled through IoT. SVPMS tracks patients' health with sensors and sends encrypted packets over the application layer to notify paramedics. The process involves gathering the data, processing it, and finally analyzing it to evaluate a finalized decree so that patient gets on-time medical assistance. In case any patient doesn't need any help from medical staff, then transmission of false data could also be prohibited so that forged sensor values are not communicated.

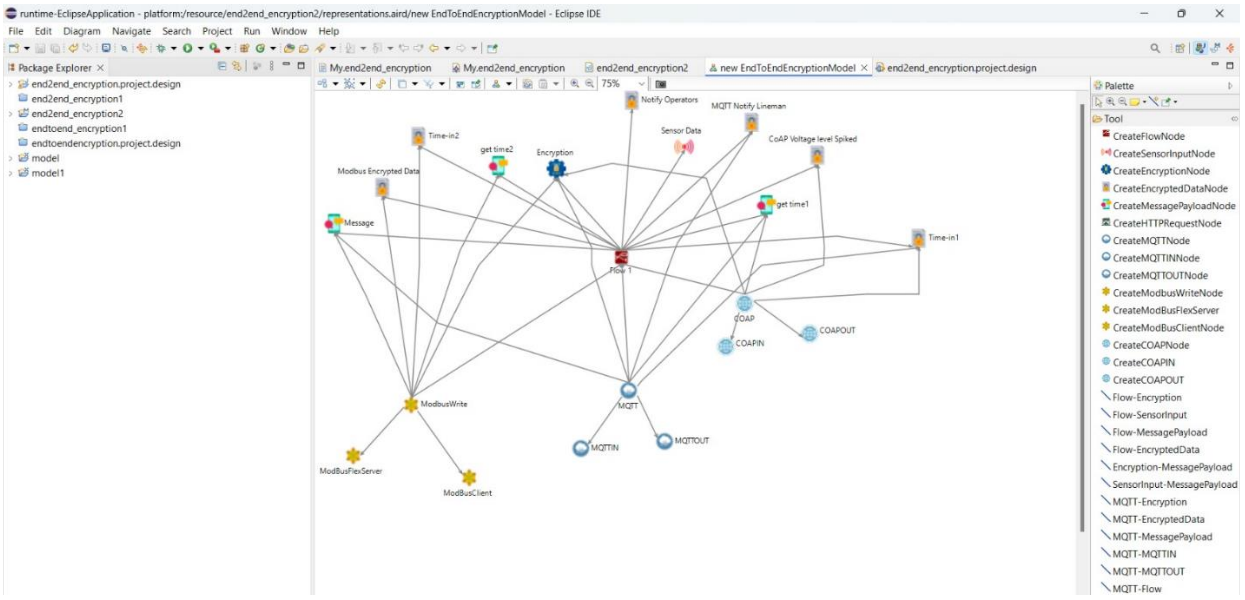


Figure 21. Graphical View of SVPMS

4.2.1. Acceleo Transformation

For a formal verification framework, we have developed a transformation engine. The Ecore Metamodel diagram may be transformed into timed automata using this engine, simplifying formal verification. With the help of this transformation, we may examine and verify the behavior of complex systems under precise timing constraints, assuring the accuracy and dependability of those systems. The architectural plan shown in Figure 27 is the foundation upon which the Transformation Engine has been implemented. This engine's major objective is to automate the whole verification and transformation process utilizing transformation mapping rules that preserve the semantic parity of the two languages, namely the M2 level metamodel and timed automata. We have selected the Acceleo tool, which serves as the foundation for carrying out these changes, for the model-to-text transition [59]. The User Interface and the Model Generator are the two main components of the Transformation Engine. Together, these elements make model changes quick and smooth while maintaining semantic parity between the original metamodel and the timed automata representation.

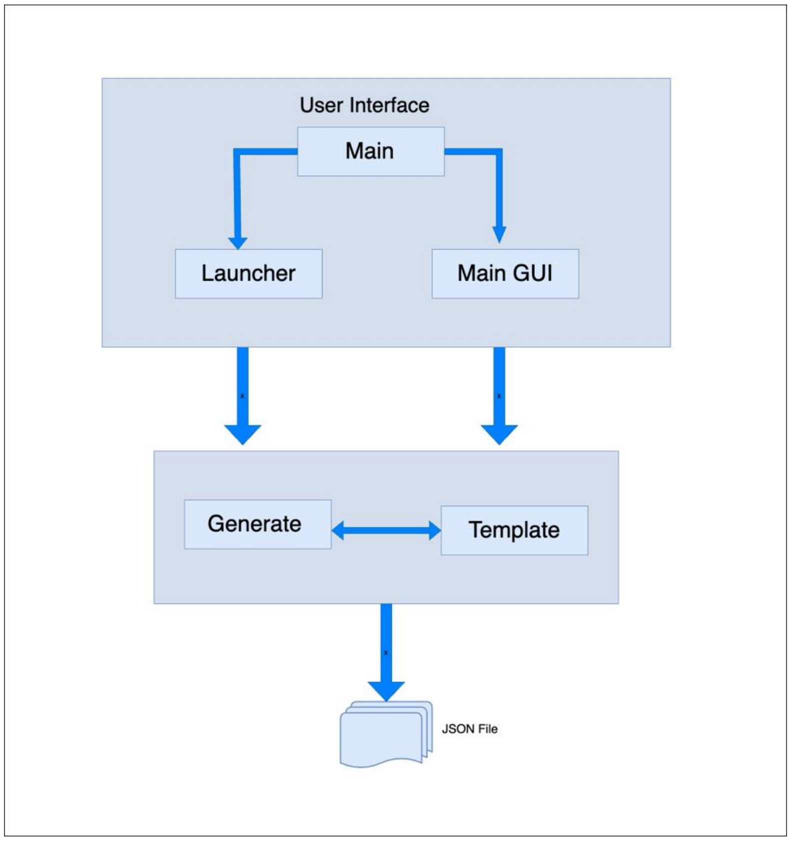


Figure 22. The architecture of the Transformation engine

The Ecore metamodel is transformed by the Model Generator into Json for validation and verification. There are two main components to the process:

Generate File (Generate.java): The logic for the Model-to-Text transformation is handled in this file, which also creates the JSON representation for the specified Ecore model from the user interface.

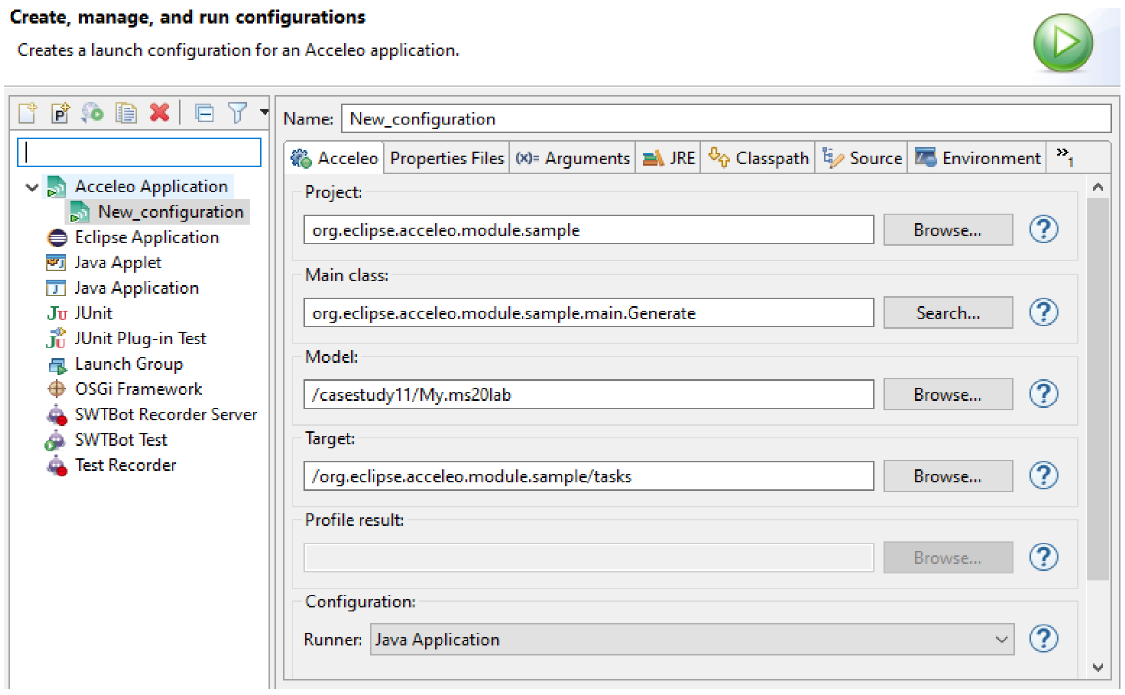
Template File (Generate): The transformation's coding logic is contained in the template file. It gives the rules for constructing the target model and describes how the Ecore elements are transferred to JSON constructs. Two output files are produced after the Generate.java and related template files have been successfully executed:

a. JSON This file contains the modified Encryption model, which simulates the system behavior shown in the core model.

b. User requirements properties based on the score model is contained in this file. These properties describe the system's desired properties and behaviors that need to be tested.

A model of the End-to-End Encryption was used to create Acceleo code, which then generates code. The Acceleo language, a template-based code generator that employs the Eclipse Modelling Framework (EMF) to convert models into executable code, was used to create the Acceleo code [60]. To provide flexibility and automation in the software development process. Acceleo makes it possible to generate code, documentation, or any other textual artifacts from models.

Once the code has been written in Acceleo, it is time to begin the run configurations by choosing the model and case study, customizing it, and then executing to produce the output as shown in Figure 28.


Create, manage, and run configurations

Creates a launch configuration for an Acceleo application.

Name: New_configuration

Project: org.eclipse.acceleo.module.sample

Main class: org.eclipse.acceleo.module.sample.main.Generate

Model: /casestudy11/My.ms20lab

Target: /org.eclipse.acceleo.module.sample/tasks

Profile result:

Configuration:

Runner: Java Application

Figure 23. Run configuration of Acceleo

CHAPTER 5

VALIDATION

In this validation chapter, Node-Red will be used to thoroughly validate each case study. We will make extensive evaluations of temporal dynamics, synchronization, and property adherence using JSON files from Acceleo transformations

5.1. NODE-RED

It is an open-source programming tool that is explicitly meant to link hardware devices, APIs, and online services in new and intriguing aspects. It is based on Node.js leveraging non-blocking, event-driven models which are best for developing scalable network applications [54] [55]. Node-Red's visual programming interface enables easy production of real-time, data-driven Internet of Things (IoT) workflows in which devices and services have to be connected without any hitches. Node-Red makes wiring straightforward through an interface where any node available can be connected with another by just using the mouse. In addition, developers can use it for IoT projects without much coding required thus saving time while improving efficiency. With it single-click deployment feature; people do not have to spend long hours typing codes before getting started hence making it possible for users who do not know programming languages at all to grasp how things work within a few minutes [56]. Node-Red was chosen for the following various reasons:

- **Event-driven Architecture:** Node-Red is built using Node.js, Node-Red can handle non-blocking asynchronous events, common in IoT environments [57].
- **Ease of Use:** Node. red incorporates graphics that make programming easier by visual means. It should break the flow of creation and editing through cycles into small pieces so that one may be able to implement complicated IoT apps more easily (and debug them too) when necessary.
- **Extensibility:** In addition, Node-Red comes with several extensions that can be used on any project without any constraints because they support many devices as well as different types of connection methods including internet services (like Twitter), sensors, etc.
- **Debugging Functionalities**: Node-Red includes robust debugging tools, which allow developers to see results at different levels and quickly identify and fix issues within the flows. In this respect, it is important to ensure IoT systems have a reliable and secure communication
- **Continuous Monitoring:** The Command Line Interface (CMD) keeps running in the background; therefore, Node-Red continuously monitors and performs flows, tracks errors, and assures their smooth operation. [58]

5.2. Proof of Concept

5.2.1. Case Study 1. Grid Station

Critical systems like grid stations in smart cities pose a lot of cybersecurity threats thus we made this a part of this research as a proof of concept. To ensure a smooth execution of a program it is crucial to ensure the safety of threads [59]. Power is supplied in homes, residential zones, commercial sectors, and industrial zones from the power grid. The collection of real-time data from all the components of the grid stations enables the smooth continuity of the operations and the generous amount of time it takes. To capture the voltage level and other things with much more efficiency and ensure economic stability and public safety from cyber-attacks. They are installed to measure the stability of metrics such as phase, frequency, and power to check the status of the station. Potential issues such as congestion systems and other things are also identified using the automated IoT [60]. The data collected from the grid stations are then transferred to control centers. To identify anomalies this data is then monitored by a centralized system and initiates the respective response that is used to make sure that the system runs smoothly. To make sure that the data that is collected from the servers reaches other servers with more authenticity. This involves conventional steps like gathering data, analyzing the data, and then processing it to evaluate whether it requires any further assistance or not.

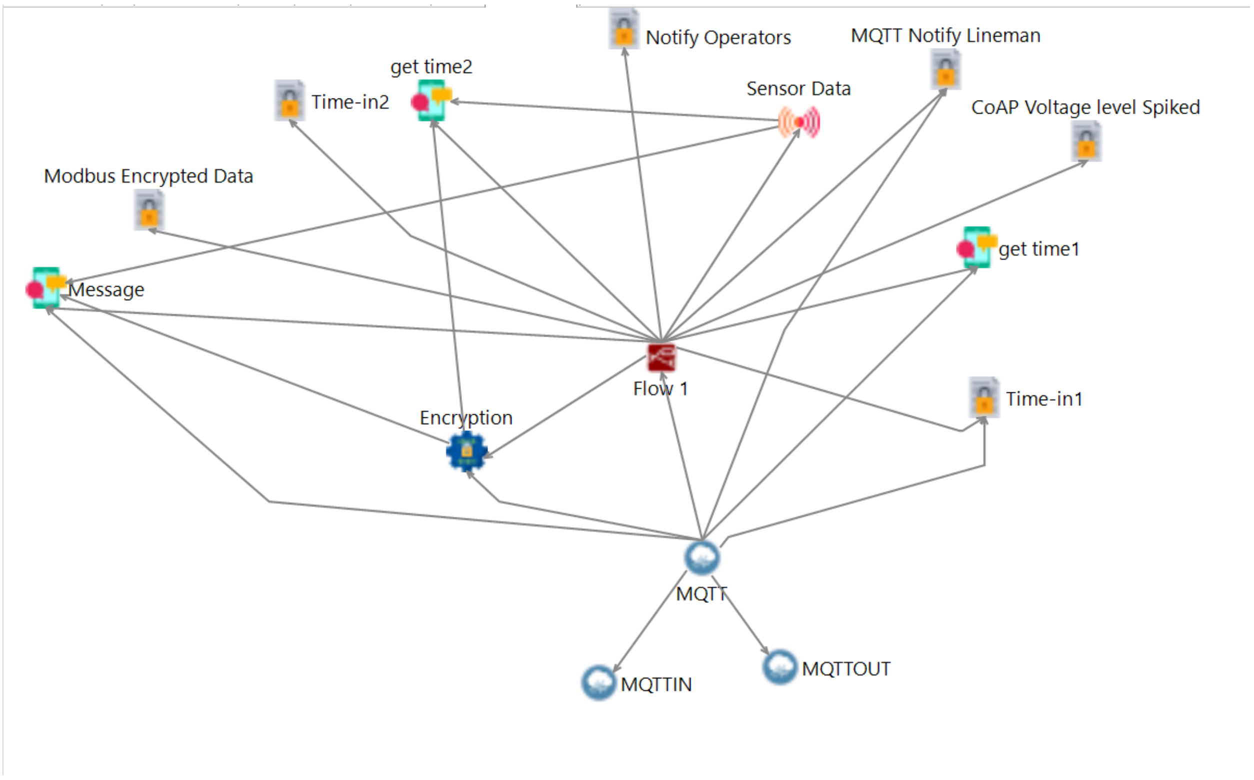


Figure 24. Grid Station Case Study with Sirius

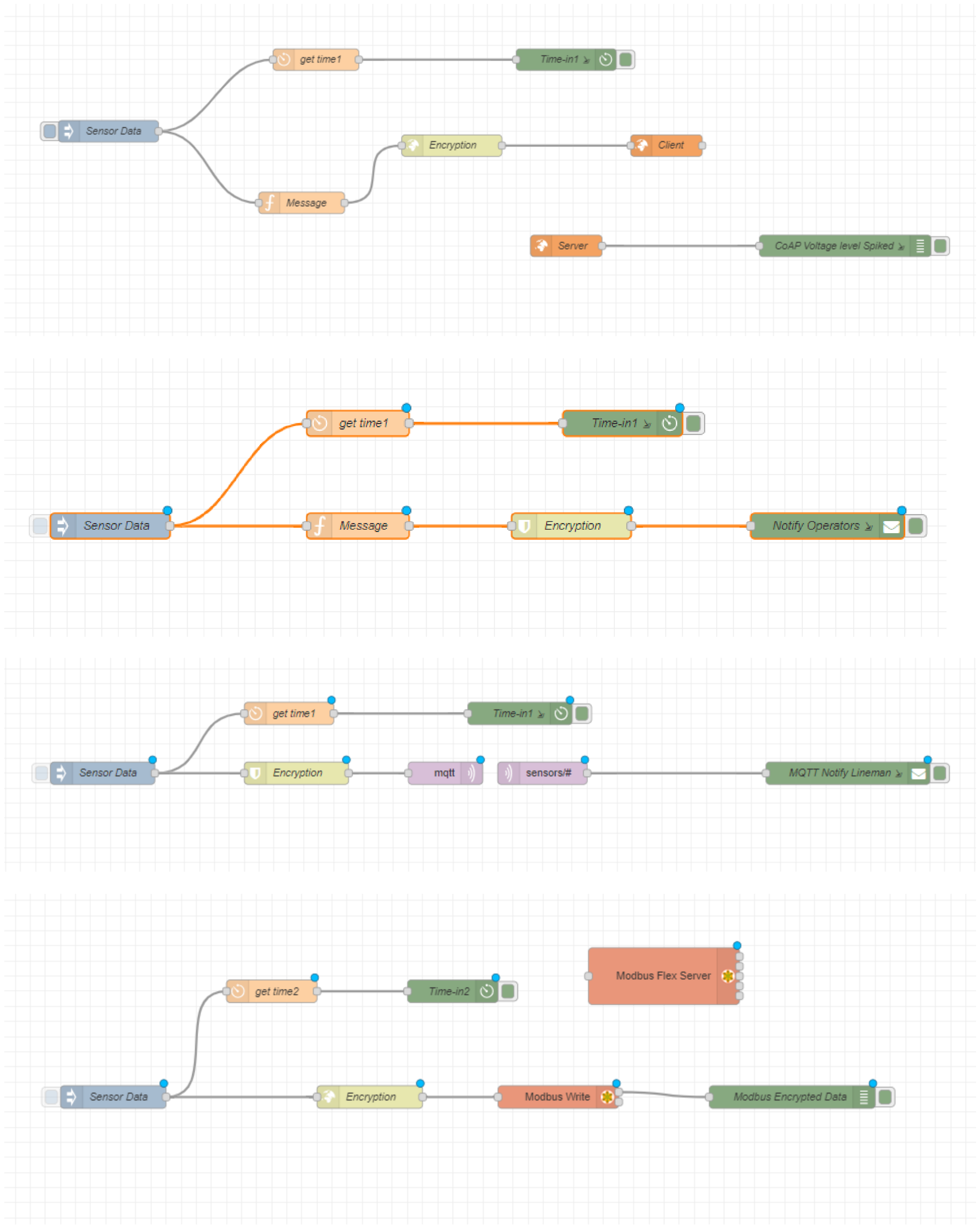


Figure 25. Simulating Grid Station Case Study in Node-Red

AES encryption is symmetric encryption and a well-known method that is used to encrypt sensitive data [61]. This process is symmetric which means both the sender should have the same key to encrypt or decrypt the data. A block cipher is defined as a method where the large chunks are broken into smaller parts where the plaintext is converted into an incomprehensible form which is a cipher text [62]. Multiple cryptographic methods are used and undergo multiple rounds for the encryption of data and ultimately provide confidentiality and integrity. The AES method protects it from brute force attacks. The security level is either achieved by a 128-bit key, 192 or 256. They undergo 10, 12, and 14 rounds of encryption. There are many methods involved such as substitution, transposition, and mixing of cipher text and plain text to obtain the final cipher output. In this process, it passes through multiple rounds, and it is split into block ciphers and has 3 main components which are as follows [63].

- **Input block:** It is the text that is received from the sender.
- **Key:** It converts the plain text into cipher text and protects the plain text and important information the user has sent.
- **Output block:** It is the output block that is generated in response to plain text after passing through some method that processes with keys and gives cipher text.

In this, a plain text is put into an array and then multiple techniques are applied to it. After putting it into an array, it is passed through cipher transformation and repeated multiple encryption rounds. Through this, a cipher is obtained which is predefined. Then it is put into a second transformation, where all rows are shifted except for the first one. In the third transformation, it passes through the Hill cipher. Hill cipher is performed on each column generating a mature cipher text. During decryption, they use multiple and copy cipher, and the layers of encryption keys are removed generating a plaintext [62].

5.2.2. Case Study 2. Smart Voice Pathology

Our case study is based on voice pathology. Voice pitch and frequency are everyone's unique features. However, the natural phenomena of generating the voice are sometimes disturbed when individuals continuously speak for longer periods or are habitual to communicate in a louder voice. Musicians, teachers, and sometimes the kids who shout very often face some distortion in their voice, called voice pathology. Other than poor vocal practices, smoking, dehydration, and laryngeal infection can also deteriorate voice quality causing fatigue and strain in vocal cords. Monitoring of voice pathology is an important factor that most patients ignore when they get temporary relief after medication. This issue leads to severe health problems if not monitored.

Smart Voice Pathology Monitoring System (SVPMS) monitors the patient's health attributes to identify their current condition. This automated system would be handled through IoT. SVPMS tracks patients' health with sensors and sends encrypted packets over the application layer to notify paramedics. The process involves gathering the data, processing it, and finally analyzing it to evaluate a finalized decree so that the patient gets on-time medical assistance. In case any patient doesn't need any help from medical staff, then transmission of false data could also be prohibited so that forged sensor values are not communicated to relevant paramedics. We have simulated the workflow in the Sirius dashboard and analyzed how packets are sent over different protocols. We have evaluated the performance of encrypted data packets over application layer protocols using their size and transmission time. Before sending packets over the network, they are encrypted using a cloud-based encryption algorithm to ensure their safety on the way from sender to receiver.

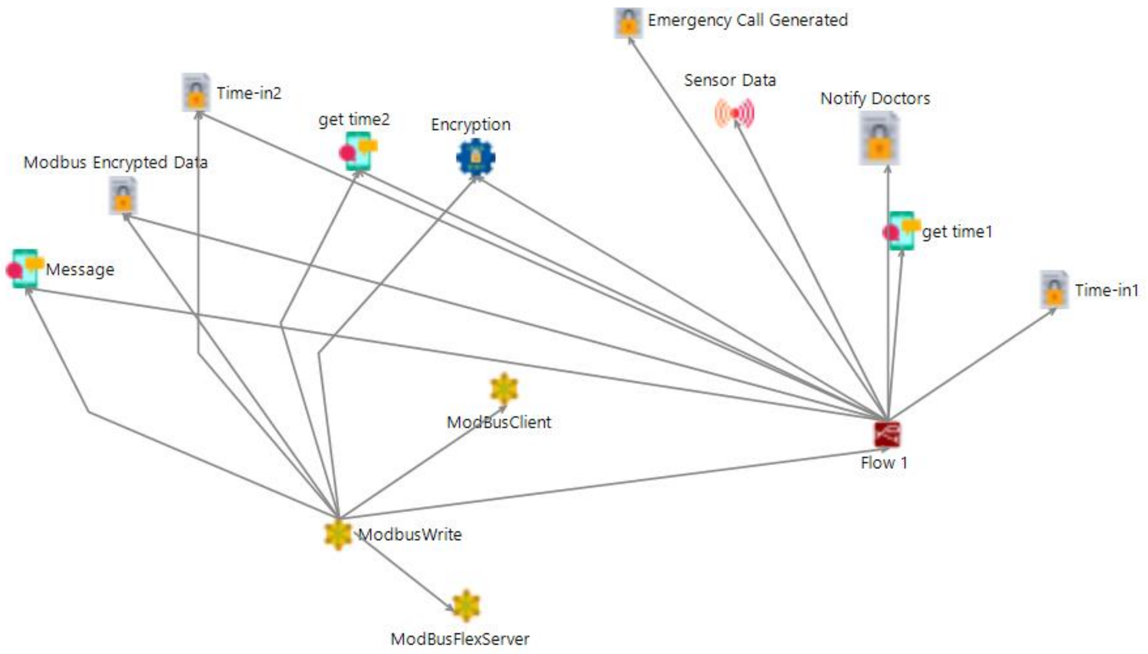


Figure 26. Simulating Case Study SVPMS

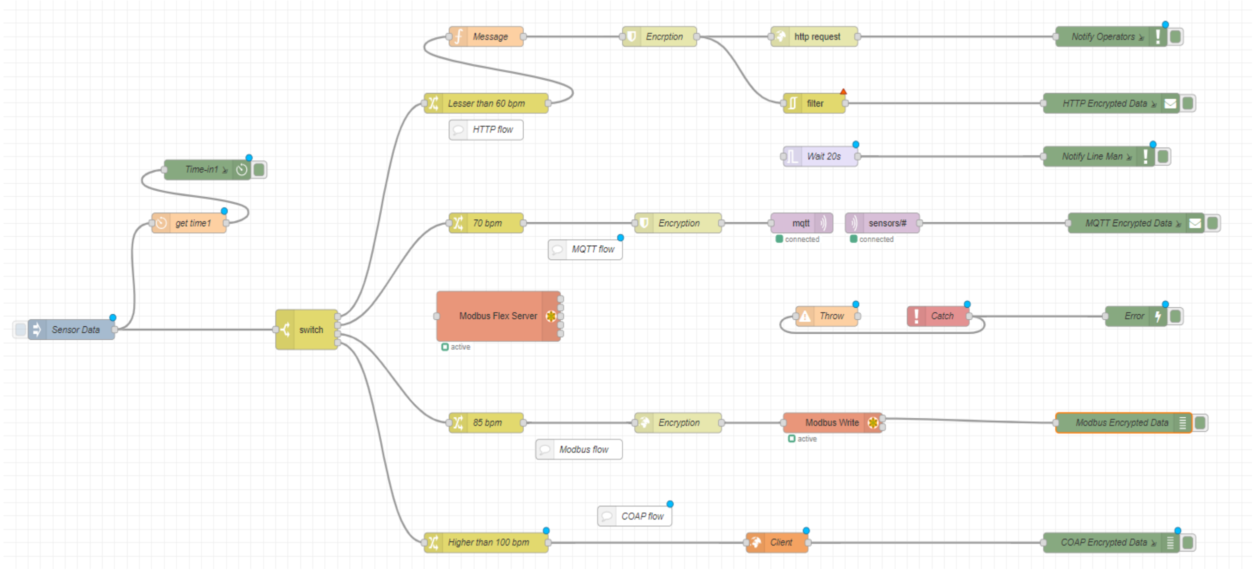


Figure 27. Validation in Node-Red

Summary

As proof of the proposed concept, we have made a system, which changes according to different circumstances, to keep off cyber-attacks on the electrical power grid. Every piece of information collected from the servers remains undistorted owing to our tough encoding formula.

Data Collection:

The grid has various parameters tracked by sensors among them, are voltage levels, and temperature of essential components e.g. circuit breakers; from power lines as well as transformers; the other parameters are phase, frequency, and power factors. Real-time power data is collected from Internet of Things (IoT) sensors and these continuously collect health and status data for the power grid components.

Data Encryption:

We encrypt the collected data so that no one else can ever see what it holds unless they have been given permission. Advanced Encryption Standard (AES) is employed when it comes to encrypting information. AES is an example of a symmetric cryptosystem that turns plain text messages into scrambled text by employing a secret key. The key length options vary from 128 bits to 256-bit ones, this provides flexibility when determining which strengths may be needed for different projects and scenarios, and in some cases where there are restrictions due to other technical or policy reasons, 192 bits may also work well after all [63].

Data Transmission:

The central control center receives encrypted data using secure channels. To make certain that the data is sent securely on the Web, secure communication protocols like HTTPS are used [64] [65].. The secure APIs can handle many communication issues making sure that there is a common interface for all data exchanges.

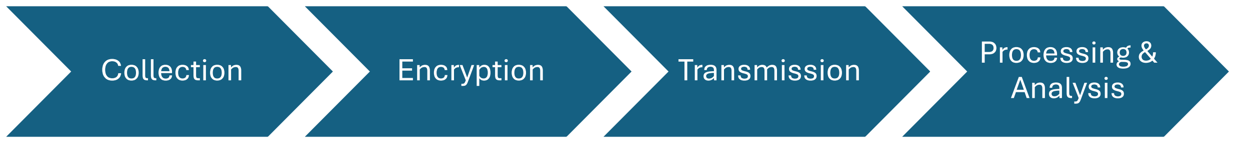


Figure 28. Flowchart of data processing through Node-Red.

Data Processing:

We use Node-Red to control what data is moving and to easily spot mistakes. There are algorithms integrated directly into Node-Red to handle the decryption part. Tools for analysis plus models that rely on machine learning can also assist us in seeing trends or irregularities within the information thereby giving us a hint on how the grid behaves practically speaking.

5.3. The Significance of the Modeling Approach

Modern software engineering critically depends on Model-Driven Development (MDD), particularly in the face of IoT's highly intricate nature and security challenges [35]. With modeling methodologies like ours, the aim is to address encryption and data reliability issues in a structured manner. The benefits of the Modelling approach used for this encryption process are mentioned below [66]:

5.3.1. Complexity Reduction and Enhanced Abstraction:

The complexity level of trying to control interconnected connected systems when it comes to standard program creation methods is generally higher. But if we use MDD we will have to think less about code structure and more about how everything works together at a high level in our applications [66]. By involving Node-red as a tool, a visual programming interface was possible which helped in the process of creating, managing, and debugging the workflows in IoT become easier and more fluent. The cognitive load on developers and the risk of errors is also minimized [54]. With its drag-and-drop feature, Node-Red allows people to develop IoT apps faster because they do not require a lot of coding skills. Those who know little about programming will also find it easy to use. Node-Red is modular meaning that it enables new components or superior functionalities to be integrated into it and makes sure that the system can be in tune with changing needs without a major rewrite [50].

5.3.2. Time Efficiency and Agile Development

Quick prototyping and real-time iteration were possible due to involving MDD in the methodology of the thesis, these characteristics of MDD are proven to be crucial IoT environments for quick adaptation based on testing and feedback [31]. Node-red's one-click change deployment capability massively advances the development process through its characteristic ability to speed up update frequency as well as improvements [50].

Node-Red provided strong debugging tools to troubleshoot and resolve issues at every phase of development which reduces downtime and increases system reliability. This means anomalies and errors are detected early enough and fixed immediately hence the system operates smoothly and remains stable through continuous monitoring via the Command Line Interface (CMD).

5.3.3. Enhanced Security and Data Integrity

To ensure security at all times, we have incorporated Advanced Encryption Standard (AES) in the design. This is crucial in securing data during storage and its transmission [63]. Cyber-attacks could lead to the loss of sensitive information which requires protection. In the methodology we use, data is safeguarded along its life cycle using end-to-end encryption forms while in transit or when stored. Such systems as electrical grids which ought to have integrity of information are very critical.

It is very important to keep alive the reliability of IoT systems by maintaining their data integrity. The latter act is done through strong encryption techniques and secure communication channels that determine the truthfulness of data reaching the server for monitoring. When it comes to Node-Red, the application of artificial intelligence welcomes high-scale detection of error margins using specific devices that indicate nonconformism [57]. Consequently, it enhances trust and dependability of the overall function and safety of interconnected devices in the Internet of Things.

When IoT networks expand, it is no longer possible to manage security manually. Encipherment was systemically implemented using modelling approaches that make security uniform and thus consistent Interoperability, this process involved the creation of interfaces to allow communication between diverse cryptographic protocols and standards hence systems from one manufacturer can interact with devices produced by others within the same network

The Model developed helps to ensure that the information collected from IoT sensors remains correct and reliable, which is crucial in making decisions based on real-time data. Inbuilt automated divergence processes and validation in modeling tools aid in consistently checking data credibility to reduce the chances of data corruption or manipulation.

CHAPTER 6

DISCUSSION

In this thesis, we have devised a methodology to deal with modern IoT system problems that are complex and lack security features, especially in important infrastructures. This research combines MDD along with enhanced methods of protecting to make strong, efficient frameworks that can control these networks. The thesis contributed significantly by applying MDD to simplify and improve the development of IoT applications. This involves using Eclipse and making the Sirius tool representation to take the method up a notch in terms of abstraction and visualization making it easier for programmers to put up testing environments and integration configuration files as well as write complex systems. By reducing the cognitive load experienced by software engineers or developers, we can hasten the rate at which software products are developed. It is dynamic and therefore suited for IoT environments owing to their continuously changing needs and configuration requirements.

The thesis integrates encryption standards and secure communication protocols to address critical data integrity concerns. Within IoT networks, it is important to guarantee the reliability and security of data since breaches can cause great disruptions which might make devices susceptible to different forms of attacks. Sensitive data will continue to be protected from unauthorized access and cyber-attacks by using cloud-based encryption practices. Especially in critical systems where human and material resources are at stake, this is quite important because security breaches can have widespread effects and are grave.

We demonstrated the practical utility of the suggested approach in a comprehensive examination of grid stations in smart cities. It is shown that if actual-time information is collected through IOT sensors and needs to be transmitted in a network where there exists a doubt of anomalies, this framework is an appropriate solution to enhance the robustness while keeping the limited resources in consideration. The proposed model is adaptable and thus makes a key contribution to the domain of IoT security and system management. Utilizing MDD and cutting-edge security techniques, it presents a scalable, effective, and secure way of handling intricate IoT networks, guaranteeing their dependability as well as robustness in crucial applications.

CHAPTER 7

CONCLUSION

This thesis proposed the idea that aims at dealing with the problem of encrypting data in IoT systems using Model Driven Approach. The basis of this lies in combining MDD (Model-Driven Development) with advanced security techniques which leads to the creation of a strong, efficient, and secure framework for managing these networks. Traditional software development models like the waterfall model are inadequate for the dynamic and interconnected nature of modern systems, the thesis addresses the issue by introducing MDD as a solution that eases the design, implementation, and testing of complex systems through higher levels of abstraction and visualization. Eclipse came in handy for its capability to simplify development through visual programming and event-driven architecture are highlighted in the methodology of this thesis. The central premise of this thesis is to boost end-to-end encryption of communication in the IoTs. By relying on robust protocols, the leak of crucial data in IoT systems is minimized since the methodology focuses on maintaining their security against intruders and cybercrimes. This is especially important in ensuring consistency and confidence in the functioning of important facilities like power grids.

A case study on smart city grid stations is carried out to examine the practical application of the suggested methodology. The purpose of deploying IoT sensors is to collect real-time data concerning various parameters that are afterward analyzed by Node-Red to ascertain their stability and effectiveness. Additionally, integrating machine learning algorithms for detecting anomalies enhances the system's ability to preemptively identify any threats that may arise. In managing complex IoT networks, by using MDD together with advanced security approaches, this research work offers a scalable, efficient, and secure solution that assures the dependability and resilience of such networks in important applications.

To prevent an increase in the size of data, encoding algorithms should be improved and made secure as one possible direction for research in the future. It is further suggested that exploring compression methods combined with encoding may reduce the problem of data expansion. Therefore, simple forms of encryption protocols should be designed especially for IoT-based environments so that they do not consume too much system power. To further strengthen the model's ability to detect and respond to threats, improving the anomaly detection mechanisms with the application of some advanced machine learning techniques. Such practical insights would in turn help researchers in doing further modifications and advancements on how they manage security systems of devices used in IoT.

References

[1] K. A, P. Prombage and . L. M, "Introduction to iot," 2020.

[2] M. Liyanage, A. Braeken, P. Kumar and M. Ylianttila, "IoT security: Advances in authentication," John Wiley \& Sons, 2020.

[3] G. Solomon, P. Zhang, R. Brooks, and Y. Liu, "A Secure and Cost-Efficient Blockchain Facilitated IoT Software Update Framework," *IEEE Access,* 2023.

[4] A. Dorri, S. S. Kanhere and R. Jurdak, "Towards an optimized blockchain for IoT," in *Proceedings of the second international conference on Internet-of-Things design and implementation*, 2017, pp. 173--178.

[5] A. Singh and R. Gilhotra, "Data security using private key encryption system based on arithmetic coding," *International Journal of Network Security \& Its Applications (IJNSA),* vol. 3, no. 3, pp. 58-67, 2011.

[6] B. R. T. V. and M. V., "Internet of Things - A study on the security," 2015.

[7] A. W. Atamli and A. Martin, "Threat-based security analysis for the Internet of Things.," International Workshops on Secure Things 2014, 2014.

[8] L. S.-Y. C. K.-C. and L. Y., "Toward ubiquitous massive accesses in 3GPP machine-to-machine communication," IEEE Commun Mag 2011, 2011.

[9] M.-S. Pan and Y.-C. Tseng, "ZigBee and their applications," in *Sensor networks and configuration: Fundamentals, standards, platforms, and applications*, 2007, pp. 349--368.

[10] D. R. S. e. al, "Progress Report on the National Geologic Map Database, Phase 3: An Online Database of Map Information," in *Digital Mapping Techniques '01 -- Workshop Proceedings U.S. Geological Survey Open-File Report 01-223.*, 2001.

[11] L. Barreto and A. Amaral, "Smart farming: Cyber Security challenges," in *2018 International Conference on Intelligent Systems*, 2018.

[12] S. Bagchi, T. F. Abdelzaher, R. Govindan, P. Shenoy, A. Atrey, P. Ghosh and R. Xu, "New Frontiers in IoT: Networking, Systems, Reliability, and Security Challenges," *IEEE Internet of Things Journal,* vol. 7, no. 12, pp. 11330--11346, 2020.

[13] "International Organization for Standardization / International Electrotechnical Commission," in *Software Engineering - Metamodel for Development Methodologies.*, 2007.

[14] Kokoris-Kogias E, Voutyras O, and Varvarigou T, "TRMSIoT: A scalable hybrid trust & reputation model for the social Internet of things.," in *2016 IEEE 21st international conference, on emerging technologies and factory automation ETFA*, 2016.

[15] S. Sicari, A. Rizzardi, L. A. Grieco, and A. Coen-Porisini, "Security, privacy, and trust in internet of things," *Computer networks,* vol. 76, pp. 146-164, 2015.

[16] R. Tahir, A. Raza, F. Zaffar, F. U. Ghani and M. Zulfiqar, "Using SGX-Based Virtual Clones for IoT Security," in *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, 2018.

[17] A. Zanella, N. Bui, A. Castellani, L. Vangelista and M. Zorzi, "Internet of Things for Smart Cities," *IEEE Internet of Things journal,* vol. 1, no. 1, pp. 22--32, 2014.

[18] T. Denning, T. Kohno and H. M. Levy, "Computer security and the modern homes," *Communications of the ACM,* vol. 56, no. 1, pp. 94-103, 2014.

[19] S. Sontowski, M. Gupta, S. S. L. Chukkapalli, M. Abdelsalam, S. Mittal, A. Joshi and R. Sandhu, "Cyber Attacks on Smart Farming Infrastructure," in *IEEE 6th Internation Conference on Collaboration and International Computing*, 2020.

[20] O. S. Cupp, D. E. Walker and J. Hillison, "Agroterrorism inthe us: key security challenge for the 21st century.," *Biosecurity and bioterrorism: biodefense strategy, practice, and science,* vol. 2, no. 2, p. 97, 2004-105.

[21] M. B. Line, A. Zand, G. Stringhini and R. Kemmerer, "Targeted attacks against industrial control systems: Is the power industry prepared," in *Proceedings of the ACM Conference on Computer and CommunicationSecurity 2014*, 2014.

[22] BBC News, "Hack attack causes 'massive damage' at steel works," BBC, 22 December 2014. [Online]. Available: https://www.bbc.com/news/technology-30575104. [Accessed 1 6 2024].

[23] D. Barrera, L. Chuat, A. Perrig, R. M. Reischuk and P. Szalachowski, "The scion internet architecture," *Communications of the ACM,* vol. 60, no. 06, pp. 56-65, 2017.

[24] A. Perrig, "Global Communication Guarantees in the Presence of Adversaries," in *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, October 2020.

[25] K. Tange, M. D. Donno, X. Fafoutis and N. Dragoni, "A Systematic Survey of Industrial Internet of Things Security: Requirements and Fog Computing Opportunities," *IEEE Communications Surveys & Tutorials,* vol. 22, no. 4, pp. 2489 - 2520, 2020.

[26] R. McMillan, "Definition: threat intelligence," 16th May 2013. [Online]. Available: https://www.gartner.com/en/documents/2487216#:~:text=Threat%20intelligence%20is%20evidence-based%20knowledge%2C%20including%20context%2C%20mechanisms%2C,the%20subject%27s%20response%20to%20that%20menace%20or%20hazard.. [Accessed 29 April 2024].

[27] X. Zhang, H.-C. Hsiao, G. Hasker, H. Chan, A. Perrig and D. G. Andersen, "SCION: Scalability, Control, and Isolation On Next-Generation Networks," in *2011 IEEE Symposium on Security and Privacy*, 2011.

[28] R. Schlegel, S. Obermeier and J. Schneider, "Structured system threat modeling and mitigation analysis for industrial automation systems," in *IEEE 13th International Conference on Industrial Informatics (INDIN)*, 2015.

[29] A. Shostack, Threat modeling: Designing for security., John Wiley & Sons, 2014.

[30] P. Patel and D. Cassou, "Enabling high-level application development for the Internet of Things," *Journal of Systems and Software,* vol. 103, pp. 62--84, 2015.

[31] F. Ciccozzi, I. Crnkovic, D. Di Ruscio, I. Malavolta, P. Pelliccione and R. Spalazzese, "Model-driven engineering for mission-critical iot systems," *IEEE software,* vol. 34, no. 1, pp. 46--53, 2017.

[32] R. G. Anvekar and R. M. Banakar, "IoT application development: Home security system," in *2017 IEEE Technological Innovations in ICT for Agriculture and Rural Development (TIAR)*, Chennai, India, 2017.

[33] F. Ciccozzi and R. Spalazzese, "Mde4iot: supporting the internet of things with model-driven engineering," in *International Symposium on Intelligent and Distributed Computing}*, Springer, 2016.

[34] Z. M. Korani, A. M. A. A. R. d. Silva and J. C. Ferreira, "Model-Driven Engineering Techniques and Tools for Machine Learning-Enabled IoT Applications: A Scoping Review," *Sensors,* vol. 23, no. 3, 2023.

[35] M. Brumbulli and E. Gaudin, "Towards model-driven simulation of the internet of things," in *Complex Systems Design \& Management Asia: Smart Nations--Sustaining and Designing: Proceedings of the Second Asia-Pacific Conference on Complex Systems Design \& Management, CSD\&M Asia 2016*, Springer, 2016, pp. 17--29.

[36] B. B. Gupta and M. Quamara, "An overview of Internet of Things (IoT): Architectural aspects, challenges, and protocols," *Concurrency and Computation: Practice and Experience,* vol. 32, no. 21, p. e4946, 2020.

[37] S. Khan, S. Parkinson and Y. Qin, "Fog computing security: a review of current applications and security solutions," *Journal of Cloud Computing,* vol. 6, pp. 1--22, 2017.

[38] J. Yuan and X. Li, "A reliable and lightweight trust computing mechanism for IoT edge devices based on multi-source feedback information fusion," *Ieee Access,* vol. 6, pp. 23626--23638, 2018.

[39] B. Cooksey, An introduction to APIs, Zapier, Inc, 2014.

[40] A. Alawadhi, A. Almogahed and E. Azrag, "Towards Edge Computing for 6G Internet of Everything: Challenges and Opportunities," in *2023 1st International Conference on Advanced Innovations in Smart Cities (ICAISC)*, 2023.

[41] D. Ferraris, C. Fernandez-Gago and J. Lopez, "A model-driven approach to ensure trust in the IoT," *Human-centric Computing and Information Sciences,* vol. 10, pp. 1--33, 2020.

[42] G. Fortino, C. Savaglio, G. Spezzano and M. Zhou, "Internet of things as system of systems: A review of methodologies, frameworks, platforms, and tools," *IEEE Transactions on Systems, Man, and Cybernetics: Systems,* vol. 51, no. 1, pp. 223--236, 2020.

[43] G. Kumar and P. K. Bhatia, "Comparative analysis of software engineering models from traditional to modern methodologies," in *2014 Fourth International Conference on Advanced Computing \& Communication Technologies*, IEEE, 2014, pp. 189--196.

[44] M. Awad, "A comparison between agile and traditional software development methodologies," *University of Western Australia,* vol. 30, pp. 1--69, 2005.

[45] G. Papadopoulos, "Moving from traditional to agile software development methodologies also on large, distributed projects," *Procedia-Social and Behavioral Sciences,* vol. 175, pp. 455--463, 2015.

[46] S. Najihi, S. Elhadi, R. A. Abdelouahid and A. Marzak, "Software Testing from an Agile and Traditional view," *Procedia Computer Science,* vol. 203, pp. 775--782, 2022.

[47] S. Adams, P. A. Beling, S. Greenspan, M. Velez-Rojas and S. Mankovski, "Model-based trust assessment for internet of things networks," in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE*, 2018.

[48] K. M. Abbasi, T. A. Khan and I. ul Haq, "Modeling-framework for model-based software engineering of complex Internet of things systems," *Mathematical Biosciences and Engineering,* vol. 18, no. 6, pp. 9312--9335, 2021.

[49] M. Bisma, F. Azam, Y. Rasheed and M. W. Anwar, "A model-driven framework for ensuring role based access control in IoT devices," in *Proceedings of the 2020 6th International Conference on Computing and Artificial Intelligence*, 2020, pp. 455--460.

[50] K. a. D. J. Ferencz, "Using Node-RED platform in an industrial environment," *XXXV. Jubileumi Kand Konferencia, Budapest,* pp. 52--63, 2019.

[51] W. Fei, H. Ohno and S. Sampalli, "A Systematic Review of IoT Security: Research Potential, Challenges, and Future Directions," *ACM Computing Surveys,* vol. 56, no. 5, pp. 1--40, 2023.

[52] C. Ozarpa, M. Aydin and İ. Avci, International Security Standards for Critical Oil, Gas, and Electricity Infrastructures in Smart Cities: A Survey Study, 2021.

[53] K. Saeed, M. F. Adak and K. Javeed, "ML-based Smart Voice Analysis for Healthy and Pathological Voice Detection," *EXPERT: Jurnal Manajemen Sistem Informasi dan Teknologi,* vol. 13, 2023.

[54] D. a. F. L. a. D. G. Ancona, M. Leotta, M. Ribaudo and F. Ricca, "Towards runtime monitoring of node. js and its application to the internet of things," *arXiv preprint arXiv:1802.01790,* 2018.

[55] D. Torres, J. P. Dias and A. a. F. H. S. Restivo, "Real-time feedback in node-red for iot development: An empirical study," in *2020 IEEE/ACM 24th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, IEEE, 2020, pp. 1--8.

[56] K. Anam, D. N. Rofi and R. Meiyanti, "Monitoring System for Temperature and Humidity Sensors in the Production Room Using Node-Red as the Backend and Grafana as the Frontend," *Journal of Systems Engineering and Information Technology (JOSEIT),* vol. 3, no. 2, pp. 59--76, 2023.

[57] J. Kopjak and G. Sebestyen, "Event-driven Fuzzy Inference System Implementation in Node-RED," in *2019 IEEE 17th International Symposium on Intelligent Systems and Informatics (SISY)*, IEEE, 2019, pp. 000255--000260.

[58] A. Rajalakshmi and H. Shahnasser, "Internet of Things using Node-Red and alexa," in *2017 17th International Symposium on Communications and Information Technologies (ISCIT)*, 2017, pp. 1-4.

[59] M. Campusano, S. Hacks and E. Y. Kang, "Towards model driven safety and security by design," in *10th international workshop on Quantitative Approaches to Software Quality (QuASoQ 2022)*, CEUR Workshop Proceedings, 2022.

[60] V. K. Jain, A. P. Mazumdar, P. Faruki and M. C. Govil, "Congestion control in Internet of Things: Classification, challenges, and future directions," *Sustainable Computing: Informatics and Systems,* vol. 35, p. 100678, 2022.

[61] A. Saha and C. Srinivasan, "White-box cryptography based data encryption-decryption scheme for iot environment," in *2019 5th International Conference on Advanced Computing \& Communication Systems (ICACCS)*, IEEE, 2019, pp. 637--641.

[62] A. M. Abdullah, "Advanced encryption standard (AES) algorithm to encrypt and decrypt data," *Cryptography and Network Security,* vol. 16, no. 1, 2017.

[63] M.-L. Akkar and C. Giraud, "An implementation of DES and AES, secure against some attacks," in *Cryptographic Hardware and Embedded Systems—CHES 2001: Third International Workshop Paris, France, May 14--16, 2001 Proceedings 3*, 2001, pp. 309--318.

[64] S. Jaloudi, "Communication protocols of an industrial internet of things environment: A comparative study," *Future Internet,* vol. 11, no. 3, p. 66, 2019.

[65] H. Amiri, "Investigation on Modbus protocol to develop an interoperable IIoT platform using Node-RED with MQTT gateway," 2021.

[66] B. A. Mozzaquatro, C. Agostinho, R. Melo and R. Jardim-Goncalves, "A Model-Driven Adaptive Approach for IoT Security," in *Model-Driven Engineering and Software*

Development: 4th International Conference, MODELSWARD 2016, Rome, Italy, February 19-21, 2016, Revised Selected Papers 4, Rome, Italy, 2017.

[67] V. Karagiannis, P. Chatzimisios, F. Vazquez-Gallego, and J. Alonso-Zarate, "A survey on application layer protocols for the internet of things," *Transaction on IoT and Cloud computing,* vol. 3, no. 1, pp. 11--17, 2015.

[68] P.-I. Lee, C.-H. Wang, and W.-C. Hsiao, "Implementation and Analysis on Efficient Proxy-based Multicast Secure Data Sharing Mechanism with CP-ABE Supporting Outsourcing Decryption in IoT Environment," in *Proceedings of the 2023 11th International Conference on Information Technology: IoT and Smart City*, 2023, pp. 134--141.

[69] M. A. A. İ. A. Cevat Ozarpa, International Security Standards for Critical Oil, Gas, and Electricity Infrastructures in Smart Cities: A Survey Study, 2021.