A Model-Driven Framework for Water Supply Management System (MWS)

BY

Marukh Azhar

(Registration No:MS-SE-20-327813)

Supervisor

Dr. Farooque Azam

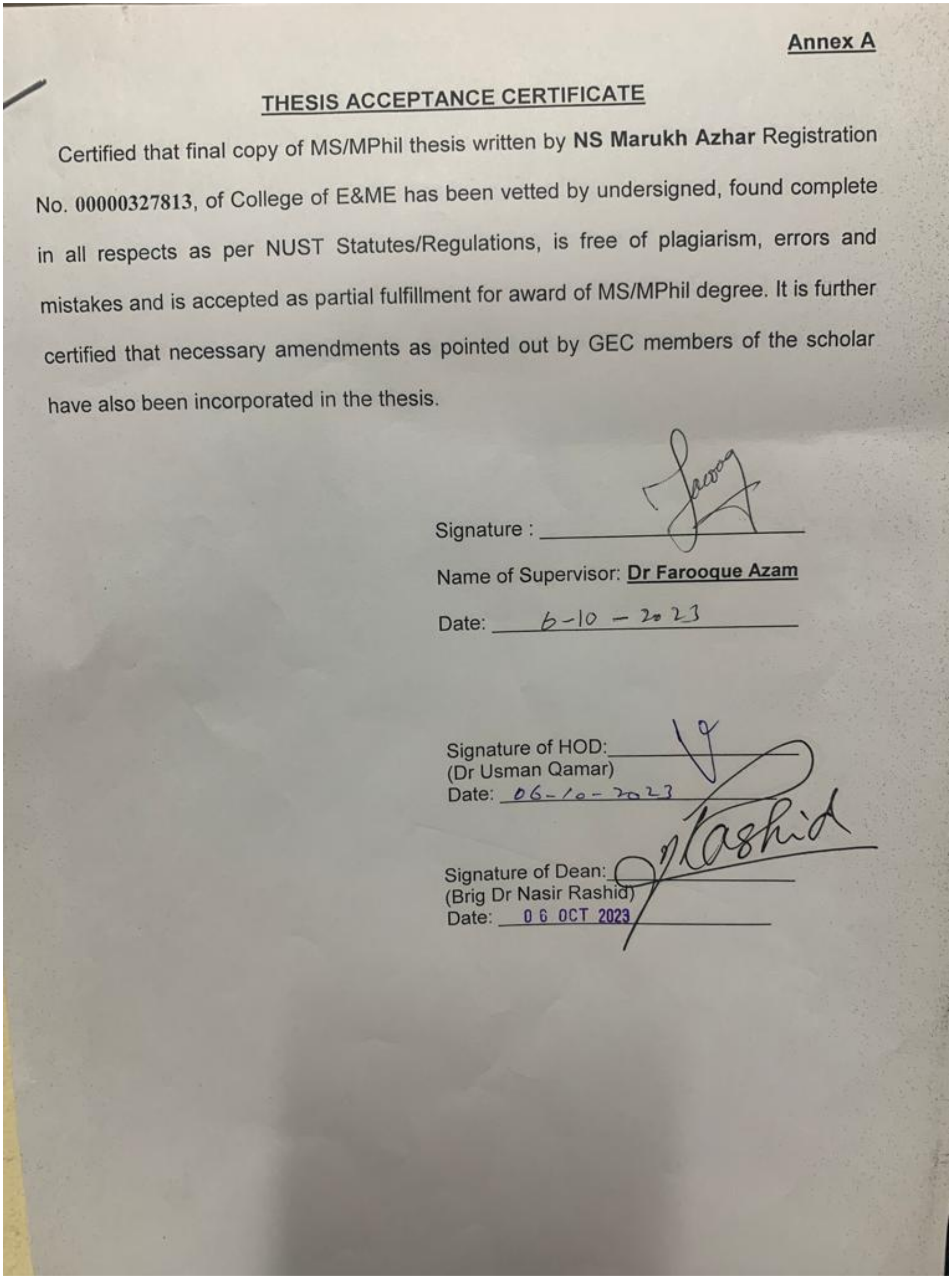
DEPARTMENT OF COMPUTER AND SOFTWARE ENGINEERING

COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

ISLAMABAD

OCTOBER 6, 2023



Annex A

THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS/MPhil thesis written by **NS Marukh Azhar** Registration No. **00000327813**, 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: 6-10-2023

Signature of HOD: [illegible]
(Dr Usman Qamar)
Date: 06-10-2023

Signature of Dean: [illegible]
(Brig Dr Nasir Rashid)
Date: 0 6 OCT 2023

Model-Driven Framework for Water Supply Management System (MWS)

BY

Marukh Azhar
(Registration No:0000327813)

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

Thesis Supervisor:

Dr. Farooque Azam

DEPARTMENT OF COMPUTER AND SOFTWARE ENGINEERING

COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY,

ISLAMABAD

OCTOBER 6, 2023

Dedicated to my beloved Parents, who have motivated and inspired me with their love and support.

ACKNOWLEDGEMENTS

All praise and glory to Almighty Allah (the most glorified, the highest) who gave me the courage, patience, knowledge, and ability to carry out this work and to persevere and complete it satisfactorily. Undoubtedly, HE eased my way and I can achieve nothing without HIS blessings.

I am profusely thankful to my beloved parents who raised me when I could not walk and continued to support me throughout every department t of my life.

I would also like to express special thanks to my supervisor **Dr. Farooque Azam** for his help throughout my thesis. Also, for Software Development and Architecture (SDA) and Model-driven Software Engineering (MDSE) courses which he has taught me. I can safely say that I haven't learned any other engineering subject in such depth as the ones that he has taught.

I would also like to pay special thanks to my Guidance Committee Members **Dr. Wasi Haider Butt, and Dr. Muhammad Umar Farooq** their recommendations are very valuable for the improvement of the work. I would like to pay special thanks to **Muhammad Waseem Anwar** for his tremendous support and cooperation. Each time I got stuck in something; he came up with the solution. Without his help, I wouldn’t have been able to complete my thesis. I appreciate his patience and guidance throughout the whole thesis.

I would also like to thank my family for their support and cooperation. Finally, I would like to express my gratitude to all the individuals who have rendered valuable assistance to my study.

ABSTRACT

Water supply management is a difficult and crucial undertaking that demands effective methods and equipment. Multiple models are necessary to address this difficulty. Model-Driven Architecture (MDA) can be used to successfully construct and modify these models, with the potential to dramatically enhance water supply management. This scientific idea has the potential to completely alter how we manage our water supplies, which makes it very interesting. The capacity to model complete water supply networks virtually is one of MDA's ground-breaking features in this domain. We can test and evaluate a wide range of scenarios using these virtual models without having to spend a lot of money or risk putting the real system at risk.

We propose a thorough model-driven framework that makes use of MDA principles in this study. The first phase in our methodology's multi-step procedure is to convert the conceptual model into a textual representation. A dedicated transformation engine created with the help of the model-to-text transformation tool Acceleo® facilitates this transformation. As a result, timed automata models replace abstract software models. In the complicated field of water supply management, timed automata are a particularly useful tool for modeling and analyzing the timing behavior of complex systems. We can comprehend and improve the temporal elements of water supply processes by adding timed automata models. We carried out real-time case studies to show how useful and successful our framework is. These case studies validate the efficacy of our approach in simulating water supply management systems. With formal verification techniques, we not only ensure standardization but also lessen the possibility of potential system defects. Additionally, the addition of formal verification offers a mechanism for potential problems' early detection. The total reliability of water supply management systems will significantly rise because of our ability to spot issues early and fix them.

Keywords: *Model-Driven, Sirius, Acceleo, Water supply, UPPAAL, Liveness, Deadlock*

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

This chapter provides an overview of the research work. This chapter emphasizes the background study, research technique, problem definition, research contribution, and thesis organization.

1.1. Background study

Model-Driven Software Engineering:

Model-driven software engineering (MDSE) is a subset of software engineering in which models are treated as first-class entities throughout the development process. Models are used in MDSE to define the software system under development, and automatic model transformations are utilized for various model manipulations such as code generation, model integration and deconstruction, and so on[1]. Model-driven techniques are said to boost developer productivity, reduce software building costs (both time and money), improve software reusability, and make the software more maintainable. Similarly, model-driven methodologies promise to help in the early discovery of faults such as design errors, omissions, and client-developer misunderstandings. MDSE demands advanced technology and techniques for software development to get better results. Platform Independent Models (PIM) are changed into Platform Specific Models (PSM) in MDSE techniques by applying transformation rules. Platform platform-independent model (PIM) captures domain-specific requirements but does not include any platform-specific data. The Platform Specific Model (PSM) collects system specs as well as all relevant platform data.

Metamodel M2 level: Meta models describe the higher-level ideas and connections that specify a modeling language's structure and behavior at the M2 level. The types of elements that may be used in models, the connections between these elements, and the constraints on their use are all defined by meta-models at the M2 level. Typically, meta-models at the M2 level are generated using meta-modeling languages like MOF (Meta Object Facility), UML (Unified Modelling Language), or Ecore (used in Eclipse Modelling Framework). Software developers can construct a domain-specific language that is suited to the requirements of a system by building meta-models at the M2 level. These meta-models make it possible to build M1 models that represent instances or instantiations of the ideas they specify.

DSLs, or Domain Specific Languages, are used in modeling to define the specifics of the project in a modeling environment. In contrast to mainstream programming languages like C++ or Java, their scope is narrower, but they are more precise. They are an ideal alternative for the brilliant description of any model due to their quality. Using the DSL, business experts and developers describe the needs and features of the new program, bringing both sides together in the process. Additionally, this method makes it possible for domain experts to comprehend, validate, or alter the system[2].

Model-driven development (MDD) is a methodology that emphasizes the use of models as the main artifacts for creating software, with several degrees of abstraction known as M0, M1, M2, and M3 layers as shown in Figure 1.

The M0 layer, which is the lowest level, contains the system's concrete instances, including its data, procedures, and behavior. The M1 layer captures the system's structure and behavior at a higher degree of abstraction. It leverages modeling languages like UML and is platform-independent, focusing on the logical design of the system. The concepts, relationships, and rules that control the M1 models' structure and behavior are defined by the M2 layer, a metamodel. It gives the modeling language a formal representation and preserves the models' integrity and consistency. The M3 layer is a metamodel that defines the M2 layer and permits the development of unique modeling languages, additions to currently used modeling languages, and transformations among various modeling languages. Code creation and complex model transformations are also supported.

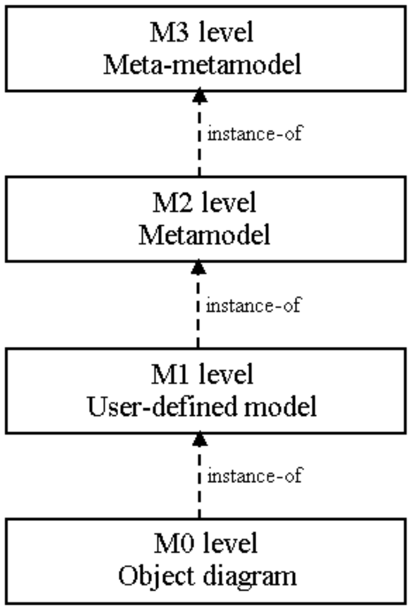


Figure 1. Meta-Model mechanism

Model Transformation:

The process of changing one model into another based on some well-formulated criteria is known as model transformation. The basic objective of model transformation is to close the gap between various abstraction levels, such as turning high-level platform-independent models into low-level platform-specific models or translating models between various modeling languages. Model-to-model transformation and Model-to-text transformation are the two forms of transformation. A model can be translated into a general-purpose programming language or a particular-purpose programming language using a model transformation that uses a model-driven methodology. Bidirectional model transformation requires a particular bidirectional model transformation language to guarantee consistency between two or more models, whereas special-purpose transformation makes it simple to refer to model elements from syntax. Worldwide, several transformation languages like Acceleo, ALT, GReAT, Epsilon family, F-Alloy, and JTL are in use[3].

Verification: Mathematical entities are used to describe and test complicated systems using formal approaches. The formal approaches may be divided into four categories: formal proofs, formal specifications, model checking, and abstraction. Software experts utilize model checking to assure the accuracy of their work, especially when developing large-scale software where it might be difficult to integrate many components. Due to this problem, the software industry used formal procedures, such as model checking, to guarantee the product's reliability. There are two processes involved in model checking. First, a model is created. In the second stage, one or more properties, such as reachability or deadlock, can be validated using property specification languages. Model checking has the following benefits:

- Model checking eliminates manual inspection and investigates every possible model state.
- Model checking improves the whole verification process by enabling thorough evaluation of properties like safety, liveness, or reachability.
- Verification properties are expressed in well-defined property specification languages, such as temporal logic.

Tools for model checking are becoming more and more common in the industry because of Model Driven Engineering (MDE). It might be challenging to choose which tool to employ for a given situation because there are many tools available with various capabilities to address various difficulties. Model-checking techniques use a variety of formalizations, such as timed automata and timed Petri nets. Real-time system verification is frequently done using timed automata, and tools like UPPAAL, KRONOS, CMC, SAVE IDE, and PRISM have been created to assist with this method. Different tools are used for design, analysis, and verification.

Tool Selection: The UPPAAL tool is found to be more suited and adaptable to enable the formal verification of software models after comparison studies amongst model-checking tools. This allows for the detection and correction of these issues during the design phase. As a result, it takes less time and money for the software developer to fix these mistakes after development.

1.2. Proposed Methodology:

As shown in Figure 2, the research approach used in this study entails some successive phases. As shown in Figure 2, this procedure starts with a vital initial step of conducting an extensive literature study with an emphasis on formal verification and metamodel. This literature study provides a basis for comprehending the amount of information that already exists in these fields. Then, a research gap is seen in the literature, and a problem is identified.

The problem identified is subsequently resolved by putting up a model-driven software engineering-based solution. This method makes use of metamodel as key artifacts for software development, analysis, and verification. The proposed solution is fully explained, highlighting its essential elements, and explaining the steps necessary to put it into practice.

The suggested approach is described, and then the specifics of its execution are given. This section provides a thorough explanation of how the suggested solution will be implemented.
It includes all the technical elements, resources, and frameworks that will be applied throughout the implementation procedure, ensuring transparency and stability. Through three case studies, the planned research work is validated. The proposed solution is tested and applied in these case studies, which act as real-world examples and experiments.

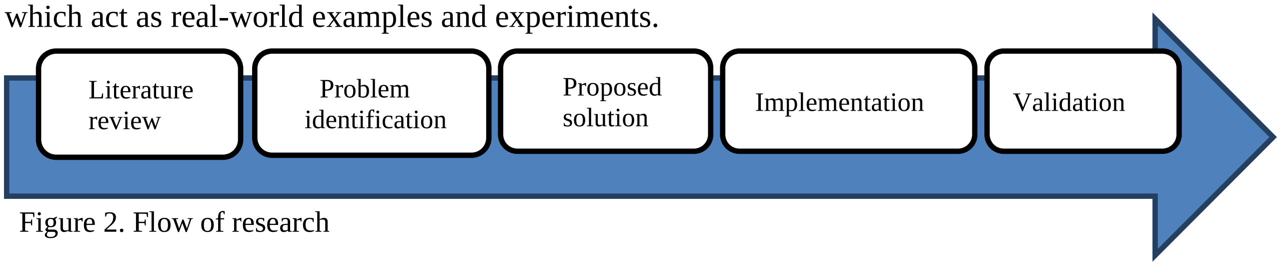


Figure 2. Flow of research

By using this methodology, the study seeks to close the known research gaps, increase knowledge of model-driven software engineering methodologies, and contribute to the fields of formal verification and metamodels.

1.3. Research contribution:

- The Eclipse Modelling Framework (EMF) features were used to build an M2-level metamodel in Eclipse. The graphical editor was used to create the metamodel, allowing for the construction of abstract syntax, semantics, and relationships.

- Transforming software models into timed automata required the development of a transformation engine utilizing Acceleo®. This transition made it possible to study and analyze the timing behavior of complex systems. For this use, timed automata are very appropriate.

- Utilizing the UPPAAL tool, verification properties are carried out during the design phase. At the design level, UPPAAL enables the expression and verification of properties including reachability, deadlock, safety, and liveness. Designers may evaluate and analyze the behavior of their system models using UPPAAL to make sure it is accurate and adheres to the defined specifications.

1.4. Thesis Organization:

The metamodel and the chosen research approach for formal verification are both briefly described in Chapter 1, which introduces the thesis. As seen in Figure 3, it emphasizes the research contribution and offers a clear organization of the thesis. In Chapter 2, a thorough literature analysis is undertaken to look at the earlier work done by various scholars in the fields of model-driven development and formal verification. Chapter 3 provides a thorough justification of the methodology used in the research and discusses the suggested way to address the identified problem. The implementation specifics are covered in Chapter 4 along with the practical and technical components of the development process. In Chapter 5, which focuses on validation, three case studies that have been turned into timed automata and had their associated properties verified are presented. The complete thesis, as well as any limitations encountered throughout the study, are covered in detail in Chapter 6. The thesis is concluded in Chapter 7 with general conclusions based on the findings and suggestions for more research in the area. Figure 3 provides a clear and straightforward picture of the thesis structure and flow by visually representing the thesis organization.

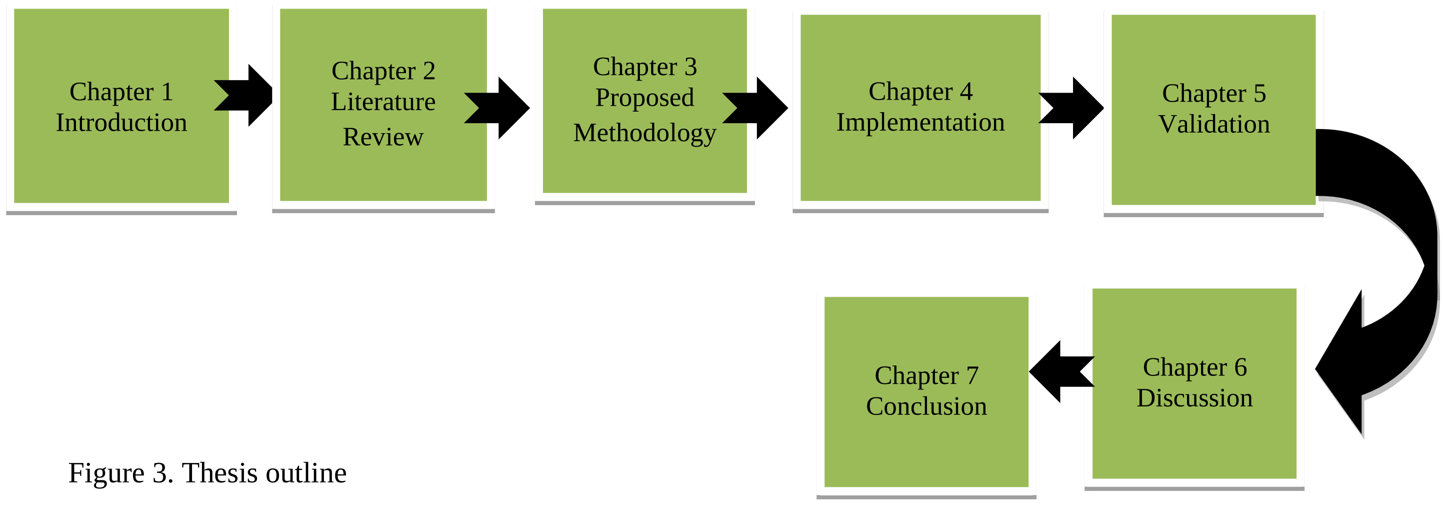


Figure 3. Thesis outline

CHAPTER 2

LITERATURE REVIEW

With two major sections, Chapter 2 conducts an extensive review of the literature. In the first section, 2.1, we thoroughly explore the field of water supply modeling while critically evaluating earlier studies' techniques, outcomes, and research methodologies. while model validation and verification are discussed in section 2.2

2.1 Model-based water supply management:

Systems for distributing water are essential for providing a steady and effective flow of clean water into communities. The use of model-driven frameworks has become increasingly popular for improving these systems' performance, optimization, and decision-making processes. The important research and developments in the subject of model-driven frameworks for water distribution systems are summarized in this review of the literature, with an emphasis on the application of models to enhance system conception, operation, and management.

To create hydraulic models that replicate water distribution networks, several modeling techniques have been investigated, including network topology models, nodal demand models, and pipe flow models. These models depict the physical properties and behavior of the system using mathematical equations and algorithms. Authors [4] have concentrated on enhancing calibration procedures, integrating real-time data, and taking uncertainties into account to improve the accuracy and efficacy of hydraulic models. The administration of water distribution systems has been transformed by the incorporation of real-time data and sensor technology. Researchers have used sensors dispersed across the network to acquire information on water quality, pressure, and flow rates[5]. Improved monitoring, control, and optimization are made possible by this integration. Advanced data-driven methods have been used to analyze sensor data and improve system performance, such as machine learning algorithms and artificial neural networks [6]. The development of decision support systems (DSS) for managing water distribution has been aided by model-driven frameworks. To aid in decision-making, these systems combine hydraulic models, data analytics, and optimization algorithms. DSS products help users with activities including demand forecasting, leak monitoring, and emergency response planning. According to [7], the use of machine learning and optimization approaches offers more precise predictions, optimal resource allocation, and enhanced system performance.

In this research, a systems-of-systems approach is used to examine the resilience of the water sector. To assess present system resilience, suggest future designs, and improve water supply resilience in the face of flooding issues, it uses model-based systems engineering [8]. This study offers limit pressure measurements as a data-driven method for locating leaks in water distribution networks. However, inaccurate interpolation and biased Kriging estimates reduce the suggested approach's accuracy. For greater performance in real-world circumstances, more advancements in interpolation methods and the inclusion of more data are required [9]. This study compares an ensemble approach and conventional anomaly detection methods for detecting attacks in water distribution networks. Comparisons between centrally trained algorithms and multi-stage detection methods reveal that the ensemble methodology outperforms density-based approaches and yields outcomes that are on the same scale as parametric algorithms. Future research will concentrate on evaluating the suggested method in high-dimensional datasets [10].

The importance of water for socioeconomic development and environmental conservation is highlighted in this research, which offers Adaptive Intelligent Dynamic Water Resource Planning (AIDWRP) as a solution for sustainable water management in metropolitan settings. By merging AI technologies with human expertise, AIDWRP uses AI modeling to improve water efficiency and data-driven decision-making. By taking yearly consumption and geographic limits into account, the Markov Decision Process (MDP) enables the optimization of environmental planning and management programs. The goals of AIDWRP are to address problems with water resource management, increase economic effectiveness, and guarantee future sustainable development but the effectiveness of the suggested AIDWRP strategy has not been supported by empirical data or real-world use [11]. To evaluate a water circulation system in a coal-fired power station, this research suggests a RAM analysis methodology. RBD, FTA, and Markov models are used to evaluate performance and pinpoint crucial pieces of equipment. One significant element impacting system availability is the boiler feed pump. The suggested approach helps decision-makers allocate resources and optimize maintenance plans for the water circulation systems in thermal power plants to operate reliably [12]. Long short-term memory (LSTM) deep neural networks are used in this study to suggest a model for predicting drinking water quality. The model uses cutting-edge deep learning theory to handle complicated large data generated by IoT-based monitoring devices for water quality.

The model is tested and trained using information from a Yangzhou water quality monitoring station, showcasing how accurate it is at forecasting future changes in water quality. The study validates the use of LSTM deep neural networks in predicting drinking-water quality it can be improved by including multi-dimensional input datasets and expanding forecasts to numerous monitoring stations while taking spatial dimensions into consideration [13]. This work used random forest regression (RFR) to simulate the distribution of water quality in the Taihu Lake basin, China, using the SHAP approach to determine the affecting factors. The produced maps revealed recurrent patterns for CODMn, TP, and TN, with home and agricultural sources having an influence on water quality. Unexpectedly, due to variations in sewage treatment, water quality in metropolitan areas was positively correlated with population density [14]. This study employs random forest regression to predict the geographic heterogeneity of water quality, with an emphasis on analyzing the model's output. The paper shows how random forest regression may be used to forecast water quality. The results assist efforts to regulate and restore water quality by offering insights into the geographical distribution of water quality. Indicators use model interpretation approaches to comprehend the factors that affect water quality fluctuations [15].The results assist efforts to regulate and restore water quality by offering insights into the geographical distribution of water quality [16].

This work suggests a better technique for quantitatively evaluating the carrying capacity of water resources by system dynamics modeling and fuzzy comprehensive evaluation. The findings show that strategic adjustments to the method of production and the distribution of water resources can increase the carrying capacity for sustainable development [17].To identify leaks and regulate water loss in urban water supply networks, this research offers a unique method employing multiscale neural networks. Using several scales and resolutions of data analysis, the approach increases accuracy. Results from experiments demonstrate notable advancements over conventional approaches, resulting in greater accuracy rates. Future research may examine additional enhancements for more effective leakage identification and water loss control while verifying the technique using real-world information [18].

The paper focuses on the use of artificial intelligence techniques to model and forecast water quality. It emphasizes the application of the feed-forward neural network (FFNN) for highly accurate categorization of water quality and the adaptive neuro-fuzzy inference system (ANFIS) for accurate prediction of the water quality index (WQI). The study shows how artificial intelligence may be used to monitor water quality [19]. The geographical distribution of water quality in the Taihu Lake basin, China, is modeled and interpreted in this work using a combination of random forest regression (RFR) and Shapley additive explanations (SHAP). The SHAP analysis identifies the driving causes, such as residential and agricultural sources, and emphasizes the importance of urban vs rural regions, while the RFR model creates maps showing the distribution of water quality for three criteria [20].AquaCrop is a model that calculates crop productivity depending on water availability and agronomic control. To mimic processes like water infiltration, drainage, evaporation, transpiration, biomass production, and yield, it includes meteorological, crop, soil, and management data. A few factors in the model that describe how crops react to water shortages provide clarity. AquaCrop is useful for creating deficit irrigation schemes and doing scenario analysis since it has a user-friendly interface and real-time tracking [21].

In this paper, an algorithm is created by fusing a water quality simulation model with a multi-objective genetic algorithm (GA) optimization model. A trade-off curve between goals for water quantity and quality is established by the algorithm. The run-time of the GA-based model is decreased by breaking the issue up into yearly and long-term optimization models. The suggested model is used to generate reservoir operating strategies for a reservoir in Iran, proving its usefulness while lowering the computational burden [22]. This work proposes a system for producing risk maps of water supply shortfalls to customers and focuses on preventative water supply management. The M3 main pipe was identified as providing the biggest risk following simulations of main pipe failures using the EPANET program. It is advised to update the M3 main pipe to reduce breakdowns and guarantee uninterrupted water delivery [23].In this study, a low-cost wireless water pressure sensor is presented that may be used to monitor municipal drinking water systems, provide accurate pressure data for system modeling, and pinpoint probable low-pressure regions. In Benton Harbour, Michigan, the system's dependability, and simplicity of deployment were verified [24].

In Khuzestan Province, Iran, the effects of socioeconomic development on water, food, and energy resources were simulated using a water-food-energy system dynamics model. Sustainable management practices were influenced by sensitivity analysis, which led to a mix of water demand and food resource methods. For sustainable water resource management, this strategy increased irrigation efficiency, altered agricultural patterns, decreased losses, and managed food demand [25]. To meet the limited water supply and rising demand, this study offers a model framework for optimizing agricultural water and land resources (AWLR) in a changing environment. The framework allows for complete allocation, balancing competing objectives, and offering flexible strategies for the long-term management of AWLR [26]. The InVEST water yield model is used in this study to evaluate the Danjiang River Basin's role in water conservation. The research demonstrates a declining trend in water conservation and identifies crucial protected zones for ecological preservation and sustainable water management [27]
The stress on water resources brought on by population growth and competing economic sectors is highlighted in this paper. It emphasizes the necessity for efficient management of water resources and the depletion of subsurface water. The assessment focuses on how information technology, in particular the Internet of Things (IoT), can promote sustainable water practices among individuals, farms, and businesses [28]. This article illustrates how data analysis, autonomous decision-making, and process optimization may be used by artificial intelligence (AI) to address problems in drinking water treatment (DWT). It underlines the necessity for efficient contamination characterization and thorough intelligence models for water treatment facilities while discussing the uses and most recent findings of AI in various DWT sectors [29].
The automated evaluation of a water provisioning network's susceptibility to threats using a high-level model and formal models is presented in this research. The method enables the identification of the most probable source of pollution and the impact of threats on network nodes, enabling the management of water resources for sustainable utilization [30]. To effectively use water resources, this article emphasizes the necessity for more intelligent control systems in irrigation networks. It includes a model-driven simulation infrastructure with case studies illustrating the efficiency of the method in Pakistani irrigation networks, as well as a domain-specific modeling language and automatic simulator production [31].

2.2 Formal verification

This section presents several studies on timed automata and formal verification. Complex systems need approaches to validate their design to function correctly. This study has concentrated on the difficult job of formal software model verification throughout the design process.
In research paper [32] activity diagrams to activity hypergraphs verified with CLKs , transforming activity diagrams into Petri nets, and using temporal logic for verification [33], transforming UML activity diagrams into Petri nets and using timed automata for verification [34], integrating formal verification into CPS design using AADL models transformed into timed automata [35], and formal verification using timed automata with synchro[36]. These techniques show enhanced verification and problem identification and have the potential to be used in real-world systems.

In a research study [37], distributed networks are formal verified, with an emphasis on modeling backup protection and handling faults in low-voltage distribution networks. This method varies from others in that it absolves designers of the duty of model correctness testing and instead checks a list of potential outcomes. Properties like liveness, reachability, and deadlock are verified using UPPAAL. Like [38], state flow models translated from EAST-ADL are formal verified using UPPAAL, ensuring timeliness and application requirements. A startup technique for distributed systems is shown in research [39], which is validated using model checking on timed automata. According to [40], formal analysis of EAST-ADL models is accomplished by transforming them into UPPAAL PORT, improving the capabilities of behavioral description and verification. [41] Improves quality through formal verification by extending Business Process Modelling Notation (BPMN) and defining mapping rules for BPMN and timed automata.
[42] Presents a unique method that, for the first time, applies model checking to routing protocols, including AODV, and enables error detection by converting AWN specifications to UPPAAL models. The "Voodu" tool, which [43] presents, enables formal verification of UML sequence and state chart diagrams, and ensures consistency and equivalence between inter-object communication and intra-object behavior. To verify robustness and flexibility, [44] offers a case study on decentralized self-adaptive systems that combines architectural modeling, model-based testing, and UPPAAL. [45] Focuses on fault models and specification models for error handling and system growth, and formal verification of real-time reactive systems using the TROMLAB formalism. The approaches discussed here have potential uses in fault-tolerant frameworks since they can calculate and evaluate new results depending on collected data. In this study area, more investigations and reviews are being conducted.
The research projects provide insightful analysis of formal approaches' modeling and verification methodologies. In [46], the authors focus on testing the safety property using TCTL in UPPAAL and use action timed automata for modeling. [47] Describes DC implementable modeling strategies and validates the reachability property. A modeling tool is suggested in [48], and AVATAR modeling diagrams are used, with an emphasis on meeting liveness criteria utilizing the CTL language in UPPAAL during the verification phase. Models constructed using SysML BDD, IBD, and SMD are shown in [49], and Timed Computation Tree Logic (TCTL) is used for verification to check for deadlock and safety features. Finally, SysML modeling is used in [50] to meet safety and liveness verification criteria utilizing the CTL verification language in UPPAAL, together with state machine and activity diagram models. Together, these papers present a variety of methods, formalizations, and verification characteristics, advancing modeling and verification methodologies in the field.

2.3 Research Gap:

Model-driven water supply management systems have progressed, but there are still gaps that need to be addressed. Previous research studies have focused on specific monitoring problems. Additionally, no suitable framework has been developed for comprehensive water supply management. Therefore, there is a need for a complete methodology based on Model-Driven Architecture (MDA) to facilitate Model Driven Framework for the water supply management system (MWS). We suggest the Model-Driven Water Supply Management Framework (MWS), an open-source system that gives a thorough approach to MWS to fill these gaps. The MWS framework may be extended to include formal verification techniques, making it feasible to rigorously check the accuracy and dependability of the models used in water supply management systems. The focus on formal verification improves the integrity and efficacy of the suggested framework by guaranteeing that the developed system designs satisfy the requirements and are free from critical flaws or vulnerabilities.

CHAPTER 3

PROPOSED METHODOLOGY

Model Driven Development (MDD), which has become more popular in software engineering, is a paradigm change that has made it possible to achieve higher degrees of abstraction and better visualization for complicated systems. By reducing complexity, MDD promotes a better comprehension of complicated circumstances through the development of system models that operate at high levels of abstraction. As primary artifacts and priceless organizational assets, 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. These models can be built using specialized Domain Specific Languages (DSLs) or general-purpose modeling languages like UML. 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.

Examining Figures 4 and 5 from [51] is important in understanding the key contrast between the model-driven method and the traditional code-centric approach. Low-level design and coding are the focal points of the conventional software development life cycle, as shown in Figure 4. 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 over time, which widens the gap between the code and the accompanying documentation and diagrams. The expense and difficulty of maintenance operations are considerably increased by this shortcoming.

In sharp contrast, Figure 5 shows 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.

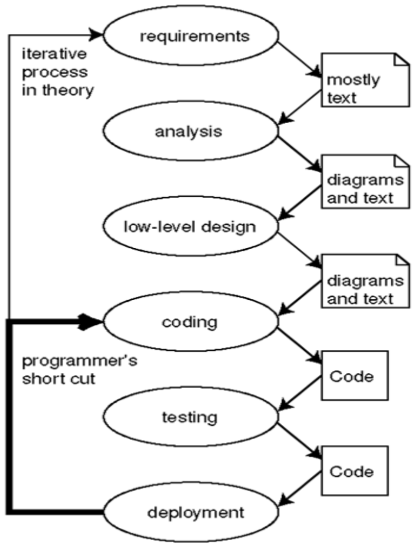


Figure 4. Traditional SDLC

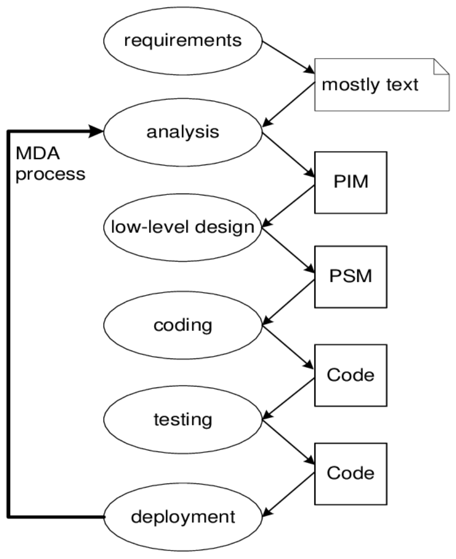


Figure 5. MDA SDLC

The M2 level concept, which is used to define a Domain-Specific Language (DSL), Metamodel, or Platform Independent Model (PIM) in EMF, lies at the foundation of this technique. The Object Management Group (OMG) has established the Ecore Meta-Meta Model as the standard collection of ideas used to describe the M2 level, and these files (with the extension *. ecore) are used to represent M2 levels. DSLs are specified at the M2 level, which captures the concepts, relationships, and constraints within a certain domain. It offers a greater degree of abstraction, enabling the development of domain-specific modeling languages within EMF. To develop system models or M1 level models that adhere to the specified Meta-Model or DSL, these DSLs serve as formal specifications. The technique makes it easier to create system models at the M1 level that comply with the DSL or Meta-Model after it has been specified at the M2 level. These M1-level models are adapted to the needs of the system being modeled and reflect actual examples of the stated concepts and connections. The methodology recommends using Model-to-Model (M2M) or Model-to-Text (M2T) transformations to change the system models. ATL for M2M transformations and Acceleo for M2T transformations are two examples of plugins that may be used with the Eclipse IDE. Models can be transformed from one representation to another using M2M transformations, whereas M2T transformations automate the creation of text-based artifacts from the models.

This methodology aligns with the abstract discussion presented in Figure 6. It illustrates the systematic flow of defining DSLs at the M2 level, developing system models at the M1 level, and performing M2M or M2T transformations using Eclipse IDE plugins.

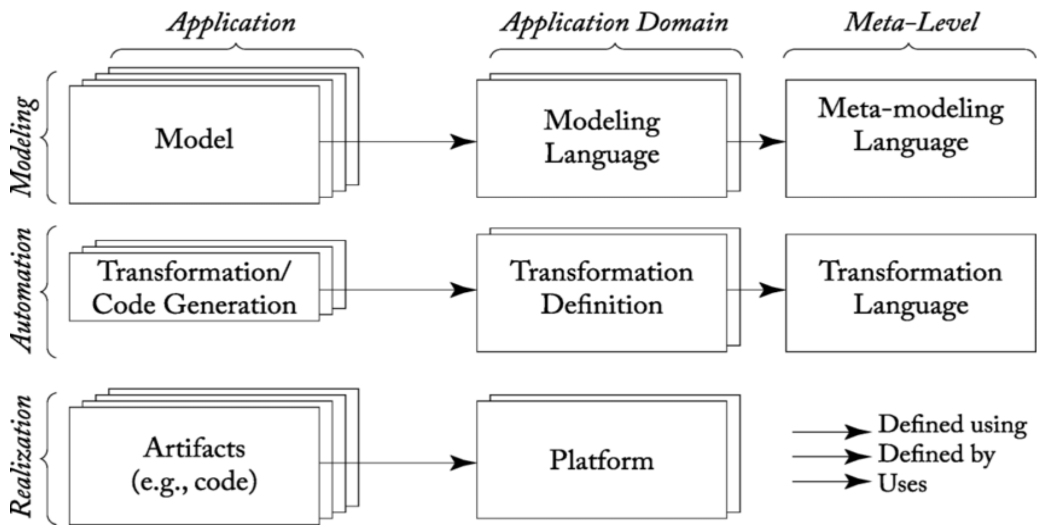


Figure 6. Overview of MDSE methodology

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. Xta 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. Xta files.

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

3.1. Proposed Metamodel:

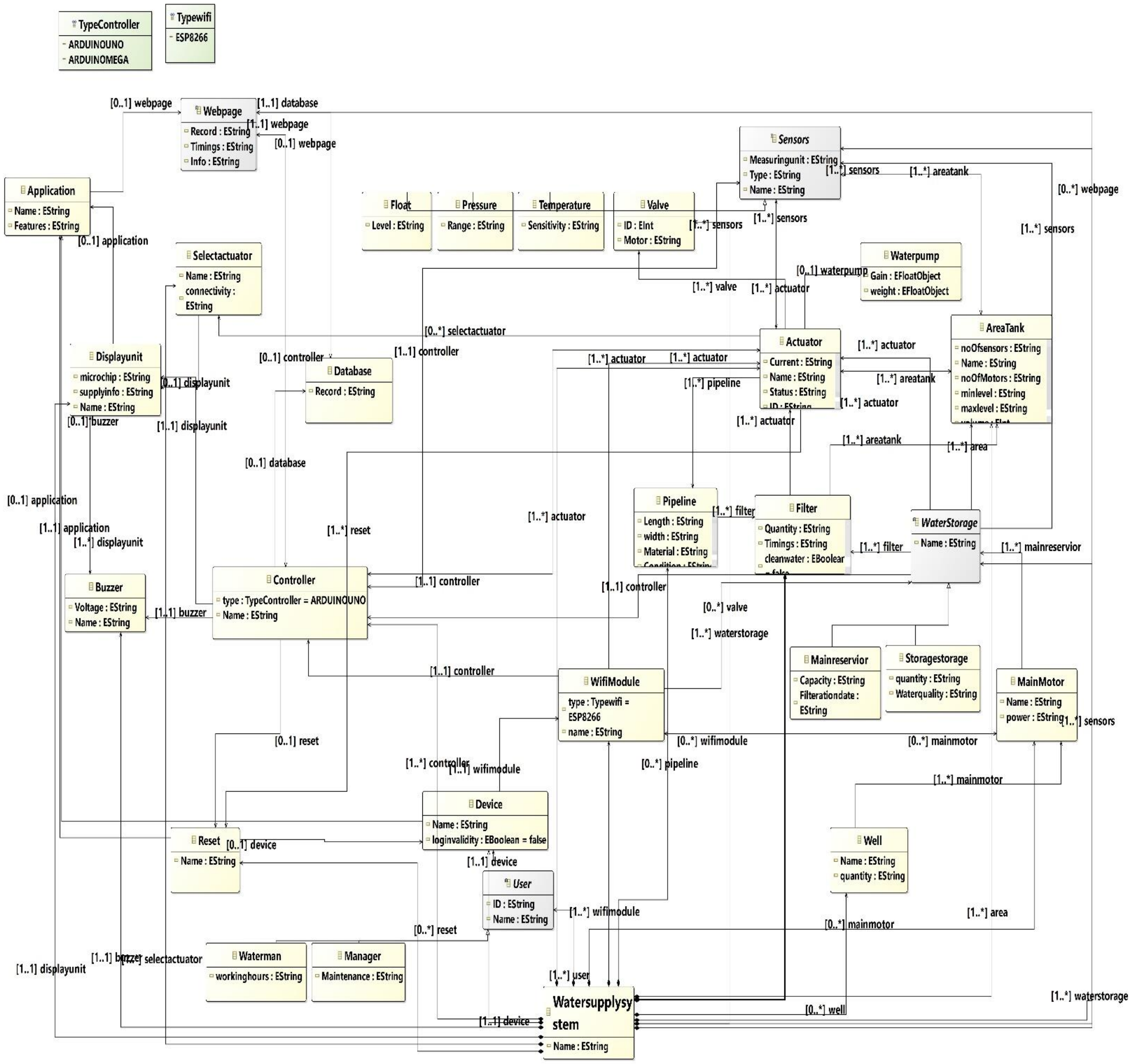


Figure 7. Metamodel of water supply management system

In our proposed framework a generic meta-model for a water supply management system is shown in Figure 7. Which incorporates knowledge from water supply system analysis and design [26]. The model's central class is the 'watersupplysystem,' comprising all other classes, such as WaterStorage, Controller, WifiModule, User, Device, Sensors, Actuators, Displayunit, valve, webpage, filter, pipeline, Database, and Area Tank. Controller, Device, Buzzer, and Displayunit possess one-to-one multiplicity with the primary class, while the remaining classes exhibit a one-to-many ('n') multiplicity. The presented meta-model aims to achieve the desired objectives of the water supply management system. **Controller:** The WifiModule, Actuators, Abstract class WaterStorage, and Abstract class Sensors all reference the Controller class directly. Within the Controller class, there is an Attribute of TypeController which references the webpage, Database, Displayunit, and Buzzer.

WaterStorage is an abstract class and has generalized classes of Mainreservior and Secondary Storage, and refers to Actuator, Sensors, Controller, filter, and Area Tank with attributes of capacity and filtration date. And Secondary has some attributes to check water quality and quantity of water. A **Sensor** is also an abstract class with attributes of status, type, and measuring unit and has generalized classes of Temperature, float, and Pressure. The temperature sensor is used to measure the temperature of the liquid. A float sensor is used to detect the level of liquid within a tank whereas a pressure sensor is used to find the pressure of water.

User which refers to Device with (one-to-one) multiplicity is also an abstract class that has generalized classes of Manager and Waterman. Waterman operates the whole water supply system and detects the leakages in the pipeline. On the other hand, the manager is to supervise the whole system to resolve water issues and update the information on maintenance. The **Device** class is composed of the primary water supply system class and exhibits one-to-one multiplicity with the wifiModule**.** The device is taken as an application through which wifiModule gets the command and works accordingly. The **AreaTank** class is directly referred to by Abstract class WaterStorage, Abstract class Sensor, Actuator class, and filter class and has attributes of several sensors attached, number of motors attached, minimum level of water, and maximum level of water.

As a result of the one-to-one multiplicity of the Controller class, one **WifiModule** can be synchronized with only one Controller at any given time. Additionally, the Controller class refers to the Actuator class with one-to-many multiplicities, indicating that multiple motors can be connected to a single WifiModule. The AreaTank class represents various water supply areas and possesses a one-to-many multiplicity with Actuators. This class also has a connection with WaterStorage. A **buzzer** or beeper is an audio signaling device. It is a device that makes an alarm when the current passes through its circuit. When water comes to its specified level, the buzzer becomes active, and makes an alarming sound. The **Liquid crystal display** on Arduino is used to get the latest updates about the water supply. It shows different commands. Figure 7 shows the meta-model which was created using the modeling software Eclipse.

3.2. 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 clearly

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].

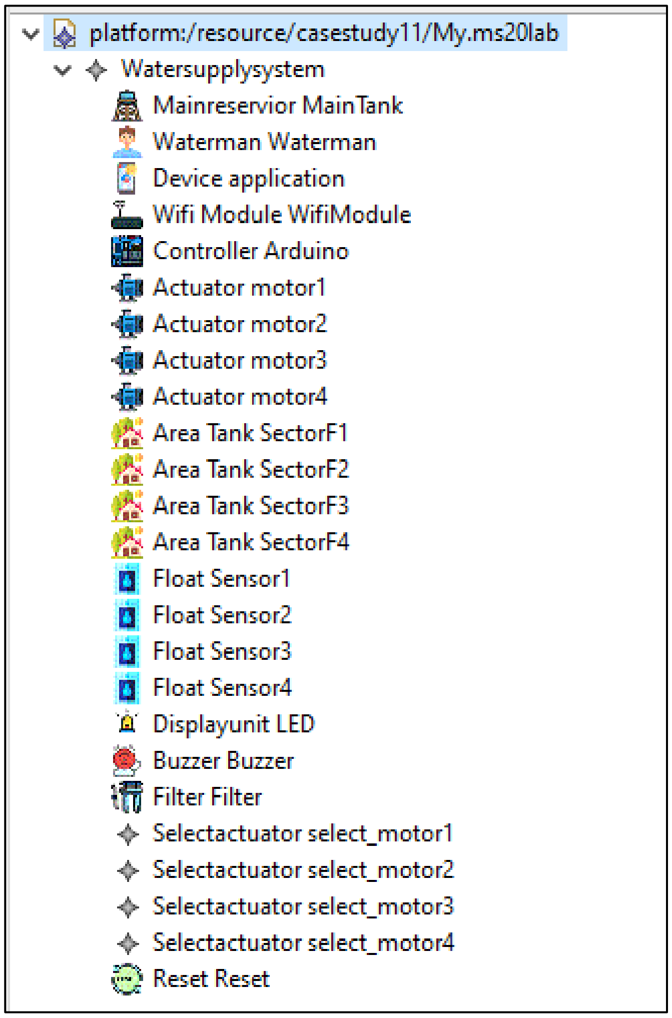
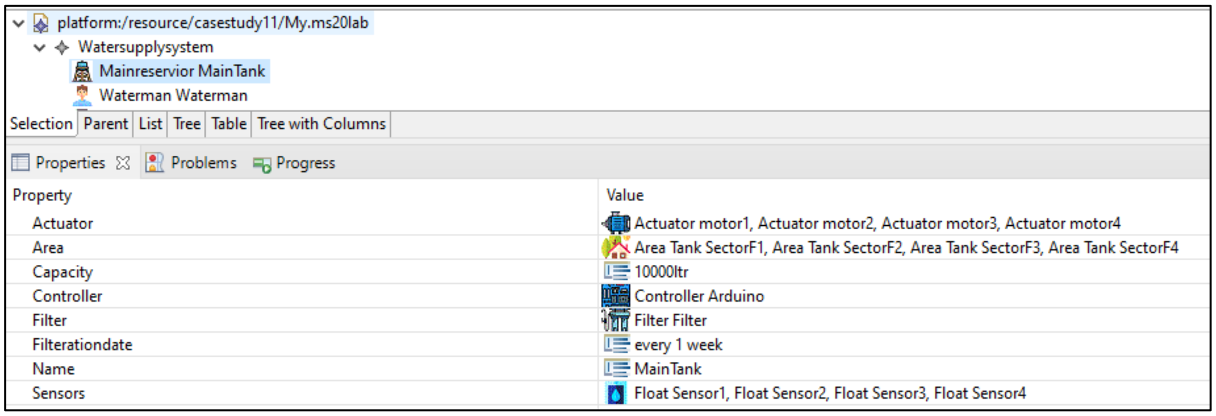

platform:/resource/casestudy11/My.ms20lab
Watersupplysystem
Mainreservior MainTank
Waterman Waterman
Device application
Wifi Module WifiModule
Controller Arduino
Actuator motor1
Actuator motor2
Actuator motor3
Actuator motor4
Area Tank SectorF1
Area Tank SectorF2
Area Tank SectorF3
Area Tank SectorF4
Float Sensor1
Float Sensor2
Float Sensor3
Float Sensor4
Displayunit LED
Buzzer Buzzer
Filter Filter
Selectactuator select_motor1
Selectactuator select_motor2
Selectactuator select_motor3
Selectactuator select_motor4
Reset Reset

Figure 8. Treeview of metamodel

Clicking on the "MainReservoir" node in Obeo Eclipse's TreeView opens an enlarged view that shows all the attributes and information related to the "MainReservoir" concept as shown in Figure 9. This provides details about the "MainReservoir" 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 "MainReservoir" notion, providing a greater comprehension of its attributes and behavior inside the metamodel.



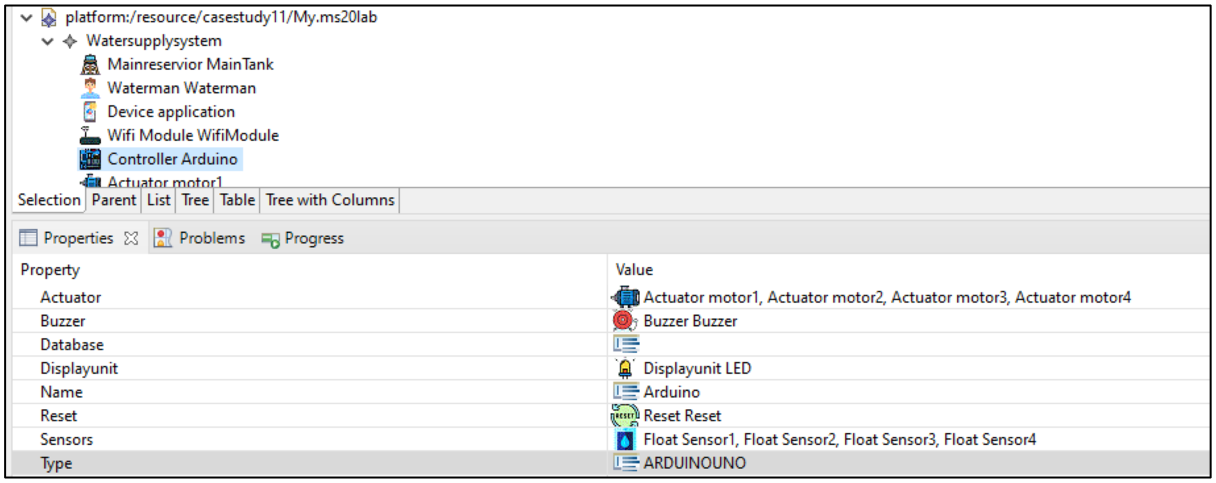
platform:/resource/casestudy11/My.ms20lab
- Watersupplysystem
  - Mainreservior MainTank
  - Waterman Waterman

Selection | Parent | List | Tree | Table | Tree with Columns
Properties | Problems | Progress

Property	Value
Actuator	Actuator motor1, Actuator motor2, Actuator motor3, Actuator motor4
Area	Area Tank SectorF1, Area Tank SectorF2, Area Tank SectorF3, Area Tank SectorF4
Capacity	10000ltr
Controller	Controller Arduino
Filter	Filter Filter
Filterationdate	every 1 week
Name	MainTank
Sensors	Float Sensor1, Float Sensor2, Float Sensor3, Float Sensor4

Figure 9. Properties of main reservoir

Users may quickly view the various attributes related to Arduino by selecting the Arduino node in the TreeView as shown in Figure 10. This illustration demonstrates how TreeView makes it possible to show attributes consistently and completely for various elements across the metamodel.



platform:/resource/casestudy11/My.ms20lab
- Watersupplysystem
  - Mainreservior MainTank
  - Waterman Waterman
  - Device application
  - Wifi Module WifiModule
  - Controller Arduino
  - Actuator motor1

Selection | Parent | List | Tree | Table | Tree with Columns
Properties | Problems | Progress

Property	Value
Actuator	Actuator motor1, Actuator motor2, Actuator motor3, Actuator motor4
Buzzer	Buzzer Buzzer
Database	
Displayunit	Displayunit LED
Name	Arduino
Reset	Reset Reset
Sensors	Float Sensor1, Float Sensor2, Float Sensor3, Float Sensor4
Type	ARDUINOUNO

Figure 10.Properties of Arduino

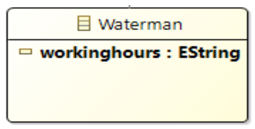
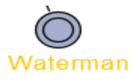
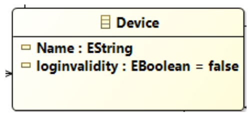
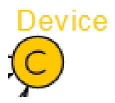
3.3. Transformation Rules:

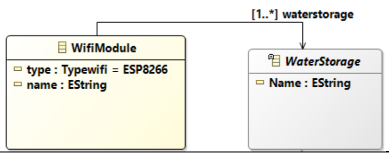
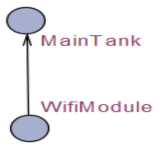
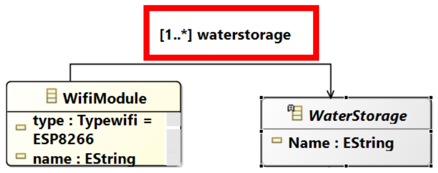
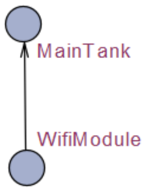
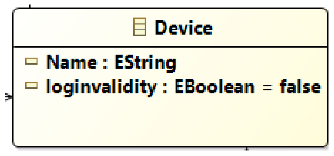
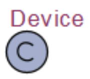
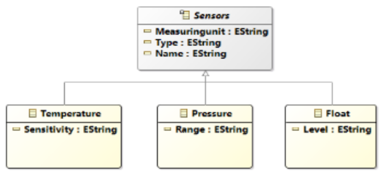
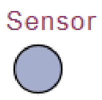
In model transformation procedures, transformation rules are essential. The logic and procedures necessary to transform input models into desired output models or textual artifacts are specified by these rules. The components, properties, and relationships in the input model are mapped and converted 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 UPPAAL models, the intended output. The elements, relationships, and constraints described in the metamodel are mapped to their appropriate representations in UPPAAL according to these transformation rules, which create a systematic method for doing so. In this study, the M1 level models are transformed into a .xta 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 UPPAAL are defined by templates, locations, and edges. The metamodel uses attributes and methods to specify properties, whereas UPPAAL uses local and global variables to express them. In UPPAAL, associations from the metamodel are converted to edges that connect the various components. The UPPAAL process transforms constraints from the metamodel into guard conditions for transitions. In contrast to UPPAAL, which allows automated verification through model checking and simulation, verification in the metamodel is often carried out manually through testing and debugging. Finally, UPPAAL exports the model as an XTA file suited for simulation and formal verification, while the metamodel is stored as source code files in programming languages like Java, C++, or Python [55].

Table 1. Transformation rules

Class 	Initial Location 	Class in metamodel is mapped to Initial Location in UPPAAL
Class  Device Name : EString loginvalidity : EBoolean = false	Location 	Class in metamodel is mapped to Location in UPPAAL
Associations	Edge	The connection between two classes is defined as Association in metamodel

 [1..*] waterstorage WifiModule type : Typewifi = ESP8266 name : EString WaterStorage Name : EString		whereas in UPPAAL it is defined as Edge
Multiplicity  [1..*] waterstorage WifiModule type : Typewifi = ESP8266 name : EString WaterStorage Name : EString	Edge 	Multiplicity in a metamodel is analogous to the edges in UPPAAL
Constraints	Guard	For each constraint in the M2 metamodel, create a guard condition for the corresponding transition in the UPPAAL process.
Class  Device Name : EString loginvalidity : EBoolean = false	Committed Location 	Class in metamodel is mapped to the committed location in timed automata.
Subclass  Sensors Measuringunit : EString Type : EString Name : EString Temperature Sensitivity : EString Pressure Range : EString Float Level : EString	Location 	Subclass in Metamodel is mapped as Location in UPPAAL
Attributes	Global and Local Declaration	In metamodel, Properties are defined using attributes while in UPPAAL it is defined in local and global declaration.

Summary:

In summary, 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 Xta files, and then verifying these files in UPPAAL. 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 UPPAAL's strong verification capabilities. We guarantee a smooth and automated process of building UPPAAL models by precisely specifying these rules, which capture the requirements 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 UPPAAL representation, we may leverage the rigorous verification capabilities of UPPAAL. To make the transition from the metamodel to the UPPAAL 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 (Mirpur city):

Mirpur Azad Kashmir is situated in Pakistan's northern region. The city is renowned for its picturesque beauty and vibrant culture, but it is currently dealing with a significant water management issue. People are experiencing severe water shortages because of poor infrastructure management and water resource mismanagement. The Mangla Dam, which supplies water to Mirpur and other nearby towns and cities, is the primary source of water for the city, however, several factors frequently cause the water supply to be interrupted.
The city's water distribution system is out-of-date and ineffective, and it cannot meet the demands of the expanding population. People sometimes rely on private water tankers, which demand excessive prices, because the infrastructure cannot supply water to the entire city. The improper use of water resources is the second problem. The Mangla Dam's water supply is frequently interrupted for a variety of reasons, including maintenance, power failures, and political upheaval. Wells and tube wells have dried up because of the misuse of water resources, which has caused the water table to be depleted. Mirpur residents are suffering from a severe water deficit, particularly during the summer [56]. By putting in place appropriate water management procedures and keeping an eye on the supply system, the water delivery system may be enhanced. By developing a more effective and efficient approach, my research effort aims to enhance the city's water supply system.

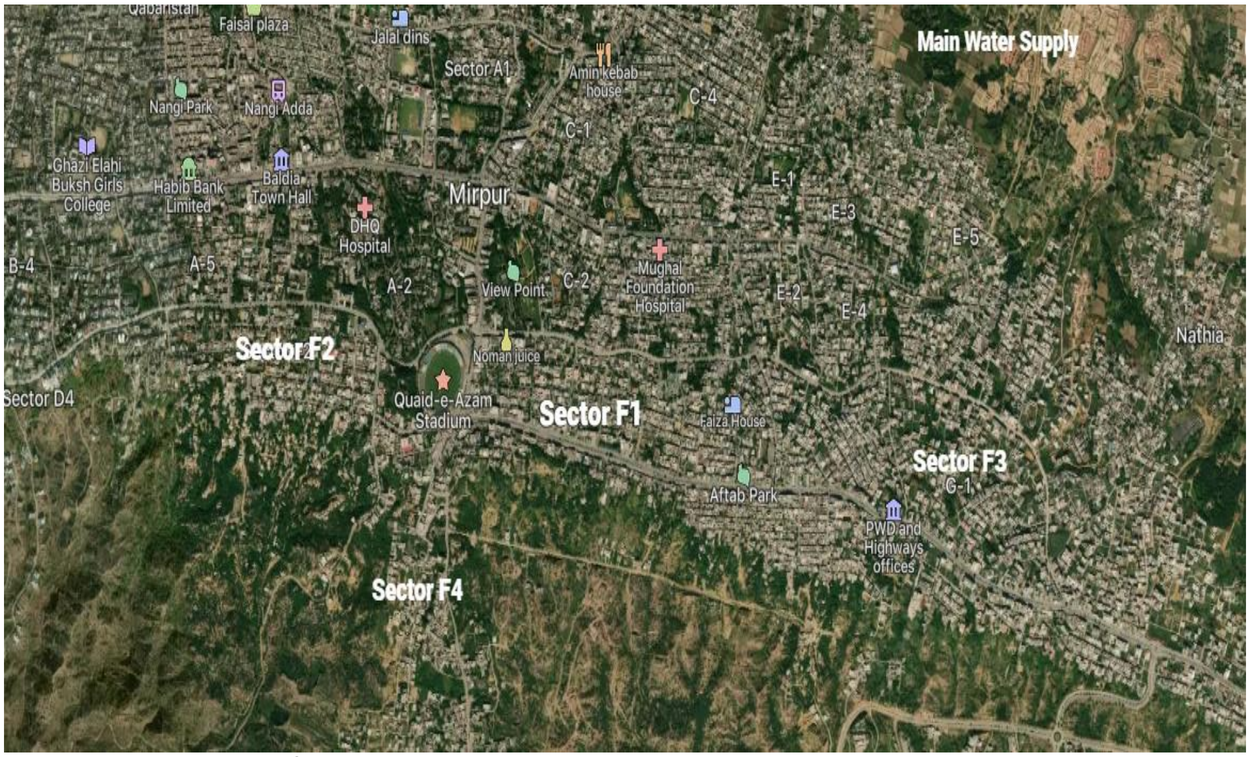


Figure 11. Aerial view of Mirpur city

For this purpose, we have determined to concentrate on four different sectors of the city. By concentrating on these sectors, we hope to highlight the advantages of our new system and illustrate how it might enhance the city's entire water supply. Our goal is to persuade the city to adopt our system on a broader scale to address the city's present water supply problems.

A full water delivery system is modeled in this section to test our proposed framework. Fig.3 shows the complete model of the Water supply management system. The proposed meta-model consists of a root-class Water supply management system that displays complete management of the water system in a city or in an area where it seems difficult to implement a complete water management system. The implementation involves water storage that consists of a secondary storage system and the main reservoir that is responsible for delivering water to the end user. This system is applied to an area where people are facing issues of water deficiency, which is developed using an ARDUINO controller, Wi-Fi module, Buzzer, Display unit, and Actuator. In this scenario, 4 sectors of a city are shown. Water is supplied to all four sectors one by one. A Wi-Fi module and Arduino are interlinked with each other through RX TX. A mobile app is also developed to manage the whole system. The waterman can manage the system even if he is far away from the city. He can turn the respective motor on/off from anywhere. Suppose we give a command through an application to turn motor 1 ON, the transformer of that specific motor will come into a working state and provide power to the motor. Following the same procedure, the rest of the motors will work. We have connected pipes with these motors through which we can provide water to different sectors of a city. If we press 1 through an application, then motor 1 will turn ON, and water is supplied to sector 1 and vice versa. Sensors were also attached to the water tank to check the water level and to ensure the equal distribution of water among different sectors. The sensors help the waterman to check the amount of water in a tank and when the water is successfully delivered to the sector, the sensor will help to display the accurate amount of water delivered on an LED attached to ARDUINO. Suppose the maximum level of the main water Tank is 1000ml, then water supplied to all four sectors is at least 250mlSensors are attached that will display a message on the display unit if the water level is equal to or more than 250ml, indicating that water is fully delivered to Sector 1. This will assist us in identifying water loss. If there is a problem, such as not enough water being provided to each sector or less than 250ml, no notification will be displayed. A Buzzer is also attached to the controller. Once the water is successfully delivered to the sector the Buzzer starts to beep and the display unit along with the controller shows a display message that 250ml of water has been successfully delivered to sector 1.

The process involves the following steps: -
Step a. Waterman accesses the mobile application.
Step b. Give a command to wifiModule through a mobile application.
Step c. WifiModule turns the motor ON or OFF accordingly.
Step d. WifiModule is connected to Arduino; it also passes information to Arduino.
Step e. LCD (Liquid crystal display) relates to Arduino, whenever the motor turns ON; a message is displayed on LCD that water starts delivering to sector 1.
Step f. Sensors are attached, so when water reaches maximum level, again the message is displayed that water is fully delivered to sector 1. In this way, the water supply is managed without wastage and is fully delivered where required; this will help us in detecting water losses. If there is some issue like water is not fully delivered to each sector, then no message will be shown.
The hardware is managed with the aid of a mobile application on a smartphone and by scheduling the times, as shown in the diagram above Figure 12.

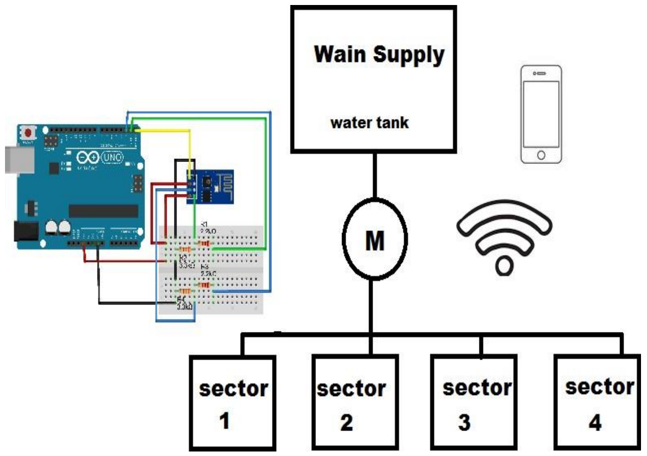
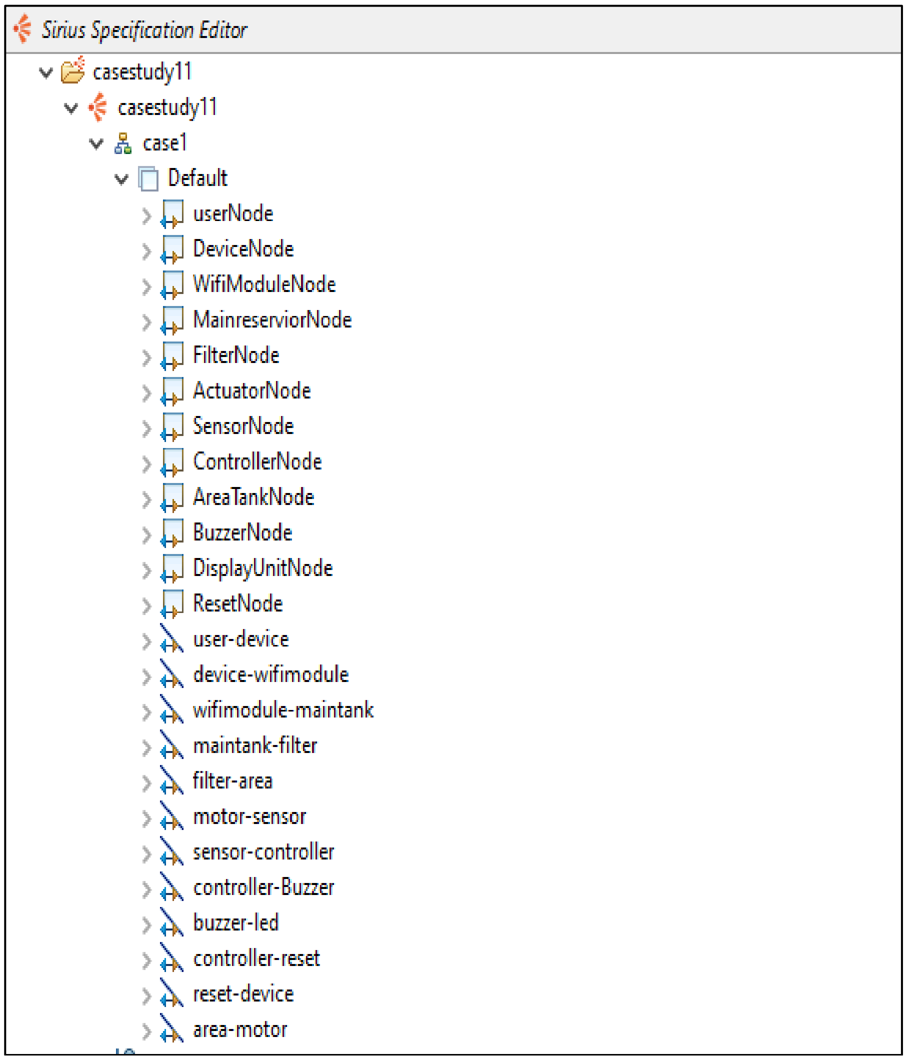


Figure 12. Block diagram of water supply

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.



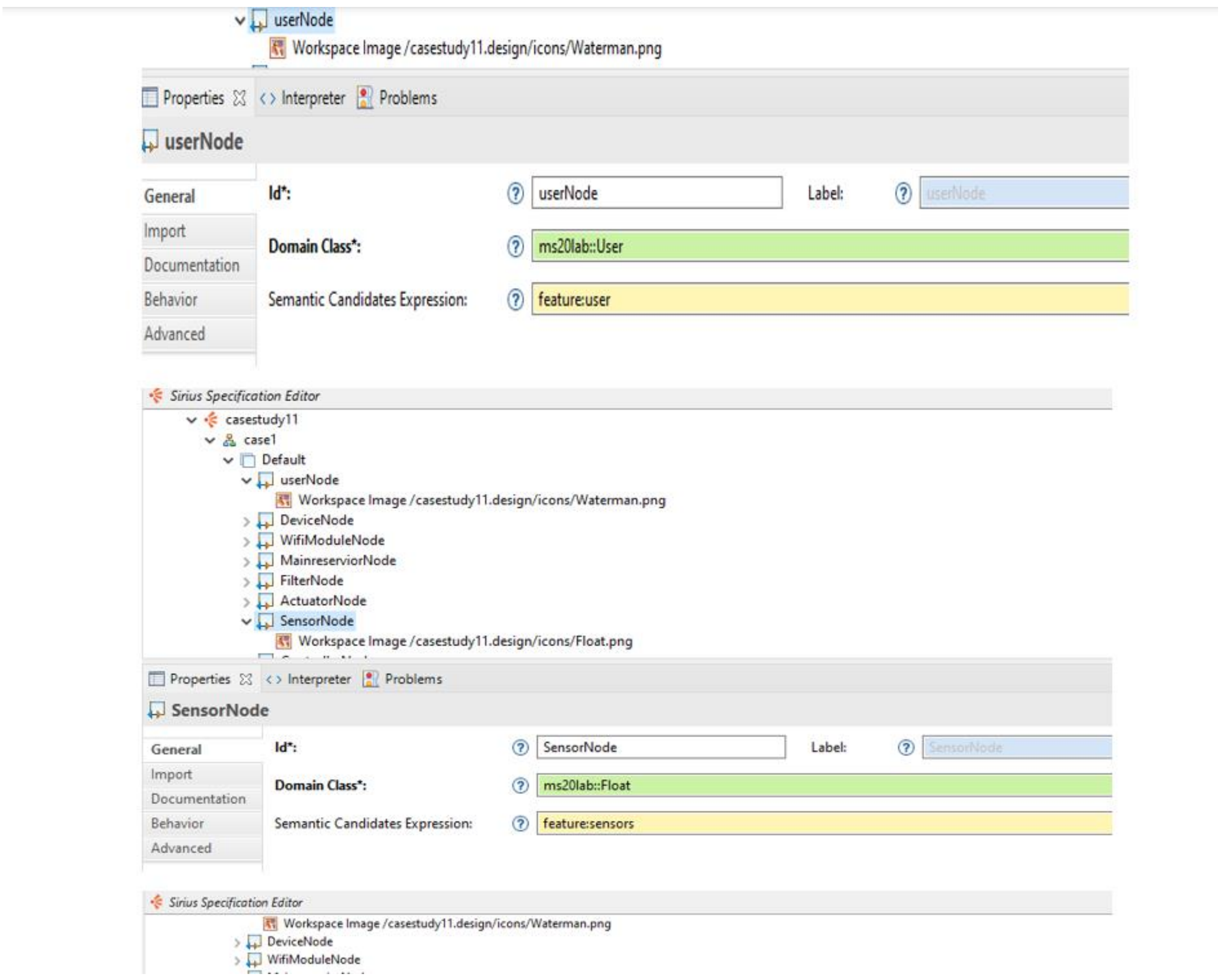
```
Sirius Specification Editor
casestudy11
  casestudy11
    case1
      Default
        userNode
        DeviceNode
        WifiModuleNode
        MainreserviorNode
        FilterNode
        ActuatorNode
        SensorNode
        ControllerNode
        AreaTankNode
        BuzzerNode
        DisplayUnitNode
        ResetNode
        user-device
        device-wifimodule
        wifimodule-maintank
        maintank-filter
        filter-area
        motor-sensor
        sensor-controller
        controller-Buzzer
        buzzer-led
        controller-reset
        reset-device
        area-motor
```

Figure 13. Sirius Design view

In this Figure 13, 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 in a simple and understandable way.
Each node in the Sirius diagram has specific properties that are accessible by clicking on the node. For instance, when you click on the User 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 User node's properties panel:

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

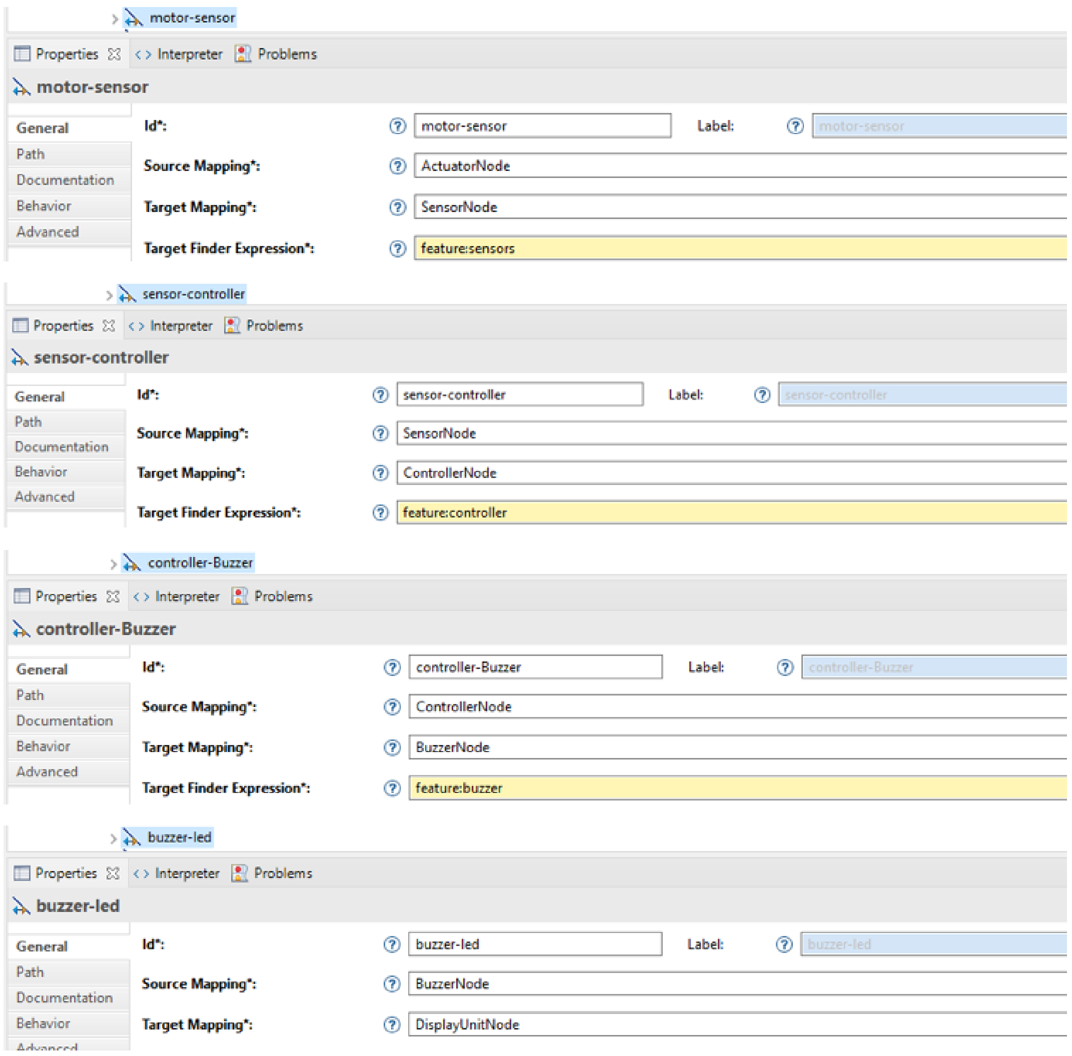
   You may guarantee that the final representation is coherent and functionally related by precisely filling out these properties for each node as shown in Figure 14. To ensure that the final representation appropriately depicts the underlying architecture and data exchanges, each node develops into a well-defined and useful component of the system.



Id*:	userNode	Label:	userNode
Domain Class*:	ms20lab::User		
Semantic Candidates Expression:	feature:user		

Id*:	SensorNode	Label:	SensorNode
Domain Class*:	ms20lab::Float		
Semantic Candidates Expression:	feature:sensors		

Figure 14. Nodes Properties in design view



motor-sensor

Id*:	motor-sensor	Label:	motor-sensor
Source Mapping*:	ActuatorNode		
Target Mapping*:	SensorNode		
Target Finder Expression*:	feature:sensors		

sensor-controller

Id*:	sensor-controller	Label:	sensor-controller
Source Mapping*:	SensorNode		
Target Mapping*:	ControllerNode		
Target Finder Expression*:	feature:controller		

controller-Buzzer

Id*:	controller-Buzzer	Label:	controller-Buzzer
Source Mapping*:	ControllerNode		
Target Mapping*:	BuzzerNode		
Target Finder Expression*:	feature:buzzer		

buzzer-led

Id*:	buzzer-led	Label:	buzzer-led
Source Mapping*:	BuzzerNode		
Target Mapping*:	DisplayUnitNode		

Figure 15. Edges properties in design view

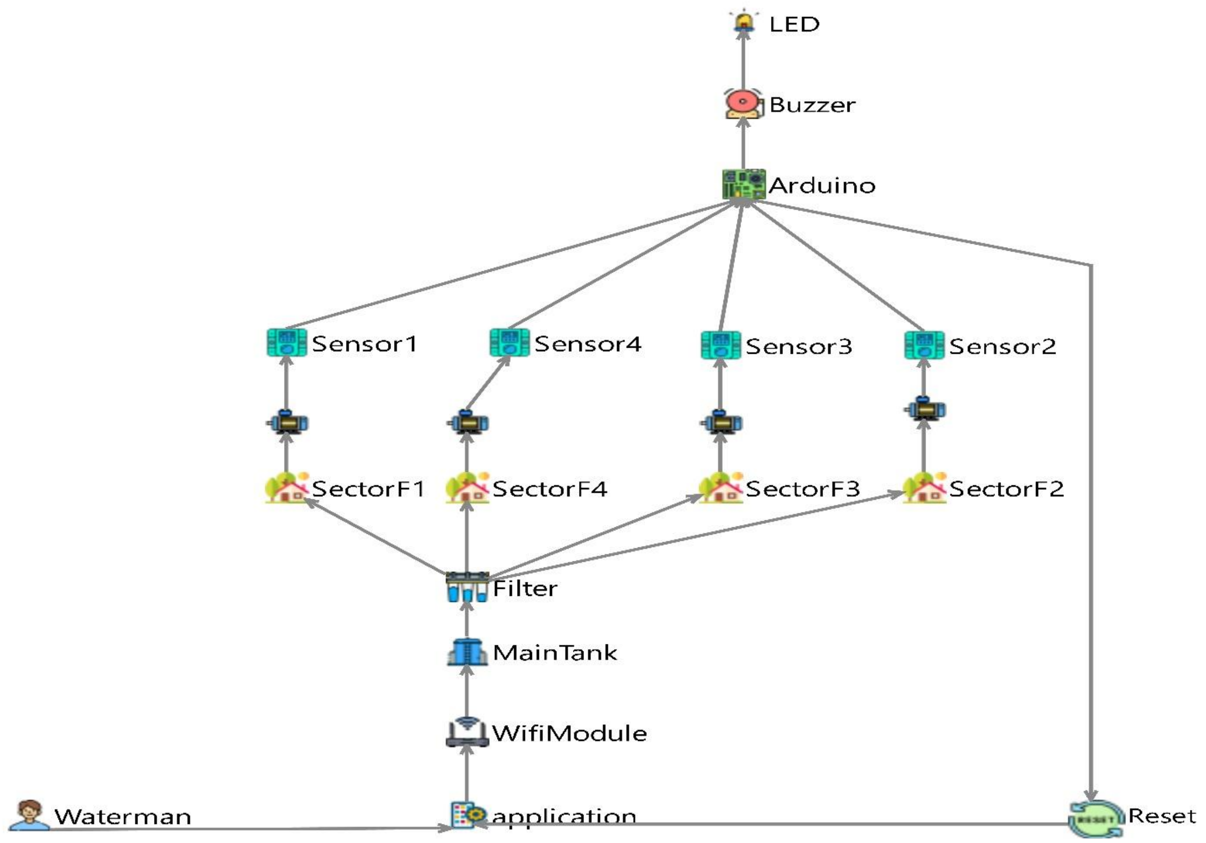


Figure 16: Graphical Representation in Sirius

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(College of EME):

The campus of EME College is the subject of the case study. It has a variety of academic and administrative buildings, including engineering departments like Electrical, Computer, Mechatronics, and Mechanical as well as administrative buildings like the College Head Quarter, Academic Studies Group (ASG), and Cadet Battalion. Boys' and girls' hostels, an Army Public School, cafes, the central mosque, the college's central library, and an auditorium are additional features.

The college's main task is successfully controlling the water supply to prevent any water-related problems on campus. A central overhead water reservoir serves as the primary water supply source. Given the sizeable population of residents and users of facilities like hostels, schools, mosques, libraries, and auditoriums, it is essential to optimize water management to satisfy the variety of water demands and provide a constant water supply across the campus. A thorough water management strategy should consider issues specific to the campus, such as rainwater collection, distribution efficiency, and conservation measures. For quick leak identification and repair, usage patterns in hostels, college, and auditoriums must be examined, as well as routine infrastructure maintenance. EME College can guarantee a sustainable and dependable water supply for all its facilities and people by putting into practice a well-designed and proactive water management plan. This will also help with water conservation efforts and lessen the college's environmental impact.

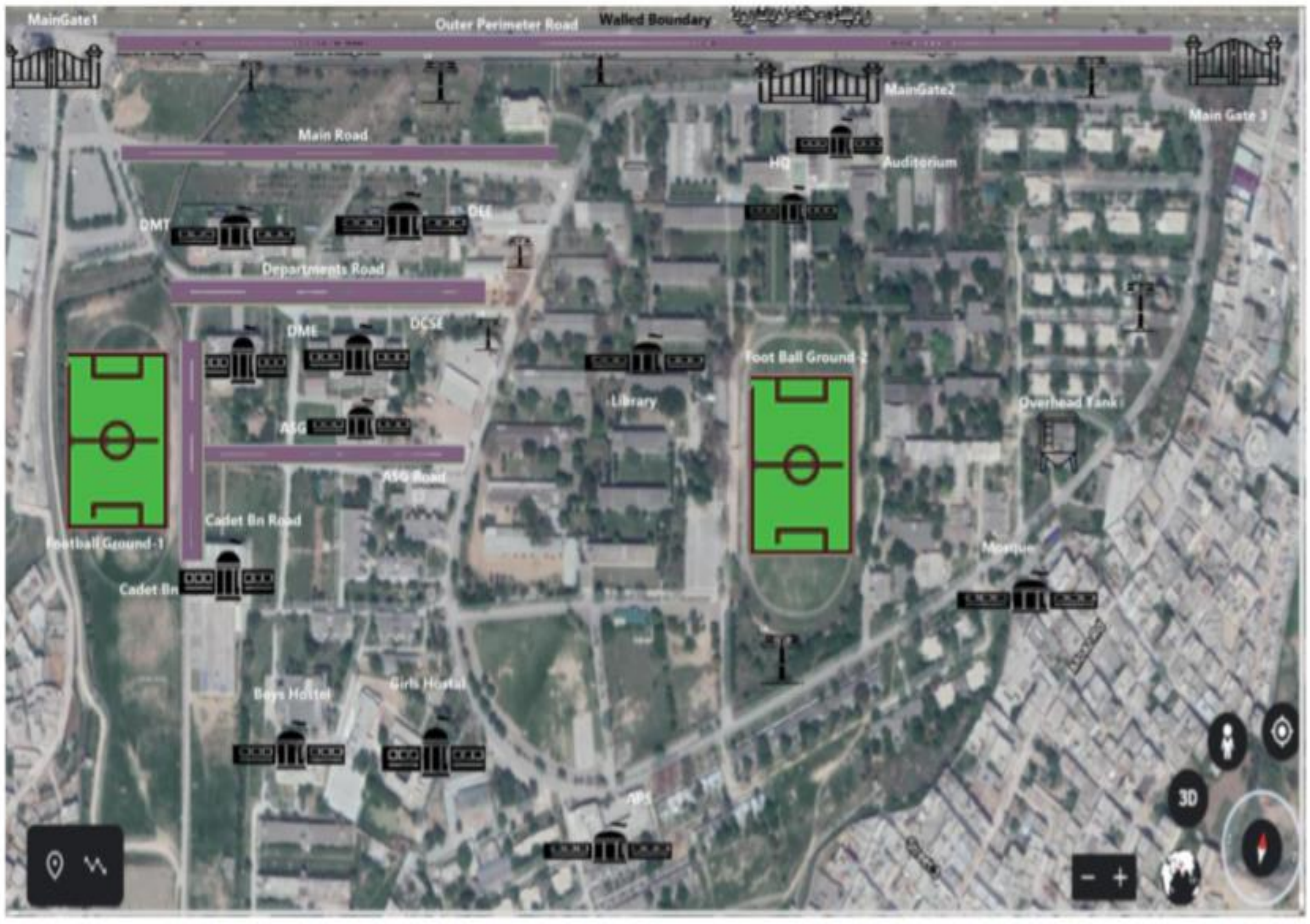
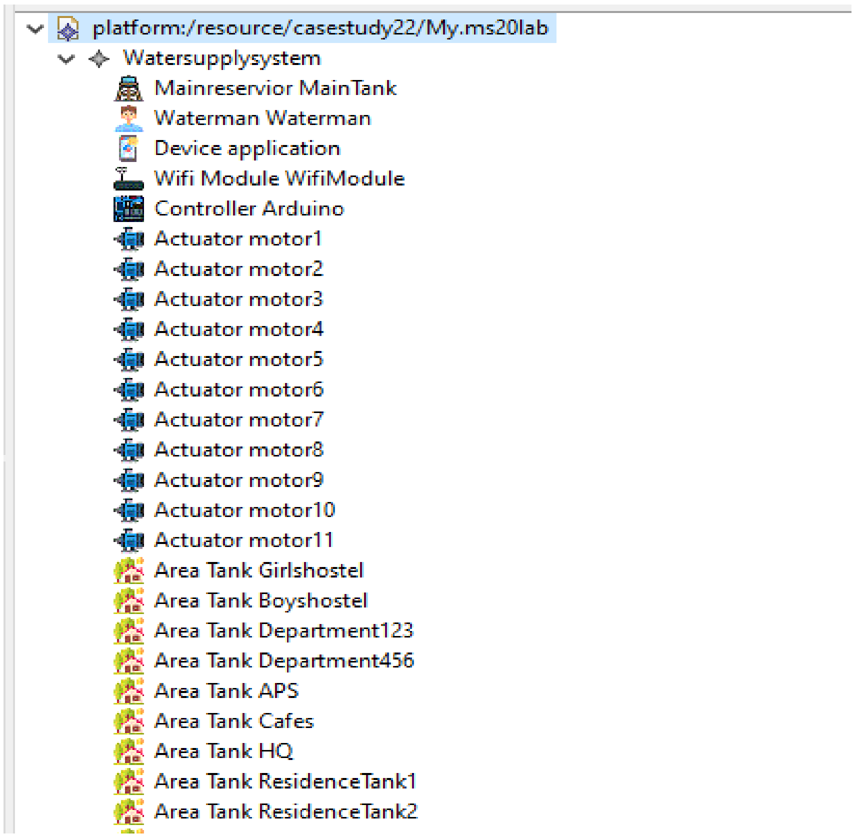


Figure 17. EME college view

Utilizing the Eclipse Modelling Framework (EMF) and Sirius to establish a robust and adaptable modeling environment, we construct the college case study using the model-driven approach. In the

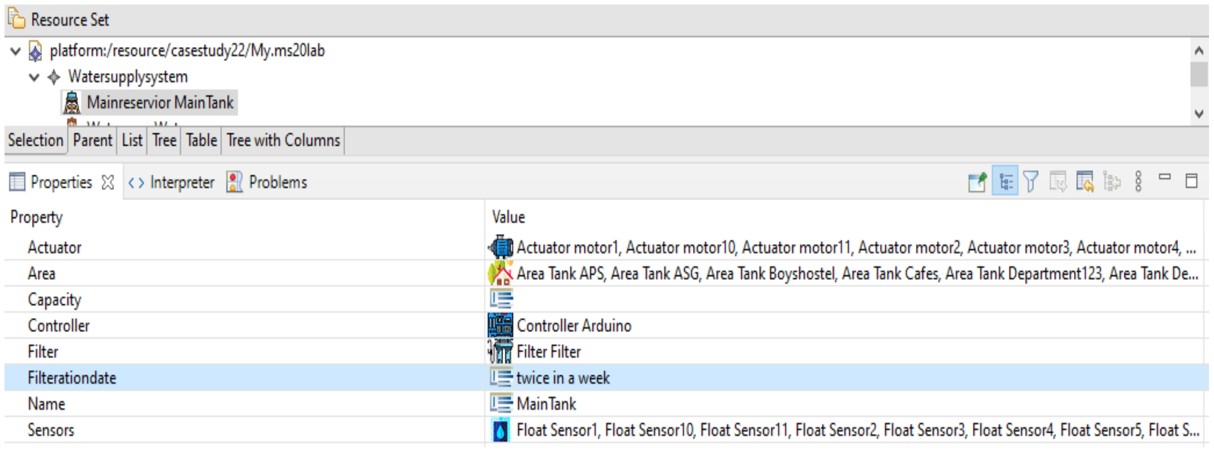
beginning, we define the metamodel, encompassing the key ideas of academic and administrative buildings, engineering departments, hostels, school, mosques, libraries, auditoriums, and the primary overhead water reservoir. The interconnected elements of the actual college campus are represented by instances of the metamodel that we create.
The college model's hierarchical structure will be represented by the tree view as shown in Figure 18. It is easy to examine and manage the many components of the college campus model because Eclipse's tree view gives users a structured and organized perspective of the case study of college.



```
platform:/resource/casestudy22/My.ms20lab
  Watersupplysystem
    Mainreservior MainTank
    Waterman Waterman
    Device application
    Wifi Module WifiModule
    Controller Arduino
    Actuator motor1
    Actuator motor2
    Actuator motor3
    Actuator motor4
    Actuator motor5
    Actuator motor6
    Actuator motor7
    Actuator motor8
    Actuator motor9
    Actuator motor10
    Actuator motor11
    Area Tank Girlshostel
    Area Tank Boyshostel
    Area Tank Department123
    Area Tank Department456
    Area Tank APS
    Area Tank Cafes
    Area Tank HQ
    Area Tank ResidenceTank1
    Area Tank ResidenceTank2
```

Figure 18. Treeview of casestudy2

The main reservoir is shown as a prominent node in Figure 19. The main reservoir is surrounded by interconnected components like actuators, sensors, filters, Arduino, and the area they serve. Each element is shown as a separate and easily recognizable symbol, facilitating identification. Any element may be selected by clicking it, and a brief properties window with all the necessary details about it will then show.


Resource Set
platform:/resource/casestudy22/My.ms20lab
Watersupplysystem
Mainreservior MainTank

Selection | Parent | List | Tree | Table | Tree with Columns

Properties | Interpreter | Problems

Property	Value
Actuator	Actuator motor1, Actuator motor10, Actuator motor11, Actuator motor2, Actuator motor3, Actuator motor4, ...
Area	Area Tank APS, Area Tank ASG, Area Tank Boyshostel, Area Tank Cafes, Area Tank Department123, Area Tank De...
Capacity	
Controller	Controller Arduino
Filter	Filter Filter
Filterationdate	twice in a week
Name	MainTank
Sensors	Float Sensor1, Float Sensor10, Float Sensor11, Float Sensor2, Float Sensor3, Float Sensor4, Float Sensor5, Float S...

Figure 19.Properties of Main tank

The graphical representation in Case Study 2 Sirius editor clearly illustrates all the nodes and edges, effectively visualizing the complete associated system. The aspects of the case study and their relationships are shown in the diagram in a clear and comprehensive approach. These nodes might be the main overhead water reservoir, the Army Public School, the Central Mosque, the College Central Library, the Auditorium, and the academic and administrative buildings. Each node has the proper label to enable quick identification of its identity.

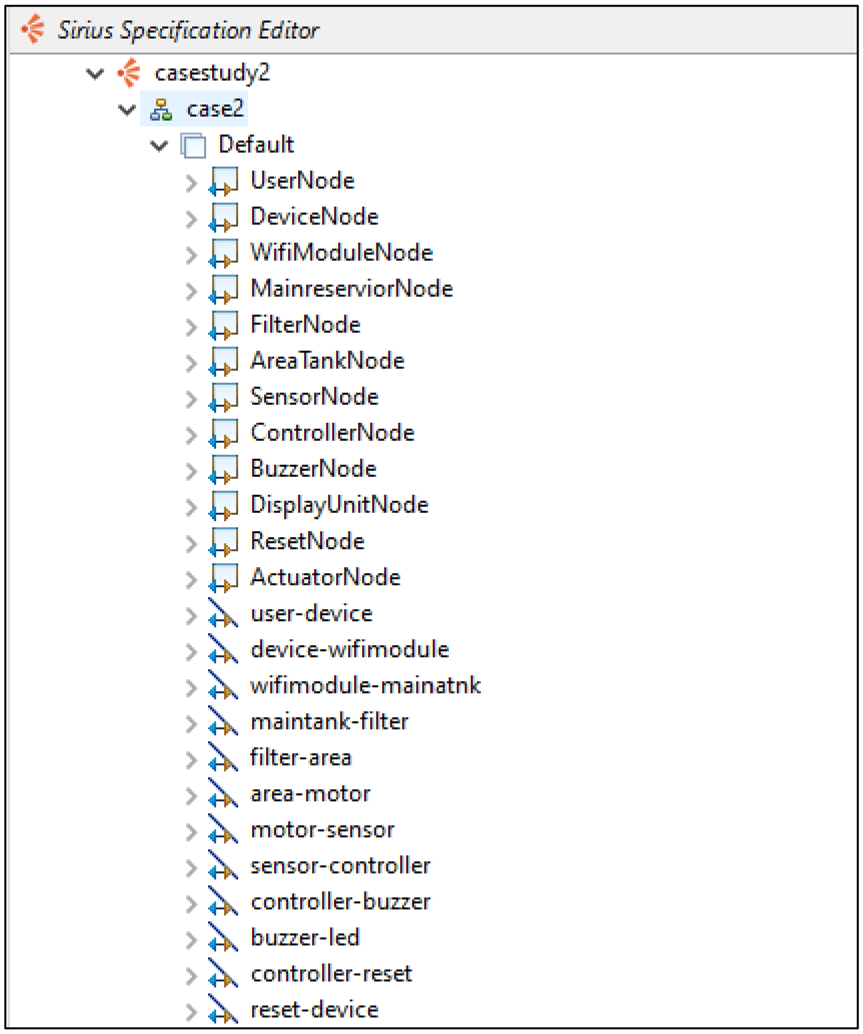

Sirius Specification Editor
casestudy2
case2
Default
UserNode
DeviceNode
WifiModuleNode
MainreserviorNode
FilterNode
AreaTankNode
SensorNode
ControllerNode
BuzzerNode
DisplayUnitNode
ResetNode
ActuatorNode
user-device
device-wifimodule
wifimodule-mainatnk
maintank-filter
filter-area
area-motor
motor-sensor
sensor-controller
controller-buzzer
buzzer-led
controller-reset
reset-device

Figure 20. Edges and Nodes in Designview

The water delivery system is clearly and precisely depicted in the graphical form of this case study as shown in Figure 21. A command is seen being given by Waterman, who is represented by a node, to the Wi-Fi module, which is another node, telling it to activate Sector 1's Motor 1. This causes Sector 1 to get water from the main tank via the filtration facility because of this action. Additionally, the graphical depiction illustrates the relationship between Motor 10 and the mosque, showing how water is provided to the mosque when Motor 10 is turned on. The mosque tank is connected to a sensor, which is shown as a node. The sensor keeps track of the water level and interacts with an Arduino, which is represented as another node. The Arduino is linked to an LED, which shows if water was successfully delivered to the mosque and how much was delivered. It also acts as a warning sign for any possible water leaks. A message certifying the water's safe delivery and the amount of water provided are displayed on the LED when water delivery is accomplished. Additionally, when the water distribution procedure is successful, a buzzer is engaged and makes a sound. If the LED does not display the message confirming a successful water supply, there may be a leak in the pipes. This crucial feature of finding water leaks gives the system an additional layer of monitoring and control, guaranteeing that any potential problems are quickly found and resolved. By enabling users to monitor and maintain an effective and dependable water supply system, this graphical representation ensures uninterrupted water delivery and reduces loss from leaks.

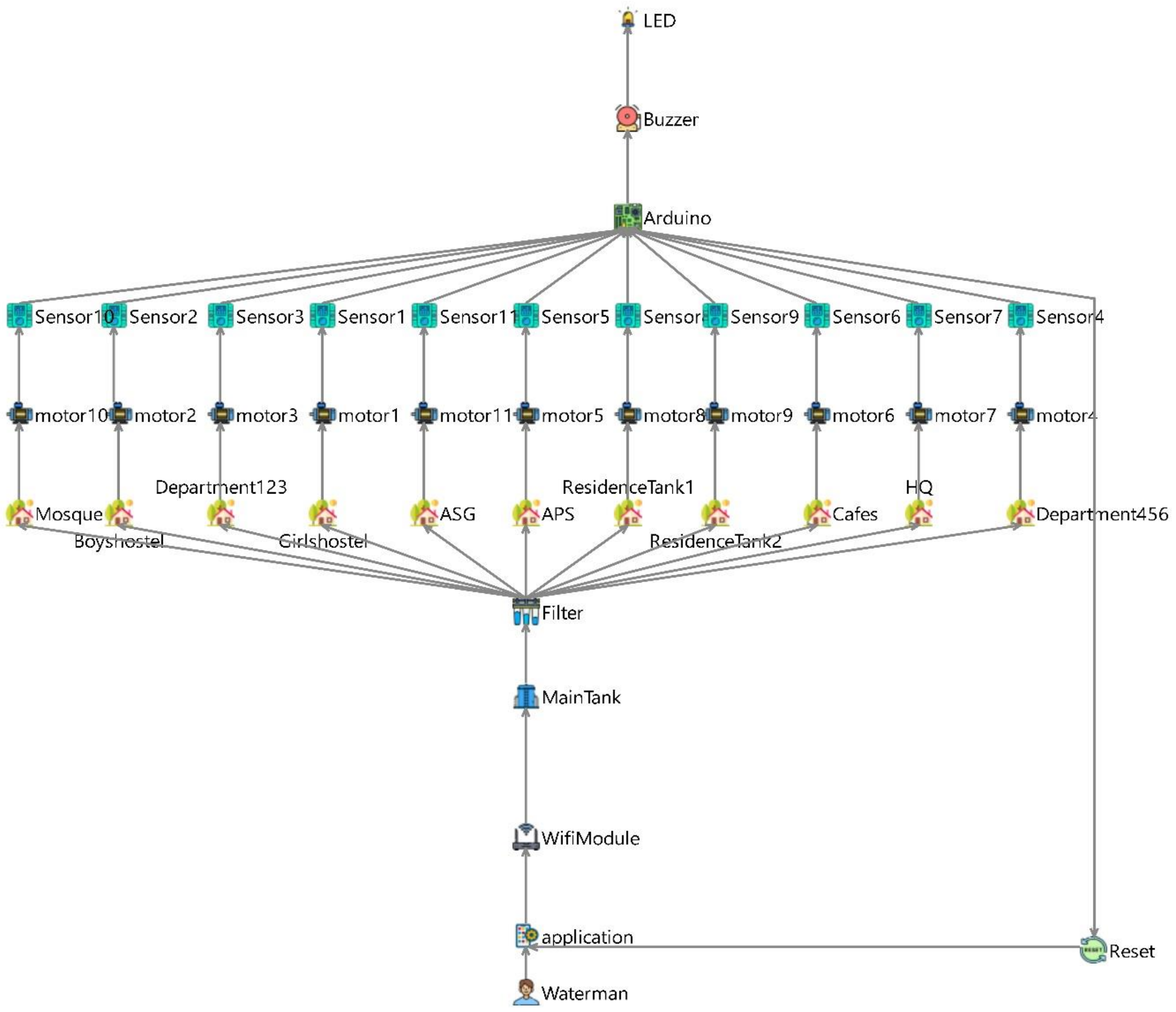


Figure 21. Graphical Representation in Sirius (CASESTUDY2)

4.3 Case study 3(Ahmadpur village)

Based on variables including population, agricultural area, and water accessibility, the hamlet of Chilka, which is situated in Ahmadpur Tehsil of Latur district, Maharashtra, was chosen for this research. It has a population of 875 people, divided among 203 households, and a land area of 571 hectares (Indian Census of 2011).

The inhabitants' main source of income comes from agriculture, which includes livestock husbandry, agricultural labor, and cultivation. Due to the consistently dry weather in the area, both home and agricultural needs for groundwater are primarily reliant on boreholes and drilled wells. Despite the existence of the nine drilled wells and four borewells seen in Figure 22, poor management of water resources has caused a serious water crisis in Chilka. An effective water management and distribution system is urgently needed to address the issue of water shortage. Even though there is physical water available, poor management is the main source of the issue. However, if water resources are handled well, they can produce noticeably better outcomes. Designing such a system requires considering the estimated lifespan of the infrastructure and estimating the village's future water needs. For the benefit of the Chilka villager community, a sustainable and dependable water supply may be secured by taking these elements into consideration and putting into practice good water management practices[58].

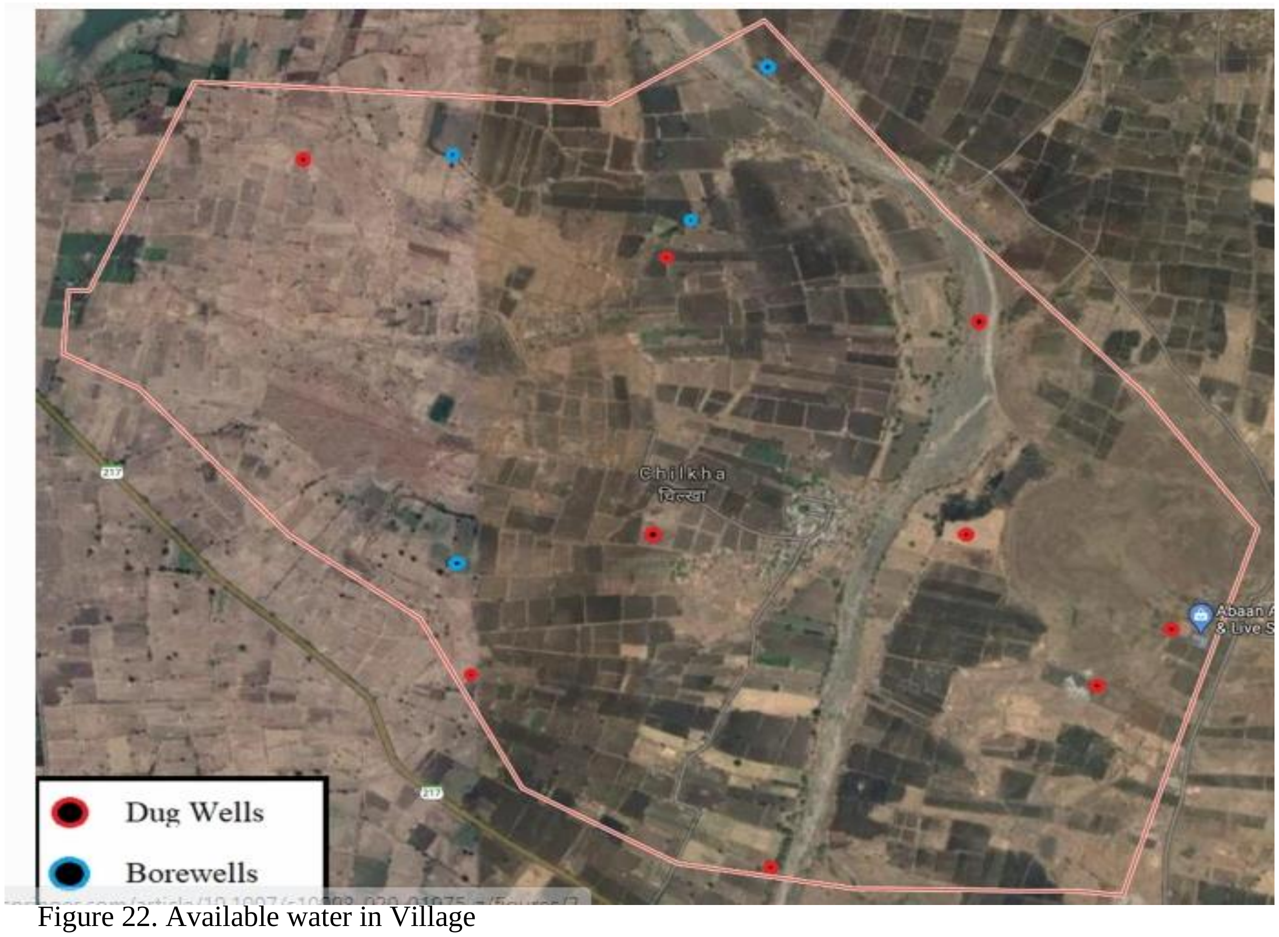
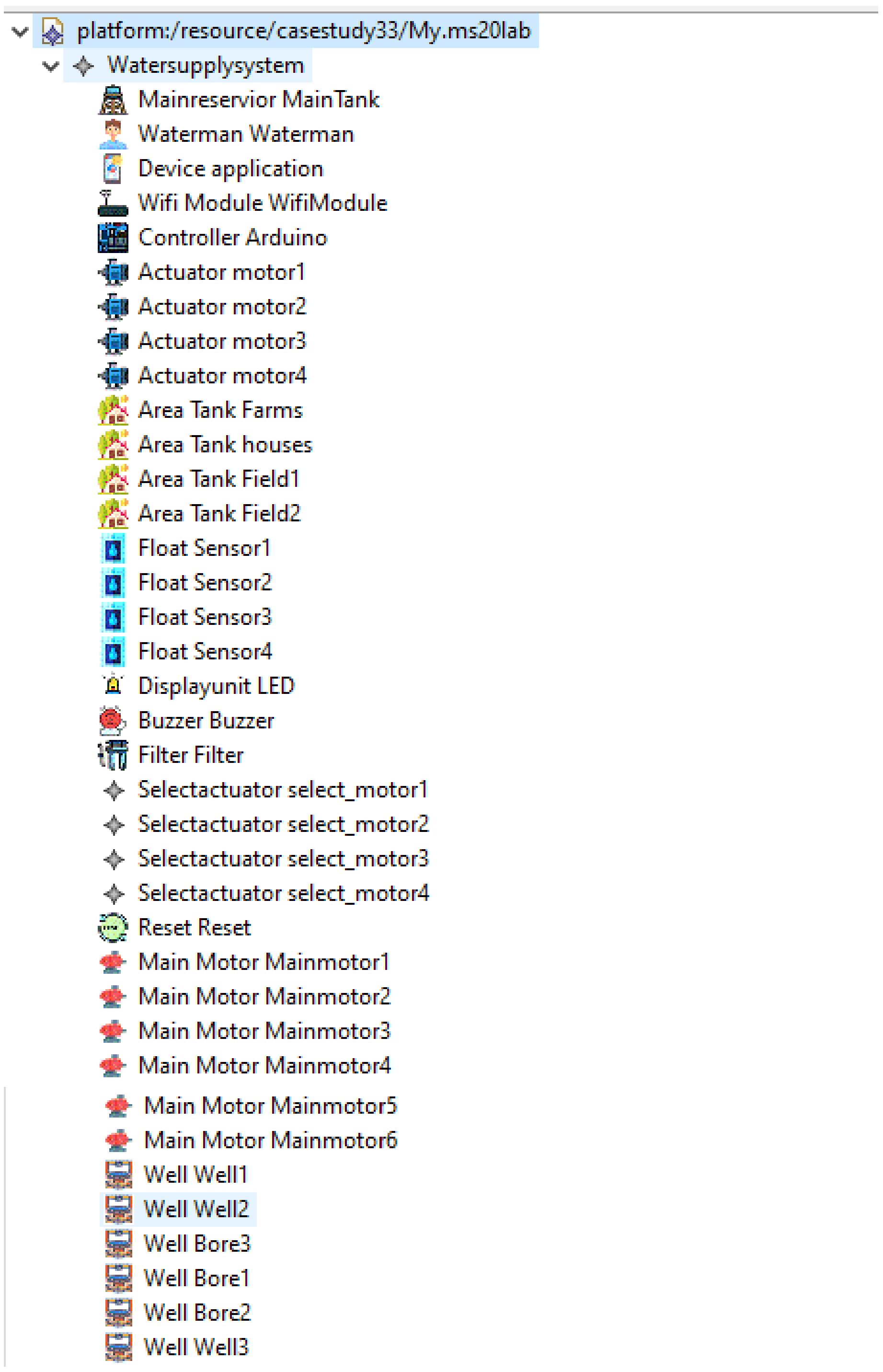


Figure 22. Available water in Village

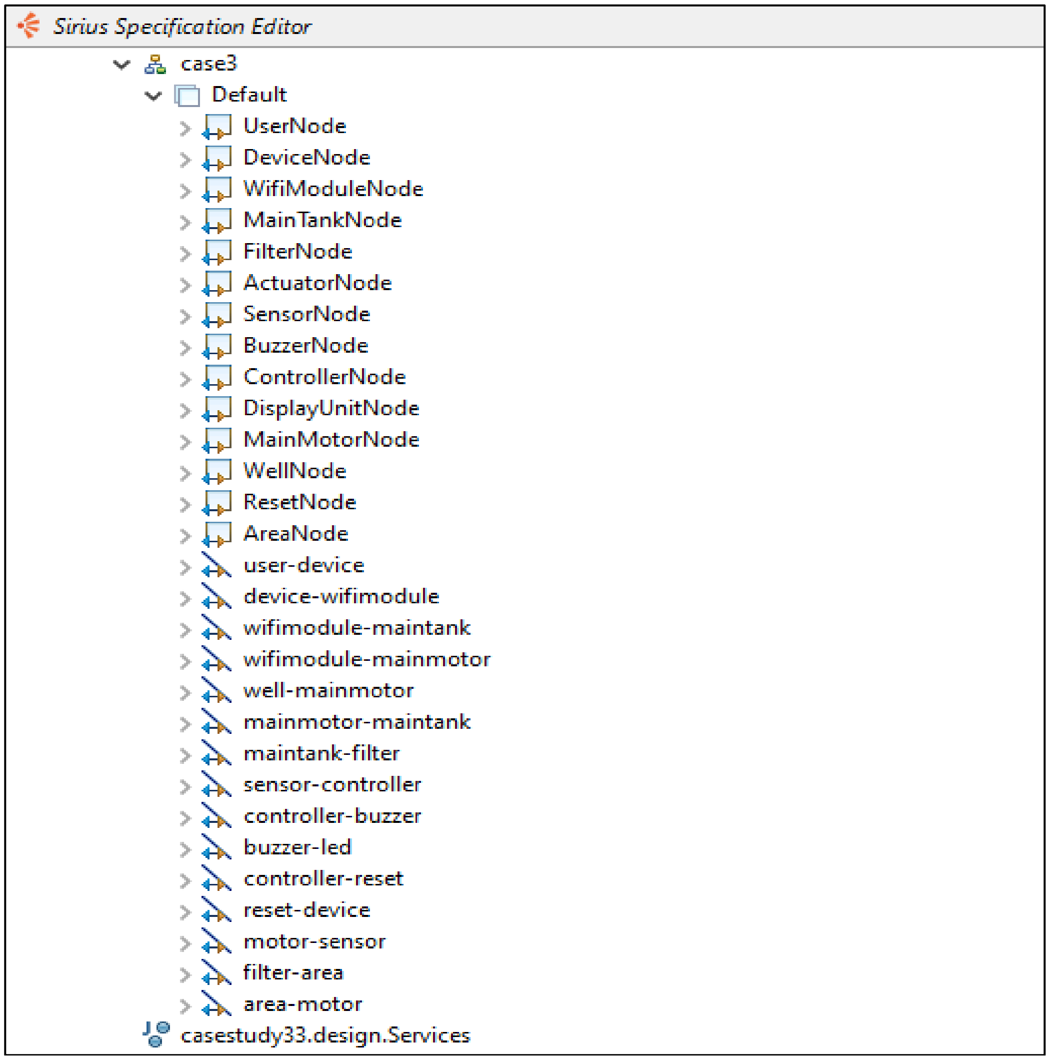
We may include the additional features of the main motor that moves water from the well or bore to the main tank as shown in Figure 23. We include the main motor as a new node under the "Water Supply System" category in this expanded tree view to show its function in moving water from wells or borewells to the main tank. Water distribution to residential and agricultural regions is the duty of the "Main Tank" node.



platform:/resource/casestudy33/My.ms20lab
- Watersupplysystem
  - Mainreservior MainTank
  - Waterman Waterman
  - Device application
  - Wifi Module WifiModule
  - Controller Arduino
  - Actuator motor1
  - Actuator motor2
  - Actuator motor3
  - Actuator motor4
  - Area Tank Farms
  - Area Tank houses
  - Area Tank Field1
  - Area Tank Field2
  - Float Sensor1
  - Float Sensor2
  - Float Sensor3
  - Float Sensor4
  - Displayunit LED
  - Buzzer Buzzer
  - Filter Filter
  - Selectactuator select_motor1
  - Selectactuator select_motor2
  - Selectactuator select_motor3
  - Selectactuator select_motor4
  - Reset Reset
  - Main Motor Mainmotor1
  - Main Motor Mainmotor2
  - Main Motor Mainmotor3
  - Main Motor Mainmotor4
  - Main Motor Mainmotor5
  - Main Motor Mainmotor6
  - Well Well1
  - Well Well2
  - Well Bore3
  - Well Bore1
  - Well Bore2
  - Well Well3

Figure 23. Treeview of casestudy3

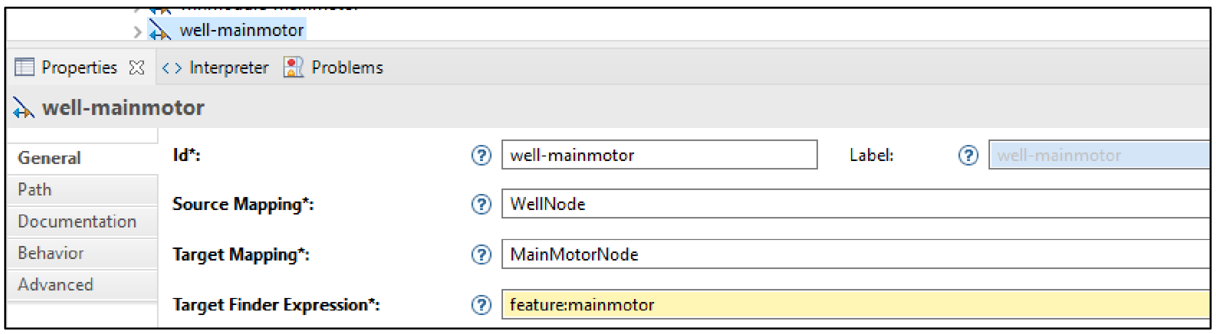
By adding additional nodes and edges in the Sirius editor, we can produce an enhanced graphical representation that highlights the main motors, bore water sources, and wells, as well as their connections. The association between the main motor and the water sources is shown graphically by an edge joining the "Main Motor" node to the "Drilled Wells" and "Borewells" nodes. An edge shows how the water flows from the "Main Motor" node to the "Main Tank," illustrating the water flow from the well to the tank as shown in Figure 24 and the connection between the well and main motor is shown in Figure 25.



Sirius Specification Editor

```
case3
  Default
    UserNode
    DeviceNode
    WifiModuleNode
    MainTankNode
    FilterNode
    ActuatorNode
    SensorNode
    BuzzerNode
    ControllerNode
    DisplayUnitNode
    MainMotorNode
    WellNode
    ResetNode
    AreaNode
    user-device
    device-wifimodule
    wifimodule-maintank
    wifimodule-mainmotor
    well-mainmotor
    mainmotor-maintank
    maintank-filter
    sensor-controller
    controller-buzzer
    buzzer-led
    controller-reset
    reset-device
    motor-sensor
    filter-area
    area-motor
casestudy33.design.Services
```

Figure 24. Design view of casestudy3



well-mainmotor

well-mainmotor

Id*:	well-mainmotor	Label:	well-mainmotor
Source Mapping*:	WellNode		
Target Mapping*:	MainMotorNode		
Target Finder Expression*:	feature:mainmotor		

Figure 25. Edge properties of Well

The community of Chilka has a serious problem because of inadequate water management, despite having access to plenty of water resources. This poor management has had a serious impact on the region's agricultural activity by causing a severe problem with the provision of water to farms and fields. The lives of the community members who significantly depend on farming as their main source of income are hampered by the inability to efficiently harness and distribute the available water resources, which has created a barrier to agricultural output.

We provide a comprehensive overview of the complete procedure, considering the additional features of well and bore water, in the graphical representation of the Chilka Hamlet water supply system using Sirius. The diagram illustrates the methodical steps that the waterman took to efficiently manage the water supply. The water sources, which consist of borewells and drilled wells, are the system's essential components. These resources are essential for supplying the village with the water it requires. The water distribution system has main motors that move water from wells and borewells to the main tank. The main tank acts as the village's central reservoir, holding water for later distribution to homes, farms, and fields. The waterman communicates with the system using a device that serves as an interface by sending commands to manage the water supply. These commands are sent to the WiFi module, which then connects with the main motors to start filling the main tank to capacity. When the main tank is full, the waterman sends further instructions to turn on individual motors for farms and houses. These motors make sure that the main tank's water supply is effectively delivered to the fields and residential areas. The sensors systematically monitor the water levels in the local tanks. An Arduino system is integrated into the design to monitor the water supply and identify any problems. The Arduino uses an LED display to convey messages confirming effective water delivery to the fields or warning about potential leaks or water theft after receiving signals from the sensor. The LED display is triggered by the Arduino to show a message stating that the farms have received all the water they need. To further indicate that the water supply was successful, the Arduino simultaneously turns on the buzzer, which emits a unique beep sound as shown in Figure 26.

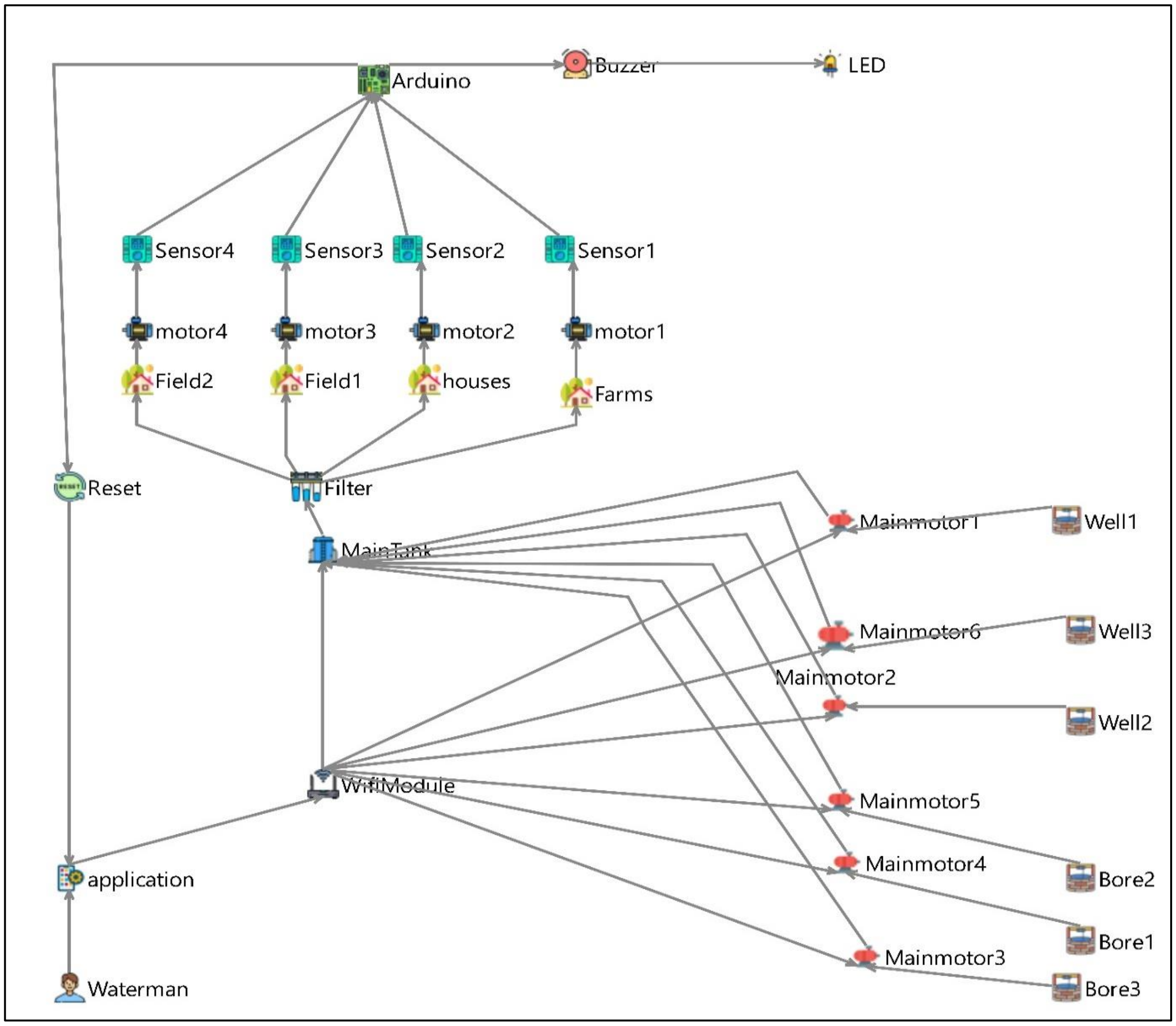


Figure 26. Graphical representation in Sirius of casestudy3

.3. 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.

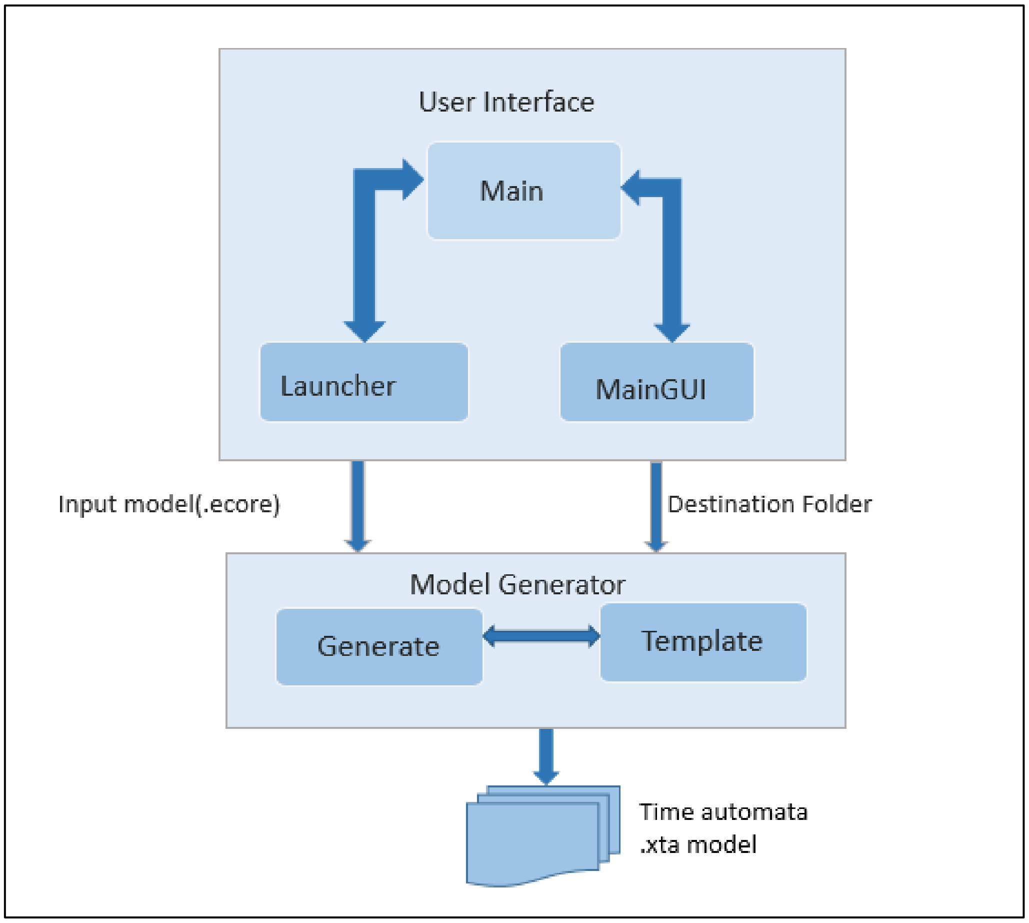


Figure 27. The architecture of Transformation engine

The Ecore metamodel is transformed by the Model Generator into Timed Automata 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 Timed Automata representation for the specified Ecore model from the user interface.

Template File (Generate.mtl): 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 Timed Automata constructs.

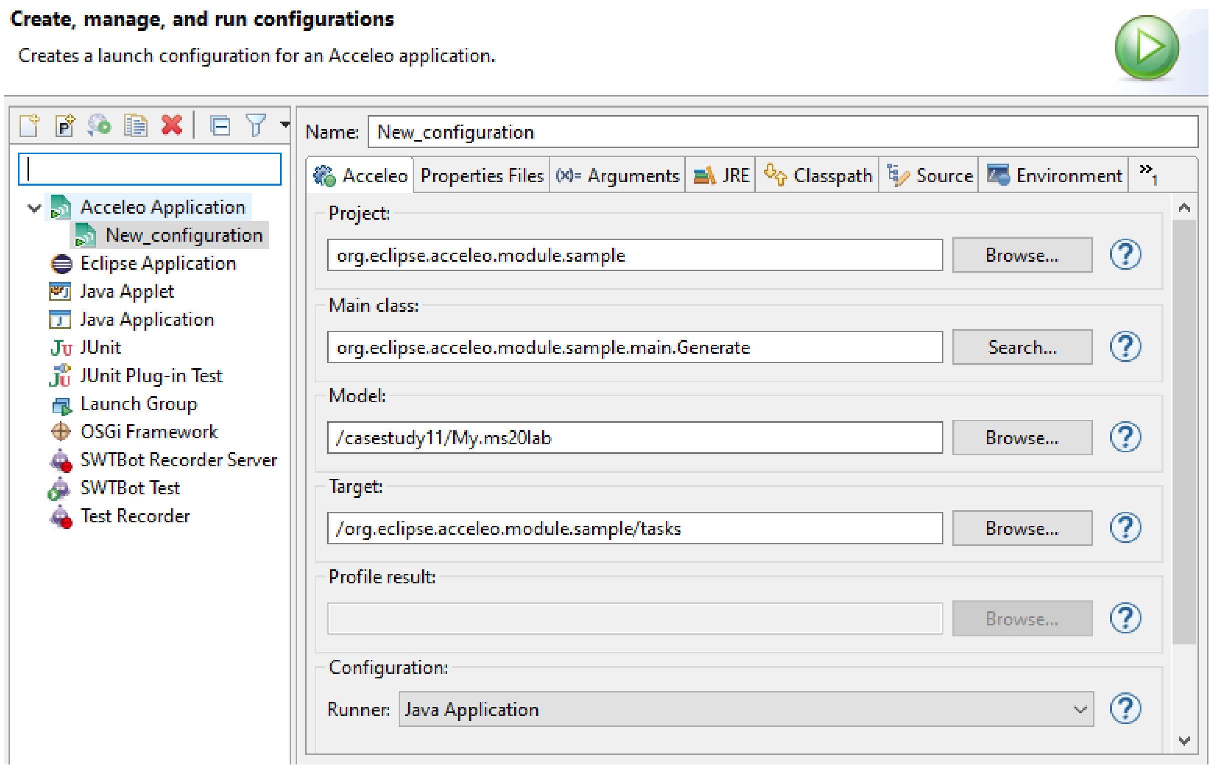
Two output files are produced after the Generate.java and related template file have been successfully executed:

a. xta this file contains the modified Timed Automata model, which simulates the system behavior shown in the ecore model.

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

A model of the Water supply management system 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.



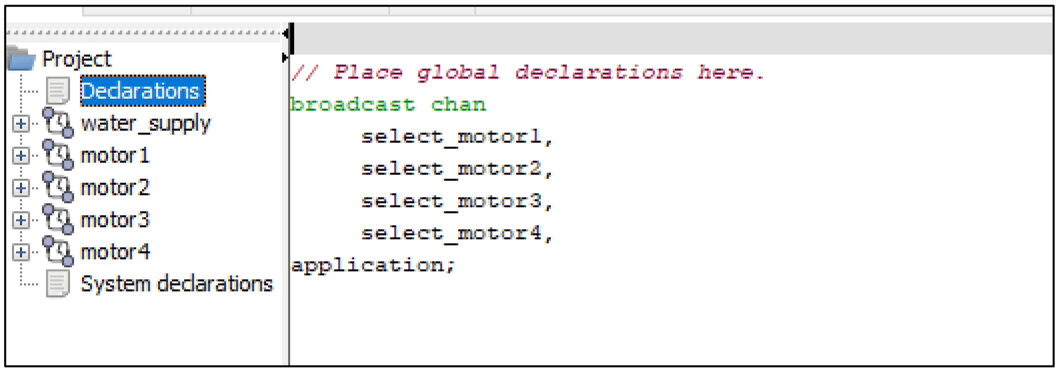
Name: New_configuration
Project: org.eclipse.acceleo.module.sample
Main class: org.eclipse.acceleo.module.sample.main.Generate
Model: /casestudy11/My.ms20lab
Target: /org.eclipse.acceleo.module.sample/tasks
Profile result:
Configuration:
Runner: Java Application

Figure 28. Run configuration of Acceleo

UPPAAL:
For formal verification and validation of the system, the generated. Xta and .q files can then be utilized using the UPPAAL tool or utility. UPPAAL helps to guarantee that the system performs appropriately regarding the modeled ecore design by determining whether the system meets the defined properties. The system design and execution are made more dependable and accurate through this procedure[61].
A strong tool for modeling and testing real-time systems with timed automata is UPPAAL. An xta file containing a UPPAAL template can be divided into the following main components:

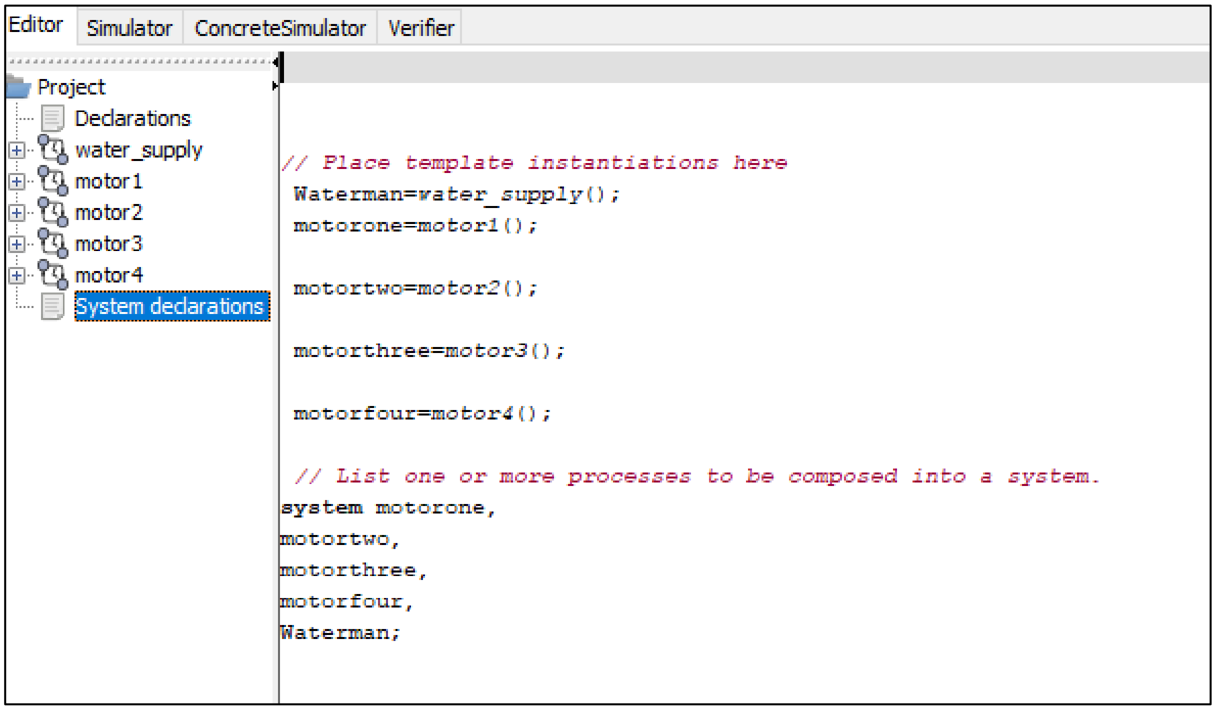
1. Global Declarations
2. Parameters
3. Local Declarations
4. Locations
5. Edges

6. System Definition



```
// Place global declarations here.
broadcast chan
      select_motor1,
      select_motor2,
      select_motor3,
      select_motor4,
application;
```

Figure 29. Global declaration



```
// Place template instantiations here
 Waterman=water_supply();
 motorone=motor1();

 motortwo=motor2();

 motorthree=motor3();

 motorfour=motor4();

 // List one or more processes to be composed into a system.
system motorone,
motortwo,
motorthree,
motorfour,
Waterman;
```

Figure 30. System declaration

CHAPTER 5

VALIDATION

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

5.1. Case Study 1 Validation:

A model checker for real-time systems that is popular in both research and business is called the UPPAAL tool. UPPAAL is a very effective tool for studying the behavior of real-time systems because it can model and test systems that contain both logical and temporal restrictions. In this study, UPPAAL is used to convert a system's metamodel into a formal verification model, which can be used to confirm that the behavior of the system is right. According to the transformation rules in Table 1, the metamodel uses classes, objects, and relationships. While UPPAAL uses templates, locations, and edges to represent components. In contrast to UPPAAL, which uses global and local variables to create properties, the metamodel uses attributes and methods to define properties.

In UPPAAL's transitions, constraints from the metamodel are transformed into guard conditions. While UPPAAL automates verification through model checking and simulation, the metamodel requires manual testing and debugging. In contrast to UPPAAL, which employs the invariant attribute to limit object relationships within a location, multiplicity constraints in the metamodel specify object associations. The metamodel is eventually saved as source code files, and UPPAAL outputs the model as a .xta file for formal verification and simulation. A .xta file is produced as the output following the transformation of the MDA-created water supply management model using Acceleo. This text-based file, which can be loaded into the UPPAAL model checker, represents the model. We may import the .xta file into UPPAAL once we have it to see the Water Supply Management System model as a graphical representation.

Five interconnected models make up the Water Supply Management System model, including a primary model named WaterSupply and four models for each motor, referred to as Motor1, Motor2, Motor3, and Motor4 as shown in Figure 32. The WaterSupply model, which starts at the Waterman location, illustrates the system's primary control flow. Location motors represent the four motors that supply water to different sectors of the city. The select_motor1! Sync condition at the MotorOne location activates the Motor1 Model control logic and synchronizes the system with the Motor1 model. Once the Motor1 model has arrived at its final location, it uses the application! Sync condition to come back to the main WaterSupply model and the same for the rest of the motor's model. UPPAAL the! And? are used to send and receive messages. The main components of the UPPAAL template for UPPAAL Properties (.q file) are mentioned below.

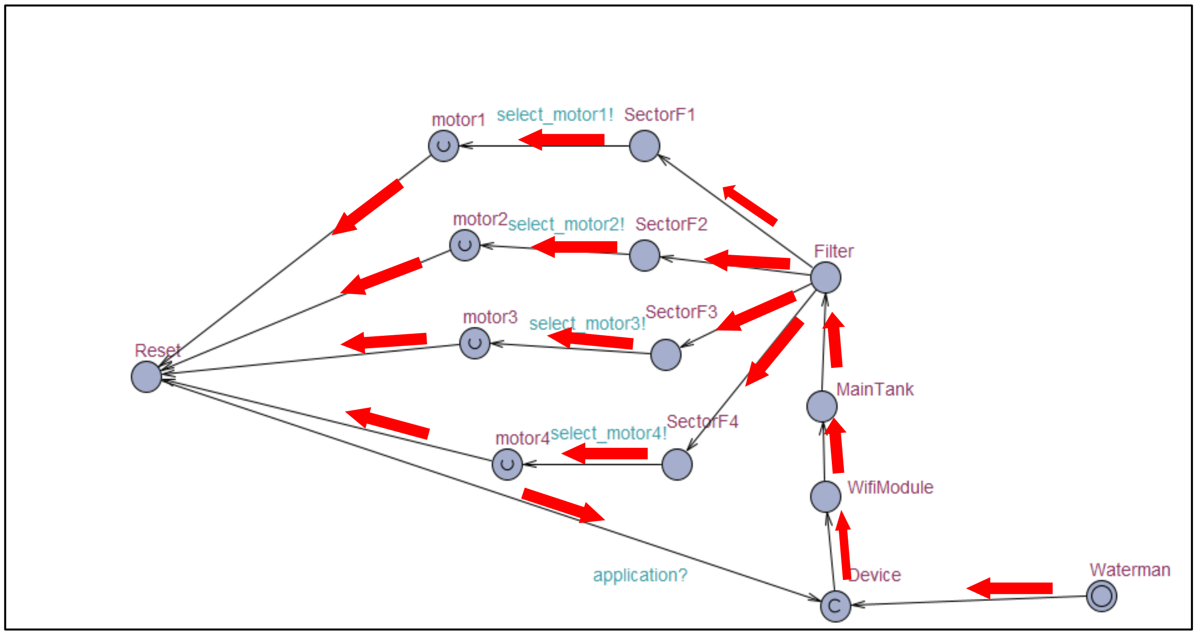


Figure 32. Main supply of case study1

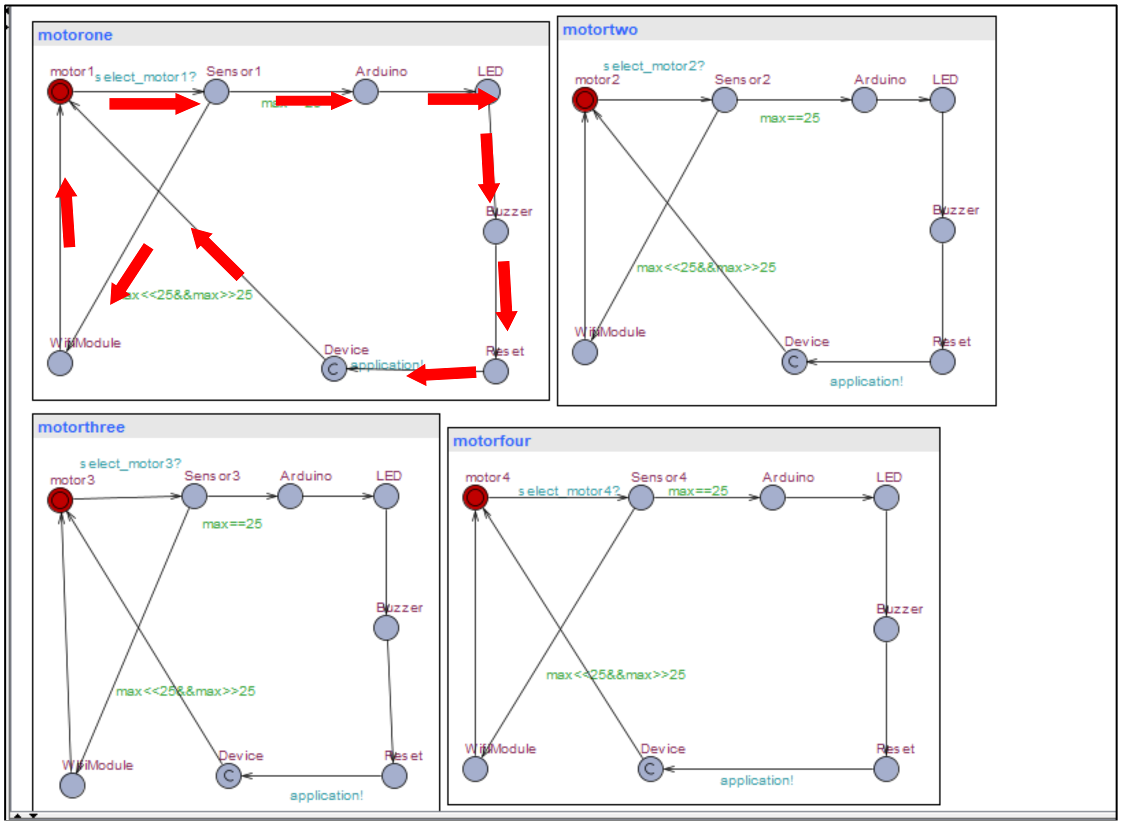
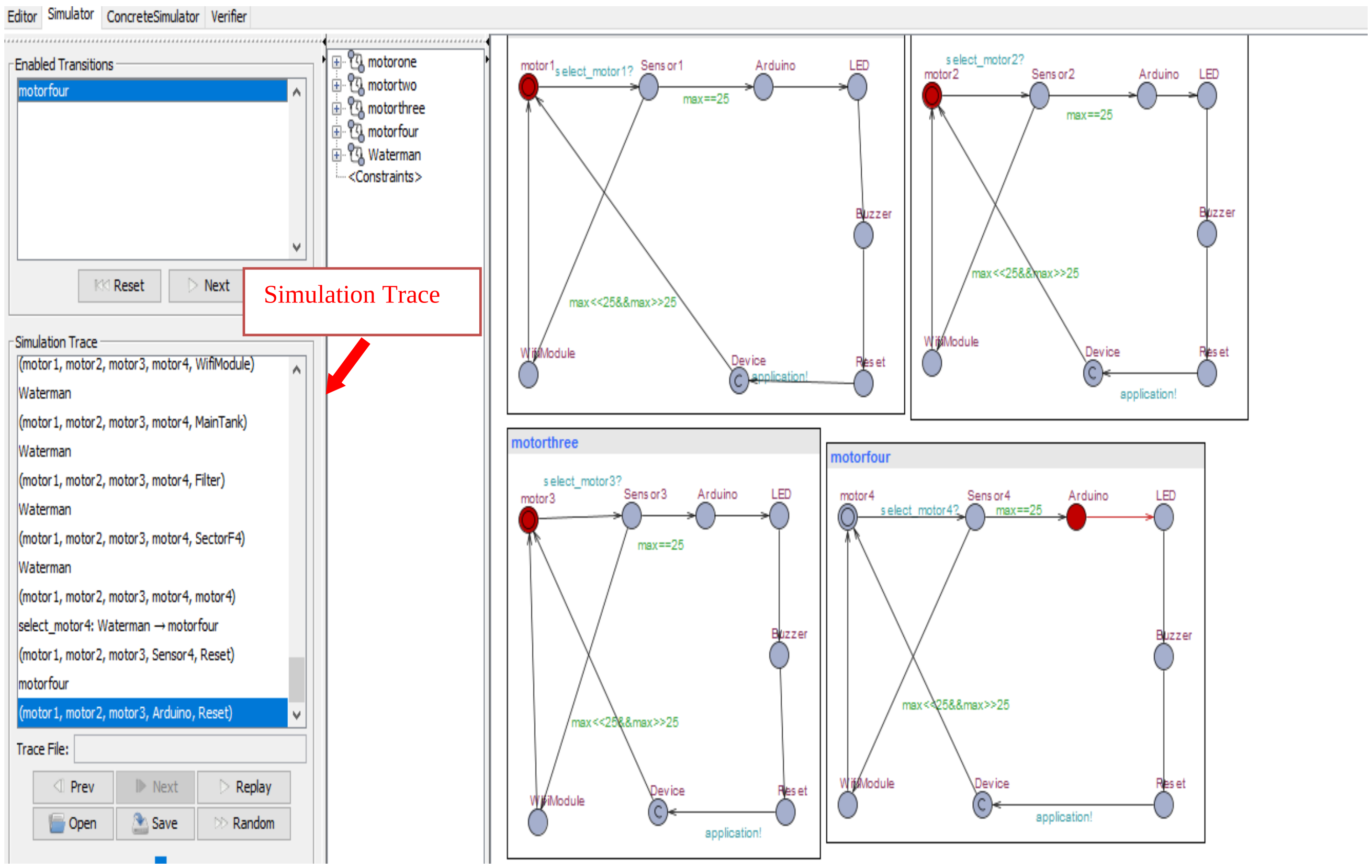


Figure 33. Motors of casestudy1



Enabled Transitions:

```
motorfour
```

Simulation Trace:

```
(motor1, motor2, motor3, motor4, WifiModule)
Waterman
(motor1, motor2, motor3, motor4, MainTank)
Waterman
(motor1, motor2, motor3, motor4, Filter)
Waterman
(motor1, motor2, motor3, motor4, SectorF4)
Waterman
(motor1, motor2, motor3, motor4, motor4)
select_motor4: Waterman → motorfour
(motor1, motor2, motor3, Sensor4, Reset)
motorfour
(motor1, motor2, motor3, Arduino, Reset)
```

Figure 34. Simulator of UPPAAL

A timed automaton model's execution route during a simulation run is referred to as a simulation trace in UPPAAL. When you simulate a model in UPPAAL, the tool explores several execution routes and creates a trace that displays the system's series of states and transitions in response to events and time elapsed.

The word "urgent" is used in UPPAAL to describe transitions that are given a greater priority than other transitions. When a transition is designated as "urgent," it signifies that it can take precedence over any non-urgent transitions and that it will always be executed as soon as it is enabled. Consider a real-time system where pressing the emergency stop button takes precedence over doing other things. In this situation, you would represent the transition corresponding to hitting the emergency stop button as "urgent." As a result, anytime the button is touched, it immediately takes precedence over any other tasks that are already in progress, stopping the system instantly[62].

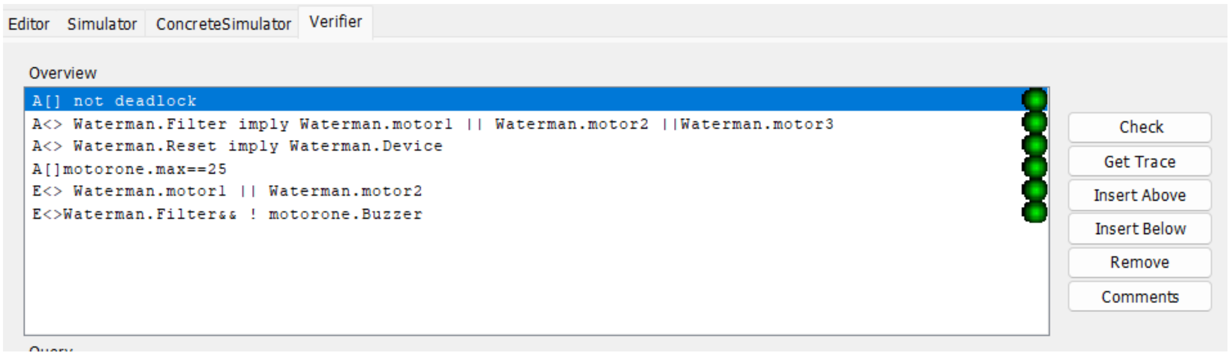
The word "committed" is used in UPPAAL to represent synchronization between several transitions that need to occur atomically. When a transition is designated as "committed," it signifies that for any of them to be carried out, they must all be enabled at once. Consider a system where a vehicle must open both the left and right doors at the same time for passengers to board. The transitions that correspond to opening the left and right doors in this scenario would be represented as "committed." This prevents any inconsistent behavior by ensuring that both doors are opened simultaneously or not at all.

Query: this portion of the template contains the UPPAAL properties. Properties should be defined as per the following syntax as shown in Table 2 [63].

Table 2. UPPAAL Properties

Sr no	**Property Type**	**Operator**	**Query type**
1	Possibly	E<>	Reachability
2	Invariantly	A[]	Safety
3	Potentially always	E[]	Safety
4	Eventually	A<>	Liveness
5	Leads to	→	Liveness

Queries types are classified as Reachability, Safety, Liveness, and deadlock. Reachability query tests if a certain state or group of states can be reached from the initial state of the model. Safety query determines if a specific attribute is always true for every conceivable system execution. It guarantees that the system never reaches an undesired state. The Liveness query determines if a certain attribute eventually holds for all possible system executions. It guarantees that the system never reaches an undesired state. The Liveness query determines if a certain attribute eventually holds for all possible system executions. It guarantees that the system eventually reaches a desired state [64].
Verification and validation of system designs should be done early in the Software Development Life Cycle (SDLC) since it removes mistakes and uncertainties with less cost, time, and resources. The UPPAAL model checker is used in this work to validate properties such as deadlock, security, liveness, and reachability as shown in Figure 35. The properties are validated after simulating the system behavior with UPPAAL. If the model checker returns a green signal, it indicates that the system is error-free and fits the requirements. The extensive validation procedure guarantees that the system works as it should and satisfies the standards for a safe and effective water supply management framework. The successful testing of qualities shows that the proposed water supply management works as expected. UPPAAL and its extensive model-checking capabilities can detect and correct any property violations, guaranteeing that the system operates predictably and consistently in real-world circumstances.



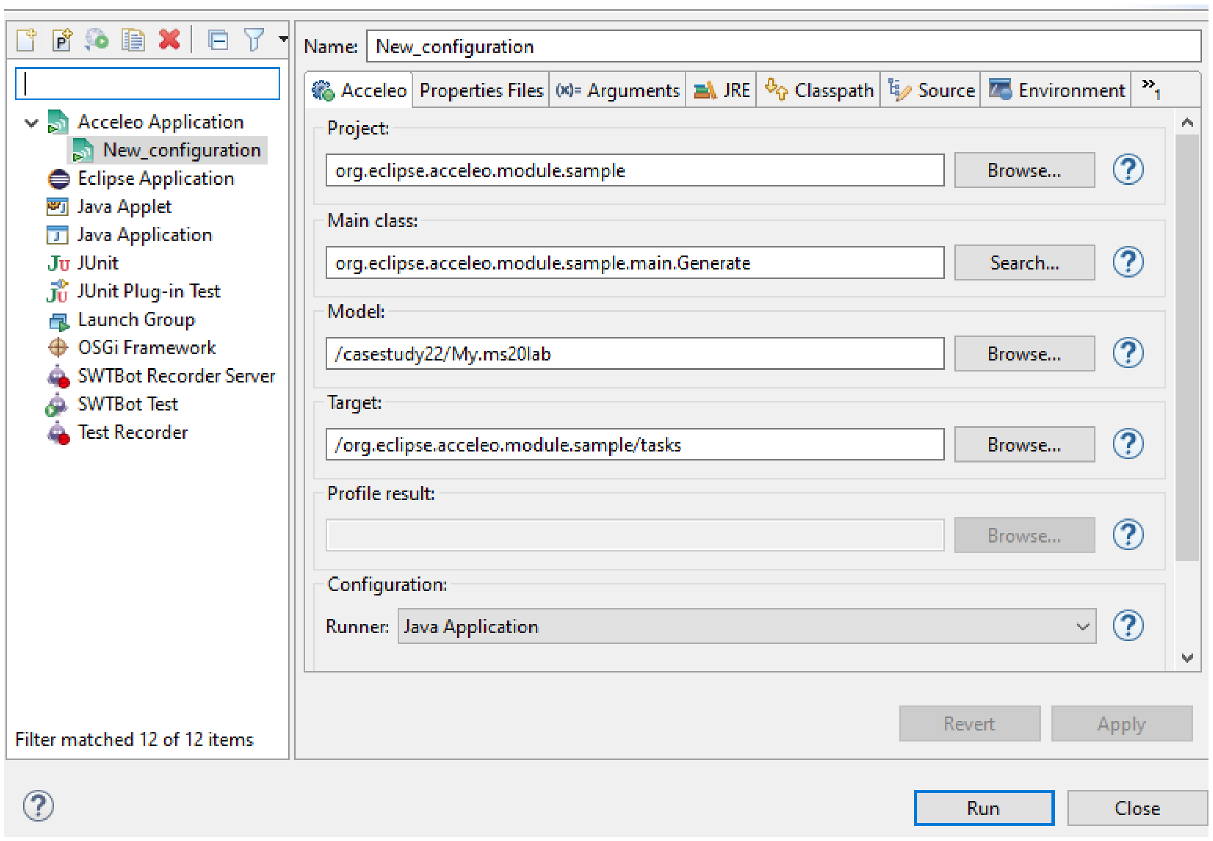
```
A[] not deadlock
A<> Waterman.Filter imply Waterman.motor1 || Waterman.motor2 ||Waterman.motor3
A<> Waterman.Reset imply Waterman.Device
A[]motorone.max==25
E<> Waterman.motor1 || Waterman.motor2
E<>Waterman.Filter&& ! motorone.Buzzer
```

Figure 35.Properties of case study 1

The property "A[] not deadlock" denotes that there won't be a deadlock in any of the system's potential future states. In other words, in every circumstance, the system is guaranteed to be free of deadlocks.
"A<>Waterman. Filter imply motorone.motor1 && motortwo.motor2 && motorthree.motor3 && motorfour.motor4 ".The expression on the left side implies the expression on the right, expressing a conditional connection. It indicates that the expression on the right side should likewise be true in this situation if "A>Waterman. Filter" is true. "A<>Waterman. Reset imply motorone.motor1"
"A[] motorone.max && motortwo.max && motorthree.max == 25". This property guarantees that the maximum value of the "motorthree" component is always equal to 25, and that the "motorone.max" and "motortwo.max" variables are always true (non-zero) in all states of the system. A violation of the given criterion would be shown if any of these requirements were to be violated at any point along any path of the system, rendering the property untrue.
"E<>Waterman.motor1 || Waterman.motor2".This property makes sure that at some point along at least one feasible path of the system, "Waterman.motor1" or "Waterman.motor2" (or both) will be true. The Boolean expression "Waterman.motor1" and "Waterman.motor2" describe an OR operation (logical disjunction) between the two propositions. It says that for the statement to be true, either "Waterman.motor1" or "Waterman.motor2" (or both) must be true.
"E<>Waterman. Filter &&! motorone. Buzzer".This property guarantees that the "Waterman. Filter" will become true at some point during the system's execution and that the "motorone. Buzzer" will also turn false at the same time. This may signify a certain desired behavior in which the Waterman filter is turned on while the motor one's buzzer is off.

5.2. Validation of case study 2:

We chose the Case Study 2 model as the source in the run configuration for Case Study 2 validation, and we selected 'Tasks' as the target to display our results. The "target" is the place or thing where the output or outcomes of the system will be seen or presented as shown in Figure 36.



Name: New_configuration

Project: org.eclipse.acceleo.module.sample

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

Model: /casestudy22/My.ms20lab

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

Profile result:

Configuration:

Runner: Java Application

Figure 36. Run configuration of case study 2

The water supply system in Case Study 2 runs efficiently and effectively, guaranteeing a steady flow of water to different areas. The main control model, "Water Supply," and ten separate motor models are the two crucial parts of this well-designed system as shown in Figure 37**.** The Waterman control device starts the process by delivering instructions to the motors via a Wi-Fi module as the system develops. The main supply model synchronizes with the appropriate motor model via synchronization signals marked by "!". Once synchronized, the motor model takes charge and effectively regulates water delivery to the target area. To maintain optimal performance, sensors are used to monitor water levels. When the water distribution is successful, LCDs show it, and a buzzer makes a beep sound.
The main supply model receives a synchronization signal (application!) from the motor model once the motor model has successfully completed a task. The main supply model can go on to the next motor and continue the procedure for continuous water delivery to all areas. The water supply system effectively meets the water demands in different areas through this well-coordinated synchronization and communication mechanism, offering a dependable and efficient water distribution process. We also carefully customize water distribution based on the unique requirements of each area; for example, hostels receive larger allocations than cafés or headquarters, ensuring the maximum water level perfectly corresponds with their demands.

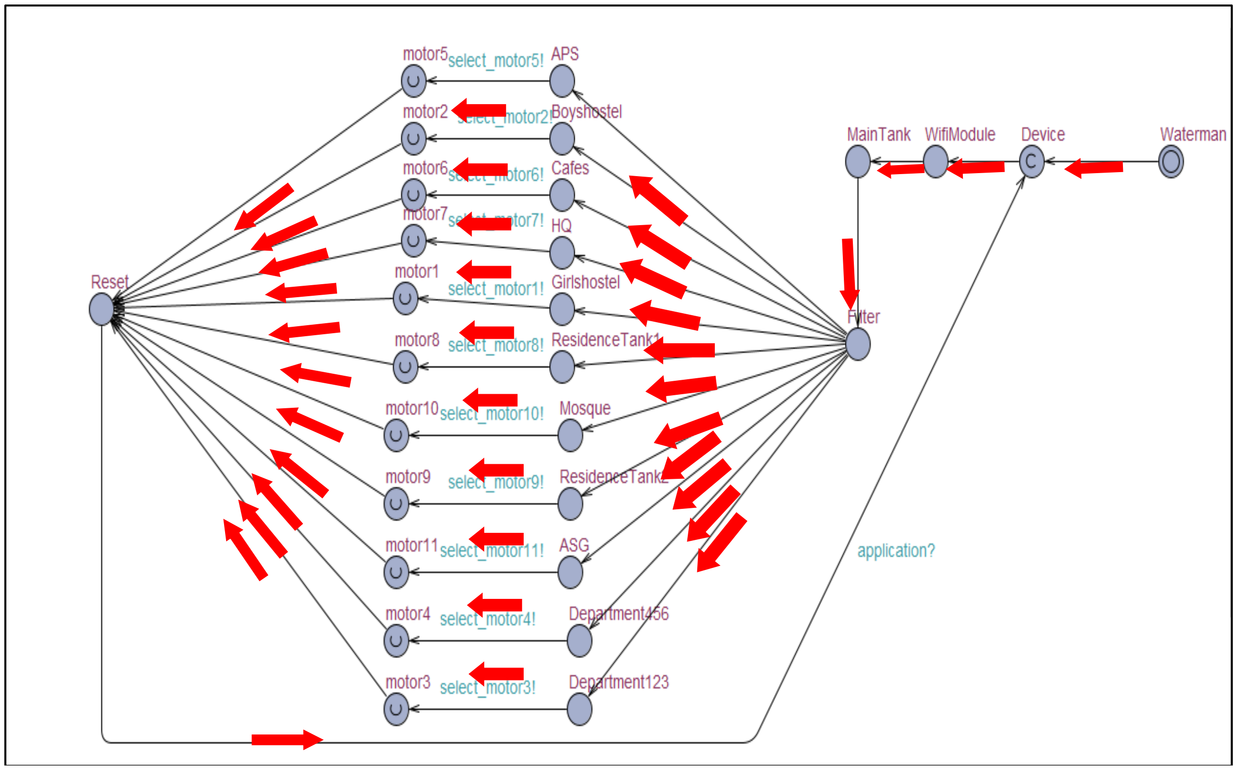


Figure 37. Main supply model of case study 2

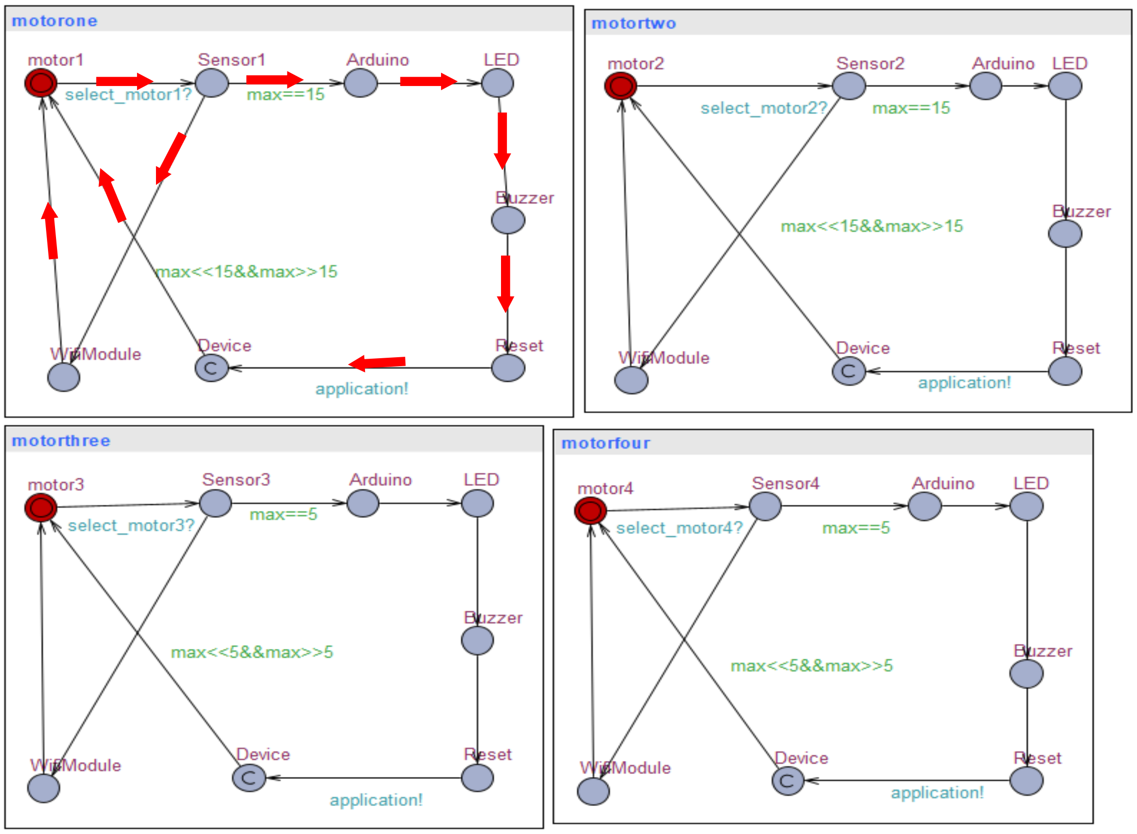


Figure 38. Motors model of case study 2

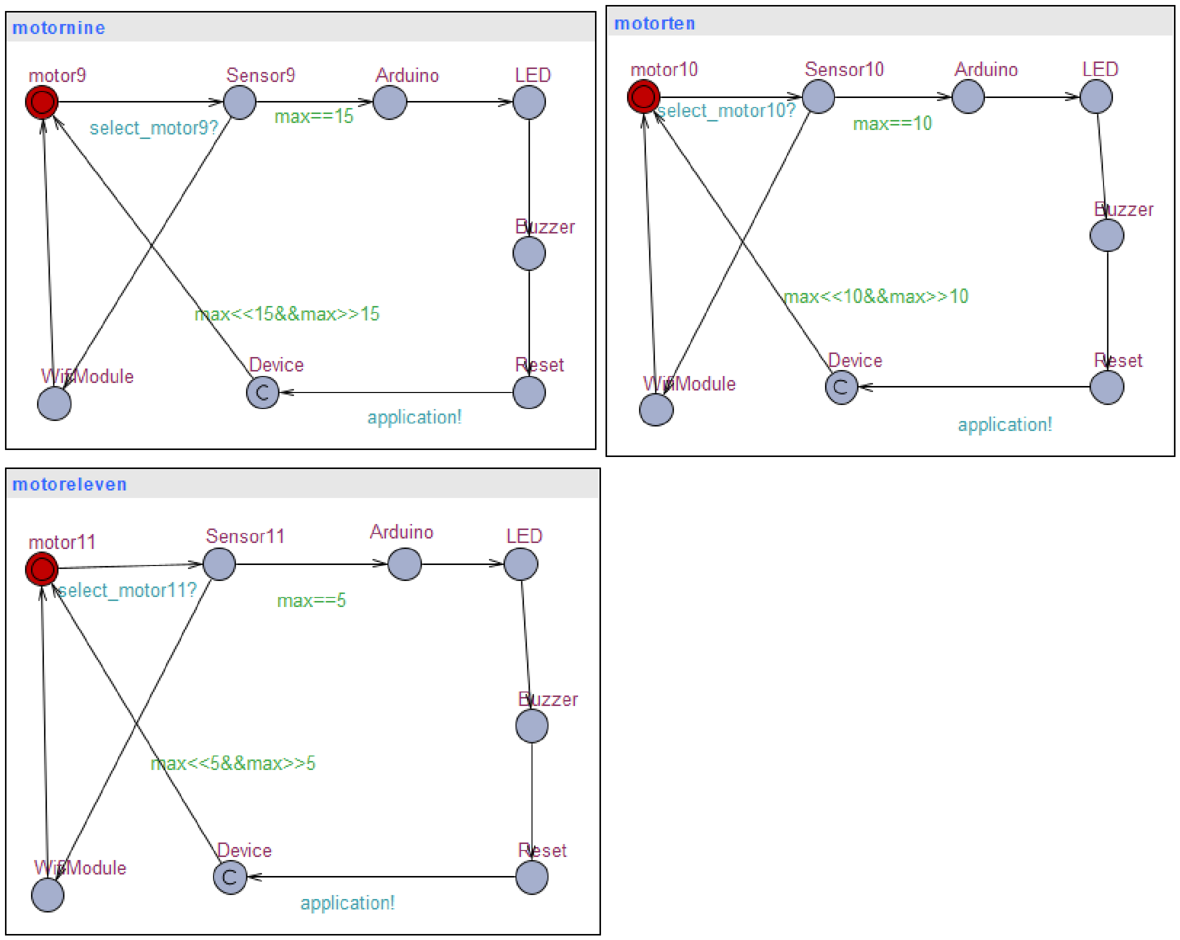


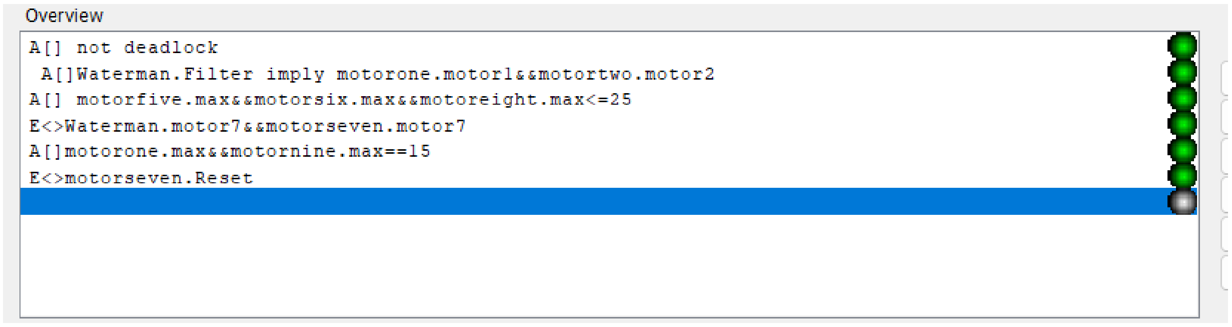
Figure 38. Motors model of case study 2

The next step is to validate the system's behavior and characteristics after running the model in accordance with the case study. Validation in the context of formal verification determines if the system complies with given requirements. These characteristics may include performance assurances, accuracy claims, safety criteria, and other system-specific specifications as shown in Figure 39.
The property "A[] not deadlock" guarantees that under every conceivable future circumstance, the water supply system will never experience a deadlock state. Processes become stuck in a deadlock when they are dependent on one another for resources, which might make the system unresponsive. The system's resilience is confirmed by validating this property, which also guarantees a constant water supply free from any blockages.

"A[]Waterman. Filter imply motorone.motor1&&motortwo.motor2 "it explains that if the 'Waterman. Filter' condition is satisfied, then it is assumed that the 'motorone. motor1' and'motortwo.motor2' conditions will also be true in all future states.

"A[] motorfive.max&&motorsix.max&&motoreight.max<=25".The maximum water level values for "motorfive," "motorsix," and "motoreight" for all upcoming states should be less than or equal to 25.

"E<>motorseven.Reset”. This property determines whether the "Reset" condition of "motorseven" will occur at some point in the future. “A[]motorone.max&&motornine.max==15".The maximum water level values for motor one and motor nine must be equal to 15


Overview
```
A[] not deadlock
 A[]Waterman.Filter imply motorone.motor1&motortwo.motor2
A[] motorfive.max&&motorsix.max&&motoreight.max<=25
E<>Waterman.motor7&&motorseven.motor7
A[]motorone.max&&motornine.max==15
E<>motorseven.Reset
```
Figure 39. Properties validation of case study 2

5.3. Validation of case study 3:

We chose the Case Study 3 model as the source in the run configuration for Case Study 3 validation, and we selected 'Tasks' as the target to display our results. The "target" is the place or thing where the output or outcomes of the system will be seen or presented as shown in Figure 40.

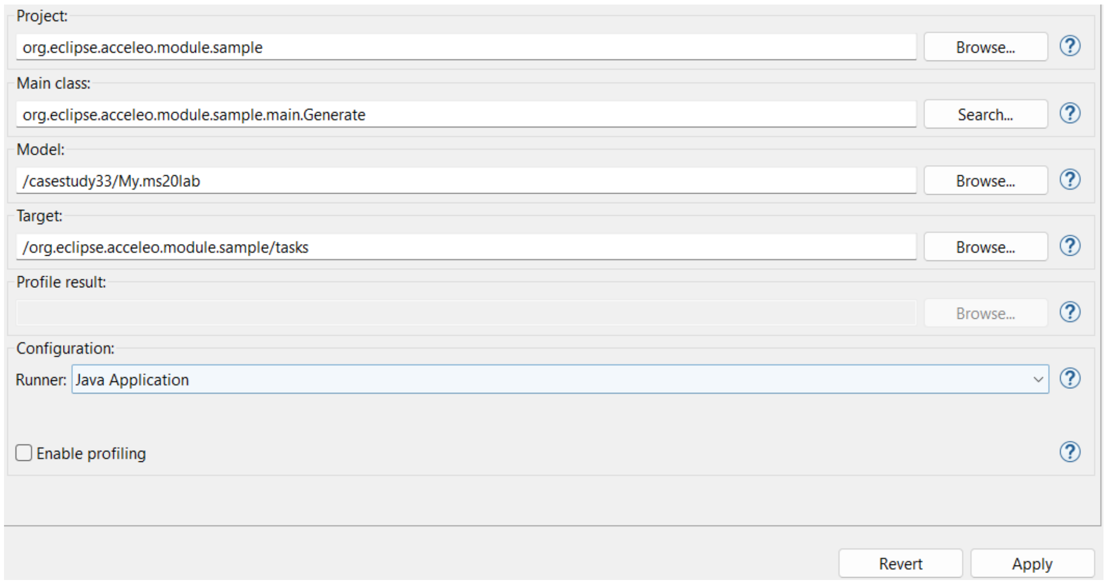

Project: org.eclipse.acceleo.module.sample
Main class: org.eclipse.acceleo.module.sample.main.Generate
Model: /casestudy33/My.ms20lab
Target: /org.eclipse.acceleo.module.sample/tasks
Profile result:
Configuration:
Runner: Java Application
Enable profiling

Figure 40.Run configuration of casestudy3

The water management system in the third case study includes additional features such as main motors connected to both the well and the bore. The system consists of a main supply system and four distinct motors for diverse areas, including fields, farms, and houses, as can be seen in the first two case studies. Waterman initiates the process by sending commands to the Wi-Fi module, instructing it to turn on the main motors linked to the well or bore that provides water to the main tank. After the main tank is full, the Wi-Fi module transmits further instructions to turn on the motors attached to the different areas while taking into consideration each one's particular water needs. To optimize water distribution, the maximum water level is set higher for fields than for farms and houses. Sensors are connected to an Arduino to ensure exact water distribution. The Arduino then receives messages indicating when the water has been completely delivered to a particular area. The Arduino turns on LED lights to show "Water Fully Delivered" once delivery is accomplished and turns on a buzzer to make a unique beep sound as an extra assurance. This water management system gives Waterman the ability to effectively manage water distribution, provide priority to areas with high water demands, and offer real-time feedback, providing an ideal and long-lasting water supply for residences, farms, and fields.

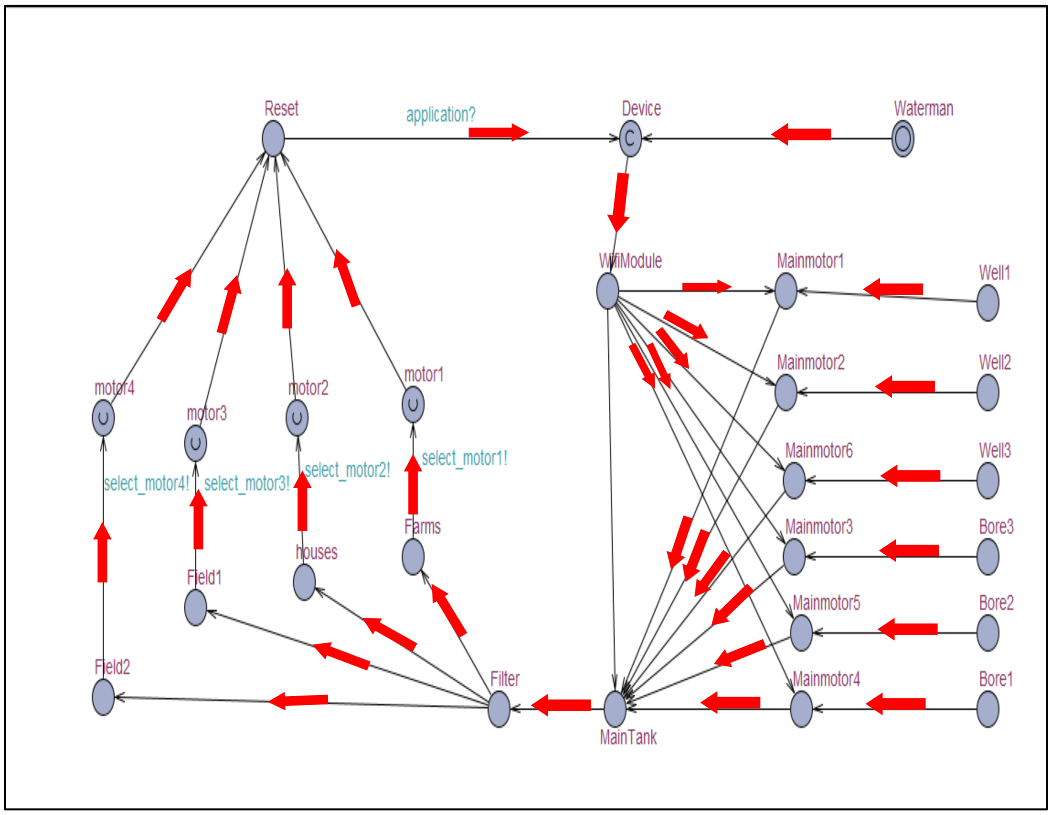


Figure 41. Main water supply model of case study 3

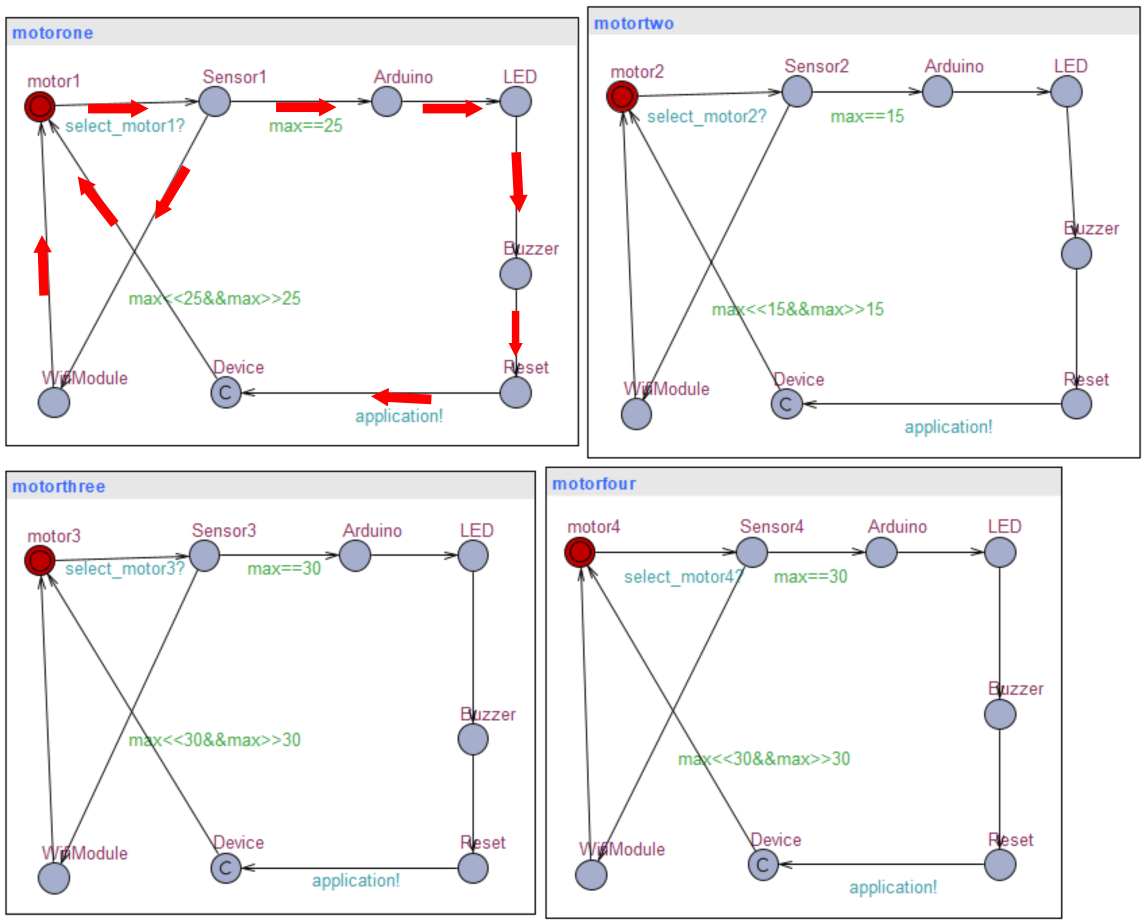
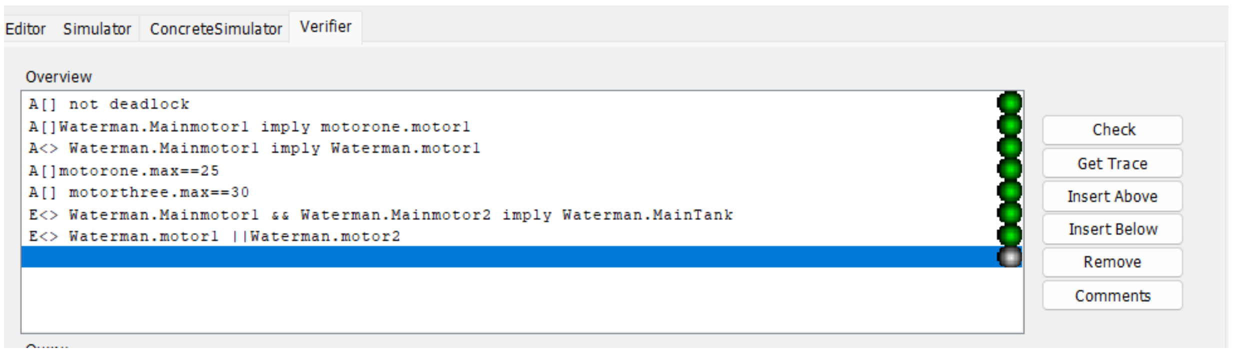


Figure 42. Motors model of case study 3

In this case study, we look at a comprehensive water supply system that consists of a main supply model at the center and different individual motor models as shown in Figures 41 and 42. The major source of water distribution is the main supply model. Through a synchronization command, the main supply model synchronizes with the matching motor model when it gets close to a certain motor location.



```
A[] not deadlock
A[]Waterman.Mainmotor1 imply motorone.motor1
A<> Waterman.Mainmotor1 imply Waterman.motor1
A[]motorone.max==25
A[] motorthree.max==30
E<> Waterman.Mainmotor1 && Waterman.Mainmotor2 imply Waterman.MainTank
E<> Waterman.motor1 ||Waterman.motor2
```

Figure 43.Properties of case study 3

In a model checking context, the property "A[] not deadlock" often denotes that there shouldn't be any deadlock situations in the system. According to this UPPAAL property, "A[] Waterman.Mainmotor1 imply Waterman.motor1," if the "Mainmotor1" of "Waterman" is active (true), it implies that "motor1" of "Waterman" must likewise be active (true) for all feasible execution pathways (A[]). According to the UPPAAL property "A[] Motorone.max==25," the value of "Motorone. max" shall always be equal to 25 for all feasible execution pathways (A[]) and the max value of motor three is 30 for all feasible execution pathways. As a "Reachability" property, the UPPAAL property "E>Waterman.motor1 || Waterman.motor2" determines if a path in the system exists where either "Waterman.motor1" or "Waterman.motor2" (or both) becomes true at some point in time.

CHAPTER 6

DISCUSSION

I took a thorough approach to modeling and improving the Water Supply Management System for my research. The first step involves collecting and carefully examining the system's requirements to create an M2-level meta-model. I created a tree view representation to simplify and increase the user's understanding of the complex system. This visual framework improved the modeling process overall by enabling a more intuitive understanding of the system's complexity. In three separate case studies of my research, I looked at different facets of the Water Supply Management System in contexts from the real world. Through this research, I was able to properly evaluate and enhance the system in a variety of real-world situations. These case study models were then transformed into XTA files using the Acceleo model-to-text transformation framework.

Importantly, I imported these XTA files into the UPPAAL modeling environment where I produced three distinct models, each in line with a different case study. This made it possible to design a thorough investigation of the Water Supply Management System scenarios. The model is verified using essential properties including deadlock, safety, reachability, and liveness verification to make sure that these models are reliable and robust. In the context of my research, a deadlock is a situation in which the system has stopped functioning and is trapped. Reachability analysis determines if critical states can be accessed when required, whereas safety features make sure that undesirable system states are avoided. In the context of managing water supply, liveness properties guarantee that the system advances continuously.

Properties are satisfied by showing green signals which means that the model is validated. In conclusion, my research is a thorough investigation that included rigorous UPPAAL-based validation, model-to-text transformation, real-world case studies, and meta-modeling. This method greatly enhanced the Water Supply Management System while guaranteeing industry compliance and providing protection against potential problems like deadlock. It also ensured safety, reachability, and system liveness.

CHAPTER 7

CONCLUSION AND FUTURE WORK

This study proposes a novel model-driven methodology intended to improve the effectiveness and efficiency of managing water supply systems. The creation of the Model-Driven Water Supply (MWS) Framework is key to this strategy. This framework is deeply based on the Eclipse Modelling Framework and acts as a strong management and optimization framework for water supply systems. The MWS Framework is made up of several integral components, each of which is essential to its performance. The MWS Tree Editor, MWS Graphical Modelling Tool, and MWS Transformation Engine stand out among these. These components cooperate to make it easier to convert instance models into UPPAAL xta files, an essential step in the system validation and verification process. The role of the UPPAAL model checker is essential to this study. It is used to thoroughly validate several water supply system properties, such as reachability analysis, system liveness, deadlock prevention, and security measures. Following the modeling of system behavior using UPPAAL, these properties are examined to make sure the water supply system runs effectively and safely. To confirm the efficacy of the suggested MWS Framework, extensive case studies are conducted. The outcomes of these case studies show how useful the framework is in modeling water supply systems. It enables users to manage water resources sustainably and effectively by optimizing water allocation and identifying potential vulnerabilities like water leaks, poor resource distribution, and poor infrastructure. The framework could potentially be improved more in the future. Future upgrades could include the incorporation of a billing system, allowing for more streamlined financial administration of the water supply system. Additionally, the addition of a water-cooling system can improve the system's overall effectiveness and sustainability. Researchers in this area might also consider combining UPPAAL with other formal verification tools like SPIN and PRISM.

REFERENCES

[1] Franzago, M., Di Ruscio, D., Malavolta, I., & Muccini, H. (2017). Collaborative model-driven software engineering: a classification framework and a research map. *IEEE Transactions on Software Engineering*, *44*(12), 1146-1175.

[2] Mohapi, L. J. (2017). A domain specific language for facilitating automatic parallelization and placement of SDR patterns into heterogeneous computing architectures.

[3] Kim, W. Y., Son, H. S., Kim, J. S., & Kim, R. Y. C. (2011). Adapting model transformation approach for android smartphone application. In *Advanced Communication and Networking: Third International Conference, ACN 2011, Brno, Czech Republic, August 15-17, 2011. Proceedings* (pp. 421-429). Springer Berlin Heidelberg.

[4] Hutton, C. J., Kapelan, Z., Vamvakeridou-Lyroudia, L., & Savić, D. A. (2014). Dealing with uncertainty in water distribution system models: A framework for real-time modeling and data assimilation. *Journal of Water Resources Planning and Management*, *140*(2), 169-183.

[5] Paul, M. J., Coffey, R., Stamp, J., & Johnson, T. (2019). A review of water quality responses to air temperature and precipitation changes 1: Flow, water temperature, saltwater intrusion. *JAWRA Journal of the American Water Resources Association*, *55*(4), 824-843.

[6] Kisvari, A., Lin, Z., & Liu, X. (2021). Wind power forecasting–A data-driven method along with gated recurrent neural network. *Renewable Energy*, *163*, 1895-1909.

[7] Jonoski, A., & Seid, A. H. (2016). Decision support in water resources planning and management: the Nile basin decision support system. *Real-World Decision Support Systems: Case Studies*, 199-222.

[8] Cantelmi, R., Di Gravio, G., & Patriarca, R. (2021). Reviewing qualitative research approaches in the context of critical infrastructure resilience. *Environment Systems and Decisions*, *41*(3), 341-376.

[9] Chwala, C., & Kunstmann, H. (2019). Commercial microwave link networks for rainfall observation: Assessment of the current status and future challenges. *Wiley Interdisciplinary Reviews: Water*, *6*(2), e1337.

[10] Ramotsoela, D. T., Hancke, G. P., & Abu-Mahfouz, A. M. (2019). Attack detection in water distribution systems using machine learning. *Human-centric Computing and Information Sciences*, *9*(1), 1-22.

[11] Krishnan, S. R., Nallakaruppan, M. K., Chengoden, R., Koppu, S., Iyapparaja, M., Sadhasivam, J., & Sethuraman, S. (2022). Smart water resource management using Artificial Intelligence—A review. *Sustainability*, *14*(20), 13384.

[12] Jagtap, H. P., Bewoor, A. K., Kumar, R., Ahmadi, M. H., Assad, M. E. H., & Sharifpur, M. (2021). RAM analysis and availability optimization of thermal power plant water circulation system using PSO. *Energy Reports*, *7*, 1133-1153.

[13] Liu, P., Wang, J., Sangaiah, A. K., Xie, Y., & Yin, X. (2019). Analysis and prediction of water quality using LSTM deep neural networks in IoT environment. *Sustainability*, *11*(7), 2058.

[14] Wang, F., Wang, Y., Zhang, K., Hu, M., Weng, Q., & Zhang, H. (2021). Spatial heterogeneity modeling of water quality based on random forest regression and model interpretation. *Environmental Research*, *202*, 111660.
[15]
[16] Nasir, N., Kansal, A., Alshaltone, O., Barneih, F., Sameer, M., Shanableh, A., & Al-Shamma'a, A. (2022). Water quality classification using machine learning algorithms. *Journal of Water Process Engineering*, *48*, 102920.
[17] Simonovic, S. P. (2012). *Managing water resources: methods and tools for a systems approach*. Routledge.
[18] . Fu, G., Jin, Y., Sun, S., Yuan, Z., & Butler, D. (2022). The role of deep learning in urban water management: A critical review. *Water Research*, 118973.
[19]. Sahu, M., Mahapatra, S. S., Sahu, H. B., & Patel, R. K. (2011). Prediction of water quality index using neuro fuzzy inference system. *Water Quality, Exposure and Health*, *3*, 175-191.
[20]. Wang, F., Wang, Y., Zhang, K., Hu, M., Weng, Q., & Zhang, H. (2021). Spatial heterogeneity modeling of water quality based on random forest regression and model interpretation. *Environmental Research*, *202*, 111660.
[21]. Rinaldi, M., & He, Z. (2014). Decision support systems to manage irrigation in agriculture. *Advances in agronomy*, *123*, 229-279.
[22]. Soleimani, R., Shoushtari, N. A., Mirza, B., & Salahi, A. (2013). Experimental investigation, modeling and optimization of membrane separation using artificial neural network and multi-objective optimization using genetic algorithm. *Chemical engineering research and design*, *91*(5), 883-903.
[23]. Boryczko, K., & Tchórzewska-Cieślak, B. A. R. B. A. R. A. (2014). Analysis of risk of failure in water main pipe network and of delivering poor quality water. *Environment Protection Engineering*, *40*(4), 77-92.
[24]. Wang, H., Zhang, D., & Shin, K. G. (2004). Change-point monitoring for the detection of DoS attacks. *IEEE Transactions on dependable and secure computing*, *1*(4), 193-208.
[25]. Keyhanpour, M. J., Jahromi, S. H. M., & Ebrahimi, H. (2021). System dynamics model of sustainable water resources management using the Nexus Water-Food-Energy approach. *Ain Shams Engineering Journal*, *12*(2), 1267-1281.
[26]. Cao, X., Xu, Y., Li, M., Fu, Q., Xu, X., & Zhang, F. (2022). A modeling framework for the dynamic correlation between agricultural sustainability and the water-land nexus under uncertainty. *Journal of Cleaner Production*, *349*, 131270.
[27]. Li, M., Liang, D., Xia, J., Song, J., Cheng, D., Wu, J., ... & Li, Q. (2021). Evaluation of water conservation function of Danjiang River Basin in Qinling Mountains, China based on InVEST model. *Journal of environmental management*, *286*, 112212.
[28]. Dlodlo, N., & Kalezhi, J. (2015, May). The internet of things in agriculture for sustainable rural development. In *2015 international conference on emerging trends in networks and computer communications (ETNCC)* (pp. 13-18). IEEE.

[29]. Li, L., Rong, S., Wang, R., & Yu, S. (2021). Recent advances in artificial intelligence and machine learning for nonlinear relationship analysis and process control in drinking water treatment: A review. *Chemical Engineering Journal*, *405*, 126673.
[30]. Kang, E., Adepu, S., Jackson, D., & Mathur, A. P. (2016, May). Model-based security analysis of a water treatment system. In *Proceedings of the 2nd International Workshop on Software Engineering for Smart Cyber-Physical Systems* (pp. 22-28).
[31]. Tariq, M. U., Nasir, H. A., Muhammad, A., & Wolf, M. (2012, April). Model-driven performance analysis of large scale irrigation networks. In *2012 IEEE/ACM Third International Conference on Cyber-Physical Systems* (pp. 151-160). IEEE.
[32]. Rahim, M., Hammad, A., & Ioualalen, M. (2017). A methodology for verifying SysML requirements using activity diagrams. *Innovations in Systems and Software Engineering*, *13*, 19-33.
[33]. Grobelna, I., Grobelny, M., & Adamski, M. (2010, June). Petri Nets and activity diagrams in logic controller specification-transformation and verification. In *Proceedings of the 17th International Conference Mixed Design of Integrated Circuits and Systems-MIXDES 2010* (pp. 607-612). IEEE.
[34]. Spiteri Staines, T. (2013). Transforming UML Sequence Diagrams into Petri Nets.
[35]. Goncalves, F. S., Pereira, D., Tovar, E., & Becker, L. B. (2017, November). Formal verification of AADL models using UPPAAL. In *2017 VII Brazilian Symposium on Computing Systems Engineering (SBESC)* (pp. 117-124). IEEE.
[36]. Waszniowski, L., & Hanzálek, Z. (2008). Formal verification of multitasking applications based on timed automata model. *Real-Time Systems*, *38*, 39-65.
[37]. Rushby, J. (2005). Test Suite Evaluation and Generation.
[38]. Kang, E. Y., Ke, L., Hua, M. Z., & Wang, Y. X. (2015, December). Verifying automotive systems in EAST-ADL/Stateflow using UPPAAL. In *2015 Asia-Pacific Software Engineering Conference (APSEC)* (pp. 143-150). IEEE.
[39]. Chen, D., Johansson, R., Lönn, H., Blom, H., Walker, M., Papadopoulos, Y., ... & Sandberg, A. (2011). Integrated safety and architecture modeling for automotive embedded systems. *Elektrotechnik und Informationstechnik*, *128*(6), 196.
[40]. Enoiu, E. P., Marinescu, R., Seceleanu, C., & Pettersson, P. (2012, July). Vital: A verification tool for east-adl models using uppaal port. In *2012 IEEE 17th International Conference on Engineering of Complex Computer Systems* (pp. 328-337). IEEE.
[41]. Watahiki, K., Ishikawa, F., & Hiraishi, K. (2011, October). Formal verification of business processes with temporal and resource constraints. In *2011 IEEE international conference on systems, man, and cybernetics* (pp. 1173-1180). IEEE.
[42]. Esfahani, F. G. (2018). Formal Modeling and Analysis of Mobile Ad hoc Networks.
[43]. Ogedebe, M. P., & Silas, F. A. Abuse of Unified Modeling Language Diagrams in Software Development.
[44]. Iftikhar, M. U., & Weyns, D. (2012). A case study on formal verification of self-adaptive behaviors in a decentralized system. *arXiv preprint arXiv:1208.4635*.

[45]. Zheng, M., Alagar, V., & Ormandjieva, O. (2008). Automated generation of test suites from formal specifications of real-time reactive systems. *Journal of Systems and Software, 81*(2), 286-304.

[46]. Halder, R., Proença, J., Macedo, N., & Santos, A. (2017, May). Formal verification of ROS-based robotic applications using timed-automata. In *2017 IEEE/ACM 5th International FME Workshop on Formal Methods in Software Engineering (FormaliSE)* (pp. 44-50). IEEE.

[47]. de Saqui-Sannes, P., & Hugues, J. (2012, February). Combining SysML and AADL for the design, validation and implementation of critical systems. In *ERTS2 2012* (p. 117).

[48]. de Saqui-Sannes, P., Apvrille, L., & Vingerhoeds, R. (2021). Checking SysML models against safety and security properties. *Journal of Aerospace Information Systems*, *18*(12), 906-918.

[49]. David, A., Möller, M. O., & Yi, W. (2002, March). Formal verification of UML statecharts with real-time extensions. In *International Conference on Fundamental Approaches to Software Engineering* (pp. 218-232). Berlin, Heidelberg: Springer Berlin Heidelberg.

[50]. Aoki, Y., & Matsuura, S. (2014, August). Verifying security requirements using model checking technique for UML-based requirements specification. In *2014 IEEE 1st International Workshop on Requirements Engineering and Testing (RET)* (pp. 18-25). IEEE.

[51]. Panach, J. I., España, S., Dieste, O., Pastor, Ó., & Juristo, N. (2015). In search of evidence for model-driven development claims: An experiment on quality, effort, productivity and satisfaction. *Information and software technology, 62*, 164-186.

[52]. Braghin, C., Cimato, S., Damiani, E., Frati, F., Mauri, L., & Riccobene, E. (2020). A model driven approach for cyber security scenarios deployment. In *Computer Security: ESORICS 2019 International Workshops, IOSec, MSTEC, and FINSEC, Luxembourg City, Luxembourg, September 26–27, 2019, Revised Selected Papers 2* (pp. 107-122). Springer International Publishing.

[53]. Shonle, M., Neddenriep, J., & Griswold, W. (2004, October). Aspectbrowser for eclipse: a case study in plug-in retargeting. In *Proceedings of the 2004 OOPSLA workshop on eclipse technology eXchange* (pp. 78-82).

[54]. Burgueno, L., Cabot, J., Li, S., & Gérard, S. (2022). A generic LSTM neural network architecture to infer heterogeneous model transformations. *Software and Systems Modeling*, *21*(1), 139-156.

[55]. Carvalho, A., Carvalho, J., Pinto, J. S., & De Sousa, S. M. (2010). Model-checking temporal properties of real-time HTL programs. In *Leveraging Applications of Formal Methods, Verification, and Validation: 4th International Symposium on Leveraging Applications, ISoLA 2010, Heraklion, Crete, Greece, October 18-21, 2010, Proceedings, Part II 4* (pp. 191-205). Springer Berlin Heidelberg.

[56]. Sial, S. (2008). Exploring the mindset of the British-Pakistani community: The sociocultural and religious context. *Conflict and peace studies, 1*(1), 1-32.

[57]. Viyović, V., Maksimović, M., & Perisić, B. (2014, July). Sirius: A rapid development of DSM graphical editor. In *IEEE 18th International Conference on Intelligent Engineering Systems INES 2014* (pp. 233-238). IEEE.

[58]. Ruidas, D., Pal, S. C., Saha, A., Chowdhuri, I., & Shit, M. (2022). Hydrogeochemical characterization based water resources vulnerability assessment in India's first Ramsar site of Chilka lake. *Marine Pollution Bulletin*, *184*, 114107.
[59]. Sindico, A., Di Natale, M., & Panci, G. (2011, July). Integrating SysML with Simulink using Open-source Model Transformations. In *SIMULTECH* (pp. 45-56).
[60]. Buchmann, T., Hammoudi, S., van Sinderen, M., & Cordeiro, J. (2012, July). Valkyrie: A UML-based Model-driven Environment for Model-driven Software Engineering. In *ICSOFT* (pp. 147-157).
[61]. Mehdy, A. N., & Mehrpouyan, H. (2021, August). Modeling of personalized privacy disclosure behavior: A formal method approach. In *Proceedings of the 16th International Conference on Availability, Reliability and Security* (pp. 1-13).
[62]. Barbuti, R., & Tesei, L. (2004). Timed automata with urgent transitions. *Acta Informatica*, *40*, 317-347.
[63]. Larsen, K. G., Pettersson, P., & Yi, W. (1995). Model-checking for real-time systems. In *Fundamentals of Computation Theory: 10th International Conference, FCT'95 Dresden, Germany, August 22–25, 1995 Proceedings 10* (pp. 62-88). Springer Berlin Heidelberg.
[64]. Behrmann, G., David, A., & Larsen, K. G. (2004). A tutorial on uppaal. *Formal methods for the design of real-time systems*, 200-236.