UNIVERSITY OF OTTAWA

A Rule-based Expert System for Predictive Maintenance of a Hybrid Bus

by

Wenjie Chen

Thesis submitted to the University of Ottawa
in partial fulfillment of the requirements for the
M.Sc. degree in Systems Science

Department of Electrical Engineering and Computer Science
Faculty of Systems Science
University of Ottawa

Abstract

This thesis describes the design and implementation of constructing a predictive maintenance system for a hybrid vehicle to meet the requirements of the STO (Société de transport de l'Outaouais). Thousands of sensors installed on the bus allow us to observe the real-time performance of the bus while it is running. Abnormal sensor values represent adverse operating conditions and bring attention to the inevitable failures of a bus's components. Therefore, by analyzing real-time sensor streams, predictive maintenance is accomplished based on the unnatural behaviour of a hybrid bus.

Currently, transport companies still employ traditional methods of maintenance planning, such as emergency maintenance and preventive maintenance. Traditional maintenance strategies require a great deal of technicians and time to inspect the buses regularly and carefully. In comparison, predictive maintenance can monitor the performance of buses based on the condition of their equipment. To collect data from the hybrid bus and share data with the Internet, IoT technology is adopted to develop predictive maintenance architecture for a fleet management system. Our team devised an IoT architecture for the fleet management system, including the perception layer, middleware layer and application layer. My work focuses on the perception layer, which is responsible for analyzing sensor values, reporting failures of a hybrid bus and connecting with cloud-servers.

As one of the predictive maintenance methods, the expert system (also known as a knowledge-based expert system) is built to store expert knowledge in a specific area. The expert system presented in this thesis can store failures of hybrid buses, symptoms of which were provided to us by technicians from the STO. Such breakdowns assist the expert system in predicting the malfunctions of the bus's components based on the symptoms. Inspired by the IDEA methodology, failure symptoms can be represented by active rules with three essential components: event, condition and action. These rules can also be translated into active database features like triggers and mapped into an active database. A gateway is installed on a bus and composed of four modules: data acquisition module, active rules module, rules management module and user interface module. Within the parameters of the architecture and the gateway, this thesis analyzes the entities, relationships and operations in the dynamic system and forms a relational database to store the information related to the bus and active rules.

Keywords: predictive maintenance, hybrid bus, IoT, expert system, active rules

Acknowledgements

I would like to sincerely thank my supervisors, Dr. Iluju Kiringa and Dr. Tet Yeap for their continuous support, guidance and encouragement. Their constructive comments on this work helped me develop my ideas throughout the project up to completion.

This research was partially supported by the STO. As such, I would also like to thank all STO staff who provided insight and expertise that greatly assisted the research and improved my work.

I would also like to extend my gratitude to all my colleagues for sharing their wisdom with me during the development of this research. Finally, I am immensely grateful to my family and friends for their advice and financial support. Without all the support I was fortunate to receive, this work would not be possible.

Contents

List of Figures

List of Tables

Nomenclature

ECU	**E**lectronic **C**ontrol **U**nit
CAN	**C**ontroller **A**rea **N**etwork
UI	**U**ser **I**nterface
API	**A**pplication **P**rogram **I**nterface
DBMS	**D**atabase **M**anagement **S**ystem
UML	**U**nified **M**odeling **L**anguage
FMS	**F**leet **M**anagement **S**ystem
IoT	**I**nternet **o**f **T**hings
PdM	**P**redictive **M**aintenance
RFID	**R**adio **F**requence **ID**
ICE	**I**nternal **C**ombustion **E**ngine
PDU	**P**rotocol **D**ata **U**nit
LSB	**L**east **S**ignificant **B**yte
PGN	**P**arameter **G**roup **N**number
SPN	**S**uspect **P**arameter **N**umber
KBMS	**K**owledge-**B**ased **M**anagement **S**ystem
AI	**A**rtificial **I**ntelligence
ADBMS	**A**ctive **D**atabase **M**anagement **S**ystem
VN	**V**ehicle **N**ode
SLN	**S**erver **L**ead **N**ode
RN	**R**oot **N**ode
DEF	**D**iesel **E**xhaust **F**luid

Chapter 1

Introduction

1.1 Motivation

Public transportation is an integral aspect of city life and plays an important role in the optimal running of modern cities. With its ability to facilitate our daily movement and planning, using public transportation saves us significant monthly expenses such as gas fees and parking fees among others. In addition, the public transportation system reduces air pollution and improves fuel efficiency. Because of its meaningful benefits, public transportation is already well-developed in most urban areas worldwide. However, the high-mobility characteristic of buses allows for higher risks of potential failure anywhere and anytime, which leads to much inconvenience for the public.

To select an appropriate maintenance schedule for the bus transit system, transportation companies must consider factors such as safety, reliability, vehicle life and cost. Maintenance planning is a regular routine in which the transportation system carefully evaluates the status of a bus and fixes any potential problems based on manufacturer recommendations. However, traditional maintenance planning, also known as preventive maintenance, is not always the most efficient way. This is because, in most cases, recommended technical inspections rely on time factors rather than the actual state of the vehicle's components. Due to limited parking space and the complexity of bus components, most routine maintenance does not occur daily but at fixed times or mileage intervals. With this approach, there is no guarantee that a bus will work fine between regular maintenance activities. If failure occurs earlier than the scheduled maintenance service, the bus will require unexpected technical inspections, which results in redundant costs and wasted time for a transportation company.

After an unexpected breakdown event of a bus, most agents would perform another maintenance approach called reactive maintenance or corrective maintenance. This approach is typically used by technicians to facilitate replacement of failed parts based on the failure symptom in order to get buses back on the road without delay. It requires less planning time and effort than a preventive maintenance approach. However, the unexpected downtime makes the reactive maintenance approach only applicable to bus components with backup and insufficient inspections may cause inefficient use of replaced parts and safety issues. Similar to reactive maintenance planning, the operations of the emergency maintenance approach are also determined based on the fault reports from a person or software. The difference between the emergency maintenance approach and the reactive maintenance approach is the severity of the bus failure. In the case of the former, the failure is more severe and needs immediate action to avoid dangerous conditions.

There are overwhelming studies, reports and articles that analyze the maintenance systems of bus companies and propose and evaluate different types of combined solutions under different circumstances. According to these studies, organizing a proper fixed maintenance system for managers is no longer a big problem and requires new methods to monitor bus operation and minimize unnecessary service interruption.

Monitoring a bus's running state requires collection of the sensor information from a bus and shows data to users in real-time. The complexity and increasing number of electronic control units (ECUs) equipped on a bus increase the difficulty of information exchange and reduce the reliability and security of information processing. Nowadays, the controller area network (CAN) is widely applied in heavy vehicles due to its extremely reliable communication and easy implementation. As a CAN application layer protocol, the SAE J1939 has been developed to provide serial data communications between ECUs. An electronic control unit controls a series of actuators to ensure optimal performance by reading values from a vast number of sensors within different parts of the bus. During the operation of a bus, a sensor detects value changes such as light, temperature or fuel level. The data is then collected by sensor networks and attached to J1939-data streams for future analysis.

1.2 Objective

The current maintenance system for public vehicles follows three maintenance strategies: preventive maintenance approach, reactive maintenance approach and emergency maintenance approach. All three strategies are performed either when a fault occurs or

on a fixed schedule. As maintenance scheduling continues to develop rapidly, an efficient maintenance system is no longer a trivial problem. It requires a new computerized method to reduce unnecessary inspections, which is a new challenge that transportation companies such as the STO are faced with. The solution proposed in this thesis aims to predict bus failure based on the bus's operating conditions rather than a specified time, which is both time and cost-effective while presenting fewer interruptions. It should be noted that this research has been conducted in close cooperation with the STO and that all information and data used in this thesis have been provided by the STO.

Since the system is expected to function based on the real-time information of the vehicle, and considering maintenance requirements, a new solution called predictive maintenance has been adopted in the thesis. Predictive maintenance (PdM) refers to identifying incoming failed parts or components when certain real-time information indicates signs of faults. Our system continues to monitor the bus operation state by regularly analyzing sensor data from sensor networks of the public vehicle. Once the sensor data satisfies the failed value range, which means that breakdown will occur at any moment, a warning sign is immediately sent to the driver or technicians to inform of the failure before it occurs. Drivers and technicians could take corresponding measures such as discontinuing bus usage or continuing to run for a while. This method improves the work efficiency of buses and ensures the safety of the public as well as the buses.

1.3 Methodology

This thesis aims to develop a predictive maintenance system to continuously track and monitor a bus's performance. The proposed predictive system adopts expert systems as its predictive maintenance strategy. This strategy is pretty straightforward and applies expert knowledge to solve problems in a particular area. The expert knowledge needs to be acquired from experts, constructed into IF-THEN rules, and stored in the knowledge base. In my work, active rules were used to provide reactive actions based on real-time sensor data. The active database management system supports active features such as triggers and stored procedures to represent the active rules. Based on the algorithms of a rule's processing, specific components of active rules were analyzed. Given the algorithms and components, the IDEA methodology and Chimera language helped to analyze the structure of the whole expert system and behavioural changes of active rules. Also, the Chimera language supported to design conceptual model with classes, operations and relationships, and active rules with triggers. After designing, I mapped the classes and triggers written by the Chimera language into tables and triggers in

MySQL. And a UI module was built to allow technicians to easily input new knowledge and track the sensors' performance. Most of the buses in the STO company belong to Nova Bus LFS HEV Series, which are electric-diesel hybrid buses. After reading a lot of materials, I listed 36 possible failure symptoms and grouped them based on mechanical systems. Meanwhile, the failure symptoms have been constructed into IF-THEN rules with components and predefined orders. Our team obtained 10-hours historical data from a hybrid bus in STO company during the normal running through SAE J1939 protocol. Empirical models of sensors were built by plotting data distributions to observe normal running state. To evaluate the correctness of actions executed by the system, several experiments using different datasets were performed.

1.4 Contributions

The main contribution of this thesis is to develop a rule-based expert system that serves the needs of maintenance. All steps together in my work constitute the proposed system and achieve the objective of predicting failures to reduce random inspections for buses. Followings are the components of my work:

1. Proposed an empirical model for a hybrid bus.
2. Designed a rule-based expert system for a hybrid bus in STO.
3. Implemented the Flask-SQLAlchemy extension of Python to build the rule-based expert system.
4. Established a user interface module.
5. Performed a series of experiments to verify the functionalities of the system.

1.5 Organization

The remaining parts of this thesis are organized as follows:

- The background of my work is an IoT-based framework for managing a fleet of the STO and diagnosing faulty buses. Besides introducing the methods used in my system and providing a literature review, Chapter 2 also explains the IoT technology implemented in the fleet management system (FMS). While building

the rule-based expert system for one hybrid bus, it is also necessary to learn how a hybrid bus works and how a rule-based expert system operates, especially the design procedures of expert systems and the structure of active rules. Only in this way can the system be understood entirely, and the models be built to standard.

- Chapter 3 describes the architecture of the entire FMS developed by our group. My thesis only focuses on the vehicle layer to make predictions for a public vehicle, while the whole FMS endeavours to use an IoT-based framework to make predictions for a fleet of public transport vehicles. Also, in his work my colleague has studied the vehicle node, discussed the structure and use case of gateway devices and generated the processing algorithms of rules. Subsequently, a detailed analysis of the requirements for the proposed system is illustrated.

- Beyond the basic principles described in Chapter 2 and Chapter 3, Chapter 4 designs conceptual models and behavioural models by studying the components, relationships and operations of the system. Mapping the entities and uniquely mapping the rules into software classes is the last but most critical step in the completion of my work.

- A series of empirical models are constructed using historical records gathered from a hybrid bus during a 10-hour regular operation period. Useful features of each sensor have been extracted to represent the bus's normal running state by analyzing historical records. Through simulation, the experiments and results of evaluating the system are illustrated in Chapter 5. The results show the performance and accuracy of the system as well as the real-time monitoring user interface.

- Finally, Chapter 6 summarizes the essential points in the thesis and discusses future work and more attributes that could be added in the system.

Chapter 2

Background Theory

This chapter introduces the advanced technology of Internet of Things (IoT) and related works on the fleet management system monitoring a fleet's operation. Then, we seek to understand the working principles and structure of a hybrid bus and the method to collect sensor data from its sensor networks. Moreover, the current development of predictive maintenance has also been discussed in this chapter. Finally, the steps of creating a rule-based expert system are presented in detail.

2.1 Internet of Things (IoT)

The IoT refers to taking everyday physical objects and connecting them to the Internet on computers or smartphones where resulting data can be collected and shared. For example, a streetlamp can be automatically switched on or off by sensing the light in its environment at dawn or dusk with light sensors. When objects connect to the Internet, it means that their information can be sent or received using different kinds of sensors. These sensors create the ability to automatically collect various information from the environment, which, in turn, helps us to make smart decisions according to this information.

2.1.1 History of IoT

IoT is now considered one of the most essential technologies, and devices with IoT technology improve the quality and efficiency of our life in many areas, such as business, healthcare, smart cars, and so forth. The idea of adding sensors to things was a topic

of discussion throughout the 1980s and 1990s. However, the advancement of this idea was restricted due to the undeveloped technologies of that time[3][4]. The term IoT has 20 years of history and was initially mentioned in a presentation by Kevin Ashton in 1999. He worked at Procter & Gamble (P&G) where he delivered a presentation called "Internet of Things" and wanted to bring radio frequency identification (RFID) to the attention of P&G's senior management. His idea, adding RFID tags to expensive pieces of equipment to help track their location, succeeded in attracting the attention of executives, but still, the term did not gain traction for the next ten years.

Since the summer of 2010, along with the Chinese government listing IoT as the top priority in their Five-Year-Plan, IoT technology has been enjoying increased popularization. In the same year, it was discovered that Google's Street View service not only produced 360-degree images but also stored thousands of people's Wi-Fi networks, sparking debate about whether Google had launched a new strategy to index the physical world outside the Internet[4]. The term has been continuously and rapidly developed since and, until now, IoT is the most popular word used to describe this new interconnected world.

2.1.2 Architecture of IoT

Previously, it was widely accepted that IoT referred to the use of RFID technology to tag and identify objects. Today, with the development of communication networks, IoT extends to the connectivity of objects at any time, anywhere and with anyone via the effective integration of sensor networks, RFID and wireless and wired networks[5]. This revolution strengthens the ability to track anything from objects to animals and people, and more importantly, it encourages the information exchange between smart devices and accessible data collection for designing more smart applications.

Despite the lack of a clear definition of IoT, its architecture at technical levels shown in Figure 2.1 is generally accepted and includes the perception layer, network layer and application layer[3].

Perception Layer This layer includes a series of IoT devices such as sensors, GPS, RFID tags or sensor networks. These devices identify objects and collect information related to them. This information can be used to describe location, temperature or pressure and can be securely sent to the network layer for processing.

Network Layer The network layer, also called transmission layer, is responsible for transferring data from IoT devices to the information analyzing system. A vast

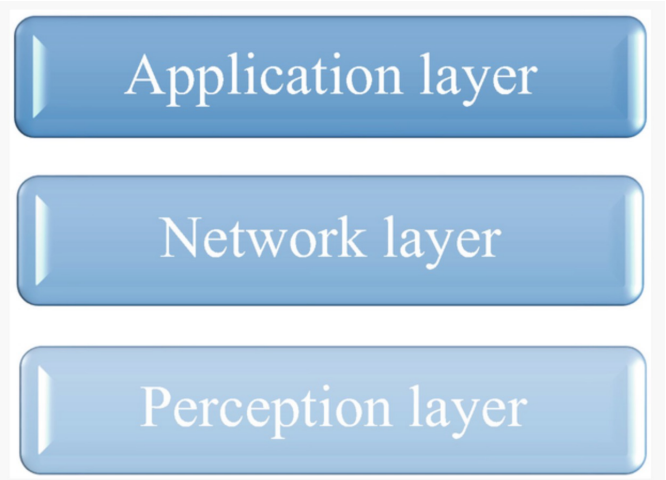

FIGURE 2.1: Architecture of an IoT system[1][2]

amount of data is delivered through communication networks such as wired or wireless networks, 3G or 4G, Bluetooth or other communication technologies.

Application Layer The application layer receives the processed data from the network layer and displays the results via applications designed for realizing smart cities, smart homes or smart businesses.

The three-layer architecture describes the main concept behind IoT. However, when it comes to applying IoT technology on large-scale applications with a massive amount of data, the three-layer architecture cannot provide precise models. Because massive data may need the cloud to manage and store data, it is necessary to consider management models and business models. Miao Wu et al. (2013) [2] proposed a five-layer model by analyzing the structure of the Internet network and the communication network. They extracted certain features from the two structures and combined them with the Internet of Things to obtain a more flexible architecture. Rishika Mehta et al. (2018) also analyze the five-layer architecture and illustrate the functionalities of each layer [5]. The five-layer architecture is displayed in Figure 2.2. Compared with three-layer architecture, the original Network Layer has been expanded by Processing Layer and Transport Layer, and Business Layer has been added into the architecture.

Perception Layer Besides the existing functionalities of identifying objects and gathering information, this layer can transform the gathered data into digital signals and allow devices with nanotechnology installed in the object to acquire more information.

Transport Layer Like the network layer in the 3-layer architecture, the transport layer is used for transmitting information from the previous layer to the next layer

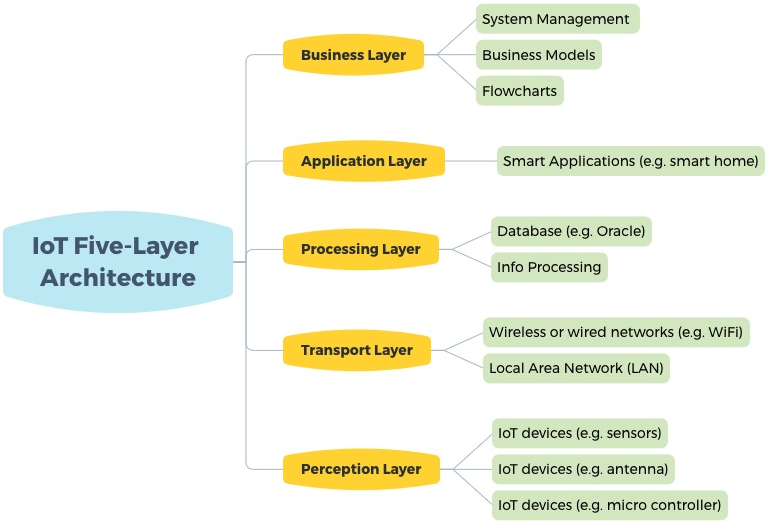

FIGURE 2.2: Five-layer architecture of an IoT system

through various communication networks such as Wi-Fi and many protocols like IPv6.

Processing Layer The processing layer, also called middleware layer, is developed given the vast information flowing into the transport layer. This layer stores information from the transport layer in the database and analyzes it using cloud computing or other advanced technologies.

Application Layer Like the application layer in the three-layer architecture, users can view the results via applications. Different applications constitute aspects of smart homes, smart cities and smart businesses.

Business Layer This layer is responsible for controlling the entire IoT system as a manager. It provides the business models, flowcharts and graphics based on the data acquired from the previous layer and may predict future actions by analyzing the results.

2.1.3 Pros and cons of IoT

Although most IoT applications are still in the initial stages of development, the IoT is widely used in many fields. IoT enables people to live in smart cities, offers intelligent devices to smart homes, and provides real-time observation to smart transport [6]. IoT, as a new trending technology, provides enormous benefits along with some challenges.

The advantages of IoT are as follows:

- The ability to acquire information from everywhere at any time on any device.
- Timely communication with other connected devices through sharing data.
- Real-time data represents actual performance of a system and provides references for subsequent decision-making.
- Has human-like behaviour which reduces labour costs and improves efficiency.

However, with the rapid development of IoT, some urgent issues have arisen:

- The improved number of connected devices and vast amount of data being shared may cause security and privacy issues. Since most data includes personal information, preventing unauthorized access is integral to securing data and protecting privacy.
- Companies may need a large amount of IoT devices to acquire massive data, which causes the difficulty of developing a standard management system to handle and maintain their systems.
- The connected devices may be affected and influenced by one another due to one bug in the system. It becomes harder to maintain the system with increasing numbers of connected devices.

The development of our system is inspired by the IoT method. The network of sensors installed on the bus can monitor the real-time operation states and diagnose fault effectively. The architecture of our IoT-based system will be presented in the next chapter.

2.2 Hybrid vehicles

A public vehicle here refers to a bus, which is commonly used in cities and towns to transport passengers short distances to their destination. A growing number of transit companies are adopting hybrid buses due to their features of less energy consumption and emissions.

2.2.1 The structure of a hybrid bus

A hybrid-electric bus is powered by an electric motor and an internal combustion engine (ICE) [7][8]. Electric motors have a higher torque density than an ICE and do not need gas to generate power, which reduces harmful exhaust emissions. However, considering the lack of energy storage in the battery of electric buses, the combination of ICE and electric motors seems to be the best solution to achieve high efficiency. The major components of a hybrid driven system include an internal combustion engine, an electrical generator, a battery pack, and an electric motor, as shown in Figure 2.3.

The power of a hybrid bus comes from two power sources, diesel and electricity [9]. Bus batteries store energy and recharge when the bus decelerates, and the diesel engine provides extra energy when the bus requires more power than battery capacity can provide.

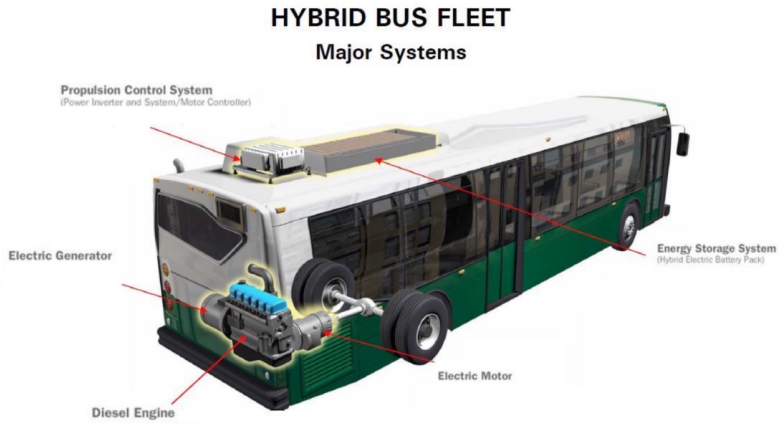

FIGURE 2.3: Predictive maintenance classifications

2.2.2 Sensor networks and communication protocol

In order to analyze bus running operations, a vast amount of sensor networks have been equipped on a bus to collect raw data, such as speed, emission, and distance. For vehicles, the sensor networks are driven by standard industry protocols such as SAE J1939. According to [10], [11] and [12], SAE J1939 protocol is a Society of Automotive Engineers' standard recommended practice used for communication and diagnostics among vehicle components. A vast amount of sensor data is collected by a sensor network and attached to J1939-data streams for future analysis.

J1939 is a higher-layer protocol placed on top of a controller area network (CAN). Communication happens between microprocessor systems, also called electronic control

units (ECU), in any heavy-duty vehicle. J1939 messages are organized into protocol data units (PDU), which consist of an identifier and eight data bytes. Numerical data that is larger than a single byte is sent with the least significant byte (LSB) first. J1939 uses CAN 2.0B with the extended (29 bit) identifier. The CAN identifier consists of a priority (3 bits), a reserved (1 bit), a data page (1 bit), a PDU format (one byte), PDU specific (one byte) and a source address (one byte). Figure 2.4 demonstrates the structure of the J1939 packet.

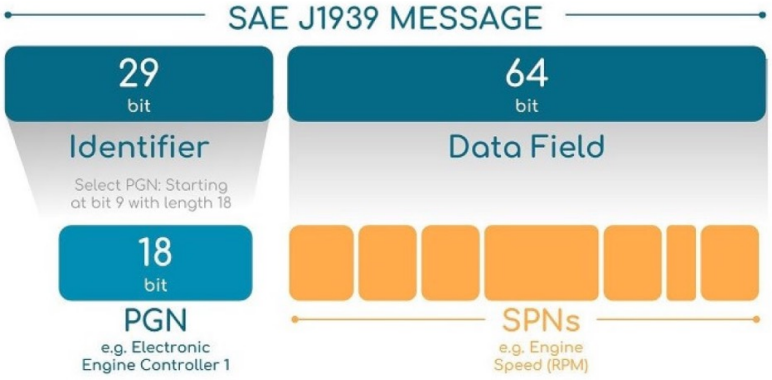

FIGURE 2.4: The structure of SAE J1939 packet

Additionally, SAE J1939 uses predefined parameter tables (parameter groups), which contain parameter group numbers (PGN) and suspect parameter numbers (SPN). The structure of the PGN allows a total of 17344 different parameter groups. For example, engine temperature as a parameter group with group number 65265 includes different parameters with different SPN, like engine coolant temperature (SPN 110), fuel temperature (SPN 174), and engine oil temperature (SPN 175). An SPN is assigned by the SAE to a specific parameter corresponding to one or more bytes of data length within a parameter group. Table 2.1 is an example of data descriptions within parameter group Engine Temperature as PGN 65262.

Bytes	SPN	Parameters
1	110	Engine Coolant Temperature
2	174	Fuel Pressure
3-4	175	Engine Oil Temperature
5-6	176	Turbocharger Oil Temperature
7	52	Engine Intercooler Temperature
8	1134	Engine Intercooler Thermostat Opening

TABLE 2.1: Data Description of PGN 65262 Engine Temperature

2.3 Predictive maintenance

For large and complex transportation systems such as airplanes, trains and ships, traditional maintenance planning is not the best approach to discovering and solving failures, especially considering the embedded subsystems and complicated elements (mechanic, electronic or software). Therefore, maintenance strategies need to be updated from the traditional "fail and fix" approach to the predicted "real-time detection" approach.

As stated in the previous chapter, predictive maintenance is a maintenance approach that monitors for potential failures in the future and makes maintenance plans in advance, before faults actually occur. The main idea behind PdM is to build proper statistical models with a relatively large amount of operation data and continuously monitor the operating parameters [13][14][15]. Based on the resulting models, failure can be predicted with real-time information. In this way, the maintenance measurements are performed based on the specific predicted faults. The concept of predictive maintenance was adopted in the 1990s in the industrial world and has now become the most popular maintenance method. From the perspective of asset management for companies, especially those that need to implement frequent maintenance, keeping reliable and long-term running states of equipment is exceedingly crucial for reducing maintenance costs. With the rapid development of various technologies such as sensors technology, big data analytics and cloud services, monitoring the condition and operation of machines is no longer a big challenge. Achieving predictive maintenance offers companies more efficient and effective machines with less downtime and cost than regular maintenance methods. According to a predictive maintenance report from Market Report Future, the market size of predictive maintenance is predicted to grow to 6.3 billion by 2022 [16]. Given the developing trends and the importance of predictive maintenance, a business should build a related framework for its production system to increase the level of productivity.

2.3.1 Comparison between maintenance strategies

Maintenance strategies can be defined as standard rules which guide practice, reflect the physical laws for repairing and summarize the understanding of the overall maintenance work. Whether or not correct maintenance planning is performed will directly affect the overall system. The development of maintenance strategies can be divided into three stages [17][18]:

Stage 1: Breakdown Maintenance As the first generation of maintenance planning

in the mid-1940s, breakdown maintenance aimed at repairing equipment after a failure has happened. However, this method is passive and not conducive to saving maintenance costs due to increased unscheduled breakdowns. Due to its shortcomings, this method gradually fell out of favour with most companies and has been retired. Now, companies still use this method after machines break down.

Stage 2: Preventive Maintenance Preventive maintenance was adopted as a primary maintenance strategy from the 1960s to the 1980s. It is based on time and does not consider the condition of the equipment. In the road, railway, navigation and aviation industries, a breakdown may be mainly caused by fatigue and wear, in which case preventive maintenance can decrease the amount of breakdowns. However, unnecessary inspection with unexpected costs sometimes happens due to time-based maintenance.

Stage 3: Predictive Maintenance With the development of monitoring and diagnostic systems, the idea of predictive maintenance was proposed in the 1980s. This method involves making a diagnosis based on the condition of the equipment, which avoids unscheduled breakdown and unnecessary inspection. Using monitoring techniques, parameters such as temperature, pressure, flow rate and other information related to the equipment are observed for reference to predict potential failure. This method mostly relies on computational techniques. Therefore, it requires colossal effort and time investment to construct and verify an appropriate predictive maintenance system.

Breakdown, preventive and predictive maintenance approaches are three major maintenance strategies used to maintain regular operations of equipment. Table 2.2 illustrates the benefits, downsides and possible costs for each strategy. The comparison of the three maintenance methods in Table 2.2 depicts that, for the longer term, the time and cost savings of predictive maintenance are most significant in comparison to the other maintenance strategies.

2.3.2 Methodologies for predictive maintenance

Current research on predictive maintenance focuses on a wide variety of fields and different methods. Hui Yin et al. [19] integrated statistical models and maintenance planning, including corrective maintenance and predictive maintenance, and tested the delayed monitoring policies in ten different scenarios for validating the models. In order to improve the performance of trains, Freceena Francis and Maya Mohan [20] proposed an ARIMA model for analyzing failure trends with real-time data and fault logs in the

clouds of processing units. Fault logs help to recognize failures trends and extract failure features using the ARIMA model. By comparing real-time sensor data and failure features, the system can predict the faults of trains and make a diagnosis. Pedro Cesar et al. [21] combined prediction activities with maintenance schedules to predict rail and geometry defects. Two years of historical rail and geometry data has been analyzed and divided into segments with details of breakdown location and maintenance measurements. Based on the segments with specific labels, features of these segments were extracted, and non-linear regression models were adopted to predict defects. For predicting the server life of buildings, an expert system was generated by Andrés JoséPrieto Ibáñez et al. [22], and as references, two standards in risk management fields were analyzed while developing the expert system.

There are plenty of studies and research introducing the predictive maintenance method. Based on the data type, predictive maintenance can be categorized into two sub-systems and considering the prediction techniques applied to each system, different approaches are proposed under two sub-systems [23]. Figure 2.5 shows the classification of the predictive maintenance approach and the prediction methods used under condition-based predictive maintenance.

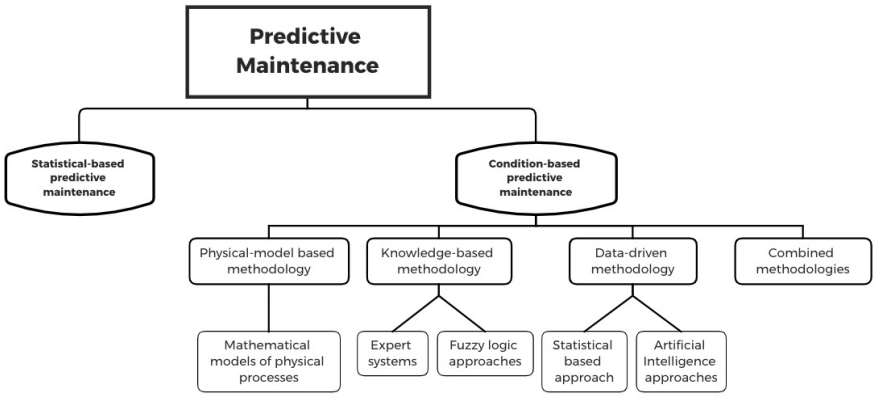

FIGURE 2.5: Predictive maintenance classifications

- Statistical-based predictive maintenance relies on historical data. By analyzing the data and extracting features to represent the failure symptoms, statistical models can be developed, showing the trends of potential defects. This method is also viewed as a part of data-driven methods under condition-based predictive maintenance. Statistical approaches such as Kalman Filter [24], neural network modeling [25], continuous hidden Markov model and support vector machine [26] are performed to predict RUL (Remaining Useful Life) to solve and conduct the health prognosing.

- In contrast to historical data applied on statistical-based predictive maintenance, condition-based predictive maintenance [27][28][29] uses real-time sensor data to prognosticate breakdown for components of the equipment. Four types of predictive methods can be applied to support condition-based predictive maintenance [15].

  Physical-model based methodology Physical modelling refers to use of mathematical models to represent physical components [30][28]. Real-time data helps us recognize the next potential stage as well as existing failures in the current stage. This methodology is easy to understand and can achieve high accuracy since it is based on an actual physical model [30]. However, for complex systems, simplifications are required and cause reduced functionality to detect all faults. In addition, the physical model must be updated as the actual model changes [31].

  Knowledge-based methodology This method has been divided into the expert system and the fuzzy logic approach [28]. The expert system adopts domain knowledge in a specific field and transfers knowledge into rules stored in the system. The fuzzy logic approach holds truth values with numbers 0 (completely false) to 1 (entirely accurate) and generates fuzzy rules for prognosis.

  Data-driven methodology In some research, statistical-based predictive maintenance includes multivariate statistical methods such as Bayesian Networks and is viewed as a subset of data-driven approaches. Artificial intelligence (AI) aims to simulate human intelligence in machines, and various methods such as machine learning or neural network [32] are applied to train a large amount of data and form models with which machines can execute human-like behaviours. This methodology does not require the knowledge of physical system and can detect complex faults by observing historical errors [30]. However, an accurate prognosis is made based on a lot of historical data and failure data. A lot of experiments are adopted to test its accuracy.

2.4 Expert systems

Considering the amount and type of data used in this thesis, the idea of developing an expert system is adopted. This requires acquisition of domain knowledge from experts, allowing us to generate maintenance rules with facts and consequences. That is, a lot of expertise and professional experience are stored in the system, which is a programming system for a particular area [33][34]. This system is then able to think and behave like

an expert, with the ability to reason and judge. It has the capability of simulating the decision-making process and making a prognosis or final diagnosis of discovered problems.

2.4.1 Components and features of an expert system

An expert system, also known as a knowledge-based expert system (KBMS), has been used since the mid-1960s and became one major branch of AI. There are several elements included in the system, such as knowledge base and inference mechanisms. All components work together to form the decision-making process and solve complex problems [35][36].

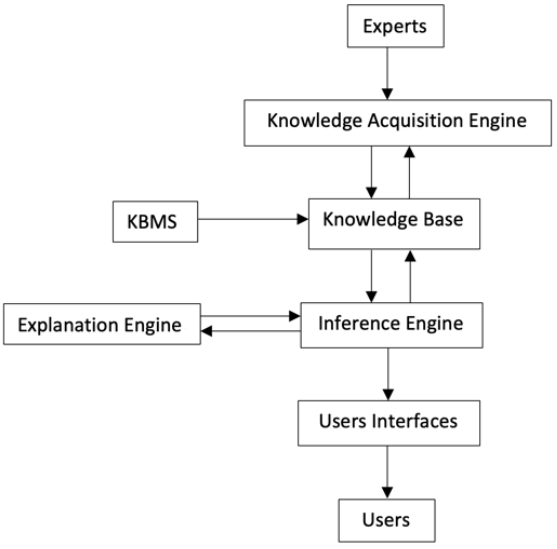

FIGURE 2.6: The architecture of an expert system

Knowledge Acquisition Engine This is used to obtain and modify domain knowledge by experts. The acquisition subsystem in this engine can help experts build or expand their knowledge base by translating domain knowledge into programming languages.

Knowledge Base It stores domain knowledge for understanding and solving problems. It also stands for an exact part of the real world through an acquisition module extracted by experts.

Knowledge Base Management Subsystem (KBMS) This is used to maintain knowledge in the knowledge base, for example, by inspecting the inconsistent errors as repetitions and redundancy.

Inference Engine This translates the knowledge stored in the knowledge base into facts so that the user interface can receive the facts and provide feedback to users. This engine offers IF-THEN rules. IF condition, THEN result. The condition represents facts of the real world, whereas the result represents a consequence that affects the real world. An inference engine acts as a rule interpreter to provide rule structure and methods to analyze and process rules themselves; it primarily works with one of three modes: forward chaining, backward chaining or hybrid chaining.

Explanation Engine tells users how the conclusion has been reached and provides a detailed tutorial to explain users the system's action.

Users Interface provides user-computer communication facilities like graphical interface and toolboxes. Thus, rules can be converted to a user-friendly level.

According to the architecture of an expert system in Figure 2.6, the inference engine looks for matched expertise in the knowledge base based on the input information and a conclusion from the knowledge base is provided to the inference engine and displayed to users as facts. In general, the knowledge base and the inference engine are the two main parts of an expert system. A crucial feature of expert systems includes the separation of the knowledge base from the inference engine. The system allows the knowledge to be modified and new knowledge to be added and continuously improves the performance of the system during operation.

As stated above, an expert system must have the capability of providing expert-level suggestions, explaining the system's behaviours and acquiring knowledge from experts [34][36]. A high-performance expert system should have the following characteristics.

Heuristic Heuristic knowledge acquisition means to use standard expertise and intuitive judgement to make inferences and associations and answer problems.

Transparency enables users to communicate with each other without knowing the structure of the expert system and understand the expertise and inferences stored in the knowledge base. The system can also explain the behaviour of the system itself.

Flexibility The separation of the knowledge base from the inference mechanism enables the system to continuously accept new knowledge, thereby ensuring that the knowledge in the system is continually increasing and updating to meet the needs of a business.

2.4.2 Building process of an expert system

Before establishing an expert system, we must first understand the approaches to represent each component. Among all components mentioned in Figure 2.6 of the last subsection, there are three components in the system that have classifications and need to be analyzed based on the requirements.

1. Knowledge Base [37]: This component stores the expert disciplines and can construct IF-THEN rules to represent stored knowledge. It mainly consists of two kinds of rules: active rules and deductive rules. Deductive rules are used to make inferences from existing rules or stored data and generate new information to solve the problem. Active rules are based on expertise and provide reactive behaviour [38]. When predetermined events happen, the consequences of the knowledge will be executed.

2. Inference engine: An inference engine is used to deduce the expert knowledge from the knowledge base and then reach a conclusion [39]. It contains three inference mechanisms. Forward chaining uses the existing knowledge and incoming information to arrive at results. In contrast, backward chaining poses a hypothesis and then looks for evidence that either supports or refutes its validity. Hybrid chaining is a combination of forward chaining and backward chaining. It uses forward chaining to determine the conclusion of the proposed hypothesis and tests the hypothesis by using backward chaining.

3. Knowledge representation [40][41]: Common knowledge representation methods can be divided into production rules, frames, semantic networks and processes. The production rule is the most popular expression method for expert systems. Expert systems represented by production rules are also called rule-based systems or production systems.

After being aware of the factors included in each component, the second step is to choose proper methods depending on the requirements of the business and design the framework according to the architecture of the expert systems. Below are some aspects of designing the three components:

1. The design of a knowledge base [37]: this mainly consists of three processes. The first is to confirm the type of rules based on the acquired knowledge, then choose the knowledge representations. This is followed by designing the knowledge base management system with functionalities such as creating, deleting and updating.

2. The design of the inference engine: the core of developing the inference engine is to determine the inference mechanism and inference algorithms, such as various search algorithm.

3. The design of the user interface: in the system, there are two interfaces provided for two purposes. While the expert-user interface is used for explaining the information and interpreting the conclusion, expanding the knowledge and maintaining the system relies on the development of the expert-expert interface.

A thorough understanding of methods and design procedures gives a clear view of how to build a complete expert system based on the requirements. The most challenging task is to extract standard features in the knowledge and represent them in rules. Finally, they must be coded with programming languages. The functionalities of an expert system can be achieved by a database management system.

Traditional database system management is passive and only executes commands (e.g. select, insert, delete), which are applied for deductive rules but not for active rules. So, in order to support active rules, active database management systems (ADBMS) with active features provide mechanisms for experts to enter reactions of domain knowledge [42][38]. The detailed instructions of active rules are illustrated in the next section.

2.5 Active Rules

Given the requirements of the proposed predictive maintenance system, and because the main idea is to store failure events and execute actions with real-time sensor data, active rules are adopted to warn if faults have occurred during the operation. There are a large number of active rules, and different rules have their own content and scope, which are extremely hard to create and manage.

2.5.1 The components of an active rule

According to [43], [38] and [44], despite the levels of complexity, an active rule consists of events, conditions and actions. That is, when an event or several events occur, check conditions, and if satisfied, actions are executed. The complexity levels of rules refer to the types, operations and relationships of events, conditions and actions.

Events The type of events includes primitive events and composite events. A primitive event can be represented by a specific function such as an operation (e.g. insert, update), a clock (e.g. every 10 seconds) or a transaction (e.g. abort, commit). Composite events are some combination of primitive events and composite events. They may consist of multiple operators and one or more transactions.

Conditions Conditions refer to the state of rules, and they are optional. Without conditions, rules are adopted in EA format, and the action would be executed while predefined events occur.

Actions In a knowledge model, actions are defined as responses with predefined events and satisfied conditions. They manifest in different forms, such as triggering another rule, updating a value or directly returning the results.

With ECA (event-condition-action) rules, the system can process a particular performance automatically under specific circumstances.

2.5.2 Rule meta-model

A rule meta-model [45] is responsible for presenting the relationship between rules and metadata and between different rules.

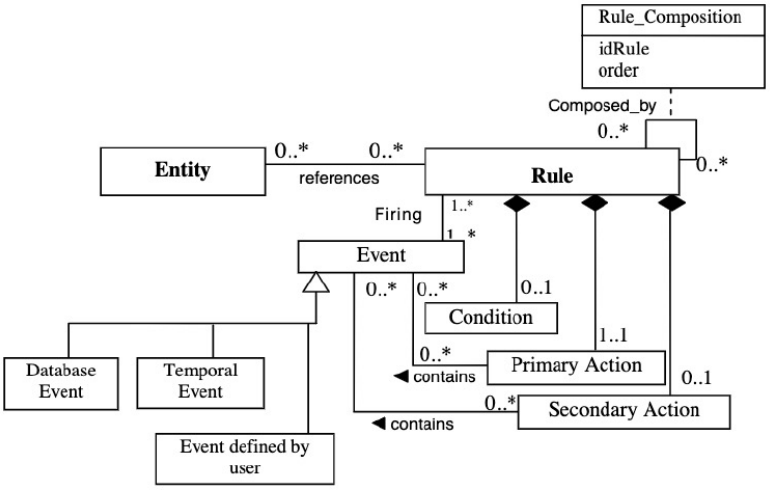

FIGURE 2.7: Rule meta-model

Figure 2.7 illustrates a UML class diagram and shows all related interactions among rule elements. Two classes, entity and rule, are included in the picture. The attribute of an entity could be defined as referencing a field of zero or many rules and vice versa. A rule is composed of one or several events, none or one condition, one primary action and none or one secondary action. Events are not only defined by users but have database

events and temporary events. Regardless of the type of event, if the event occurs, this rule would be triggered, and the condition is evaluated. Finally, the satisfied condition must trigger the execution of primary action, with the possibility of a secondary action that can be executed in the event the condition is not satisfied. The rule class contains an id field as its unique identifier. The relationship 'composed by' implies that a rule can be composed by other rules following a predefined order. This predetermined order decides the next rule after the current rule is fired [46][45].

2.5.3 Execution Model

Combined with the above relationship stated among components, Figure 2.8 describes five phases of a rule's processing.

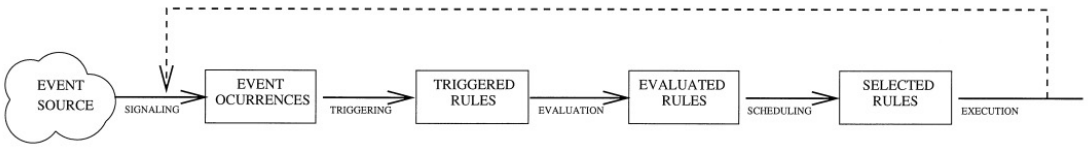

FIGURE 2.8: Steps of a rule execution

Signalling This phase depicts that the occurrence of events is due to some changes in the event source.

Triggering Rules are triggered because corresponding events took place.

Evaluation Check that the conditions of triggered rules are satisfied. If more than one rule satisfies the conditions, these rules are called *conflict set*.

Scheduling For *conflict sets*, rules must follow the predetermined order to process.

Execution The related actions are executed during this phase and may cause other events to be signalled.

2.5.4 Active database features with active rules

Compared to traditional DBMS, ADBMS overcomes the specific request execution in the passive aspect, applies the reasonable characteristics and operating rules of artificial intelligence, actively adopts object-oriented technology, and finally achieves the implementation mechanism of this system. In this way, an ADBMS with active features such as stored procedures and triggers can automatically give responses for events and incoming data by monitoring active rules.

ECA rules, as a widely accepted model, have five attributes. Except for the primary attributes (events, conditions and actions), E-C coupling and C-A coupling are also viewed as attributes of ECA rules [47].

E-C Coupling This coupling mode is used to associate the time when events are signalled with the time when conditions are evaluated.

C-A Coupling This coupling mode is used to associate the time when conditions are evaluated with the time when actions are executed.

There are three coupling modes for E-C coupling, as well as C-A coupling. Here, we use E-C coupling to illustrate the three modes:

1. *Immediate:* As the word implies, when corresponding events occur, conditions are evaluated without delay in the same transaction.
2. *Deferred:* When rules are triggered by the occurrence of events, evaluation of conditions are delayed until the termination of the transaction.
3. *Separate:* When events occur, conditions are evaluated in a separate transaction.

The modes of C-A coupling are the same as E-C coupling.

2.5.5 The operation of rules

In ADBMS, rules as database objects can be applied to create, update and delete. In addition, some special operations can affect the processing of rules.

Fire Also known as activation operation and ignition operation, it is mainly used for evaluating conditions. When the conditions are met, regular actions are performed. The execution of this operation depends on the corresponding coupling mode.

Enable The rule is put into an active state that can be activated.

Disable This operation is used to put the rule into a sleep state that cannot be activated.

Moreover, the operation of a rule should meet the requirements of transaction semantics. For example, any transaction that operates on the rule must have a corresponding

lock. Except for the fire operation that requires a read lock, all other operations on the rule must have a write lock.

Maintenance Strategies	Description	Pros	Cons	Costs
Breakdown Maintenance	Performs after a failure or breakdown occurs	1. Simple to understand and use 2. Requires minimal efforts and time 3. Independent with any sensors or software	1. High possibility of unscheduled downtime 2. Overtime labor 3. Unnecessary interruption 4. The failure may be dangerous, cause loss of life or public assets	Inexpensive to develop but comes with a vast amount of costly downtime
Preventive Maintenance	Performs based on a time schedule, can be viewed as standard practice	1. Reduces the risk of unscheduled downtime 2. Protects the loss of a bus 3. Improves work efficiency of buses and technicians	1. Can be unsuitable for managing a lot of buses 2. Unnecessary inspection and cost	Inexpensive to implement and 10%-30% more effective than breakdown maintenance
Predictive Maintenance	Performs before a failure happens by gathering data from condition-monitoring sensors	1. Maximizes the work efficiency and the lifespan of assets 2. Avoids unnecessary interruption and unscheduled breakdown 3. Requires little effort for hard-to-inspect parts	1. High requirements for software skills and condition-monitoring sensors 2. Time-intensive to set up and implement	Expensive to purchase related sensors and analytics software, but at least 10% more cost-effective than preventive maintenance

TABLE 2.2: Comparisons among three maintenance strategies

Chapter 3

IoT-based Predictive Maintenance System for Fleet Management

This chapter presents the overall architecture of IoT-based predictive maintenance for FMS, which our group is currently dedicated to developing [48][49]. So far, our team has proposed an IoT-based architecture to diagnose faulty buses, and algorithms describing the processing phases of rules for one hybrid bus. In this chapter, the details of the FMS design and how this work has been carried out will be introduced and explained.

3.1 Related contributions in fleet management system

Fleet management includes a series of functions, like vehicle maintenance, accident management, speed management and vehicle tracking. A capable fleet management system (FMS) can gather vehicle data, track the location and direction of movement of each vehicle in a fleet in real-time and report this information for users. This section presents relevant work on designing architectures to optimize performance of a fleet or transportation system.

Current research on fleet management systems aims to improve the maintainability or availability of transportation systems (e.g. trains, aircraft and buses). Emanuele Fumeo et al. [50] studied the features of train axle bearings and predicted their remaining useful

life (RUL) to prevent failures by analyzing the sensor data under different loads and rotational speed. They proposed the online support vector regression approach by adding a model selection phase with conventional support vector regression (SVR) for conditional based maintenance. Finally, they adopted three sensor parameters, evaluated their method by analyzing the data streams and concluded the feasibility of implementation. In [51], a two-layer IIoT-BDA framework has been introduced by integrating industrial IoT (IIoT) technology and big data analytics (BDA) for monitoring offshore support vessels and reducing the maintenance cost of Northwestern Norway. IIoT components and Vessel BDA constitute the vessel layer for collecting sensor data, processing and analyzing the sensor data to monitor daily operation. The Land BDA layer provides the combined solution of cloud and CPU/GPU/FPGA platforms for maintaining and analyzing results. Tony Lee and May Tso [52] analyzed the operation of MTR, a railway operator in Hong Kong, and developed a universal sensor data platform as a smart railway application. The application is capable of plotting multiple sensor data and performing holistic analytics so that faults could be identified by observing real-time sensor data. Effective maintenance decisions are made based on the graphics.

3.2 Architecture of IoT-based framework

The proposed IoT architecture intends to support various applications such as vehicle information exchange, vehicle performance monitoring, vehicle tracking and diagnostics, and smart surveillance (i.e., driver behaviour and driver management). As discussed in section 2.1.2, the development of the proposed architecture follows the general layers and classifies the overall system into three nodes at a technical level, which is described in Figure 3.1 and are outlined as follows:

- **Perception Layer** This layer is implemented by a vehicle node (VN), which consists of a gateway installed on the bus, the SAE J1939 protocol, and wireless Internet communication. The gateway of the VN is responsible for collecting, analyzing and operating data and provides the interface for connecting with sensor networks of the fleet system via SAE J1939 protocol. Besides, the administrator can control the software in the gateway remotely, which requires wireless network connections such as Wi-Fi and LTE. The gateway is connected directly to cloud-servers for further predictive analysis. It ensures that data streams obtained from these devices and systems travel securely from the edge of the Internet to the middleware layer, enabling decision-makers to securely aggregate, share, and filter data for analysis.

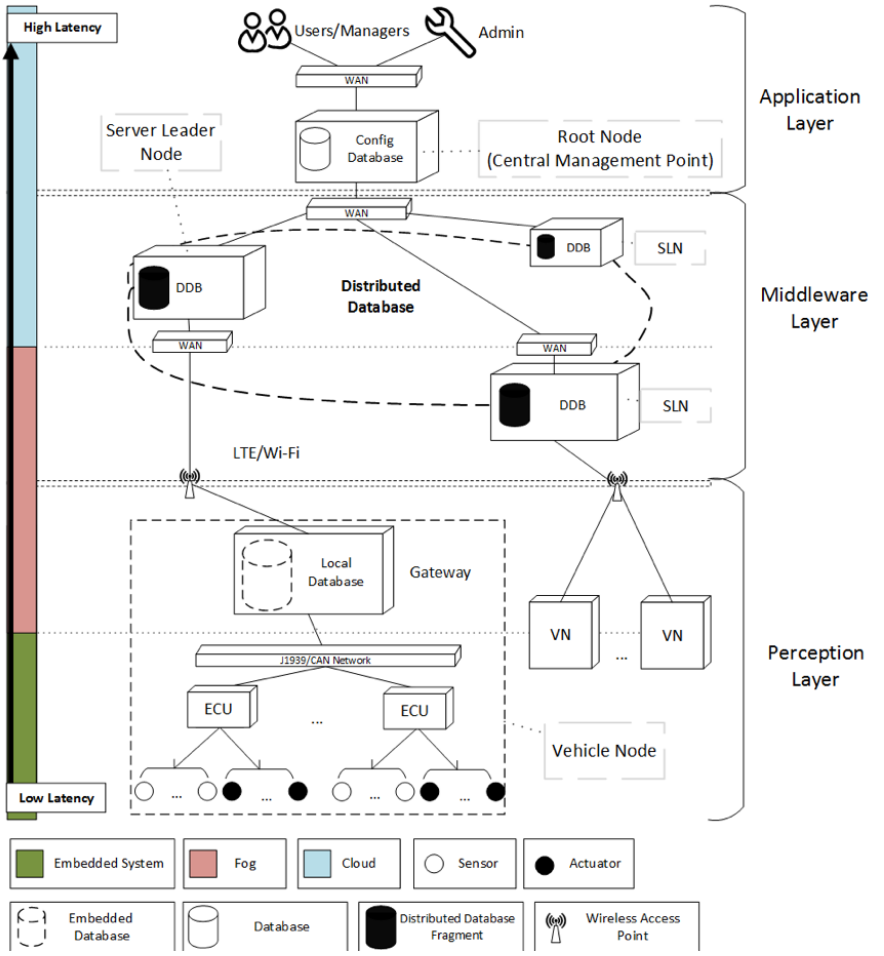

FIGURE 3.1: Generic IoT architecture of FMS

- **Middleware Layer** This layer provides a server leader node (SLN) with the power to manage a series of vehicle nodes in a fixed region. It has an MQTT broker to allow vehicle nodes to exchange information with each other as well as the SLN itself. The distributed database has been designed to collect the fleet data based on geographical location and increase the persistence of the storage, which facilitates further data analysis.

- **Application Layer** This layer provides root nodes (central points of administration) to manage the whole fleet management. It is equipped with a fleet system interface (e.g. desktop or mobile devices) to track information in other nodes via MQTT brokers and monitor fleet status remotely to achieve predictive maintenance.

3.3 Vehicle node system

Given that my research focuses on one hybrid bus, the architecture and functions of a vehicle node system need to be deeply understood. My colleague proposes the components and operations in the architecture of the system [42]. As shown in Figure 3.2, there are four modules (Data Acquisition Module, Active Rules Module, Rules Management Module and User Interface Module) comprising the vehicle node system.

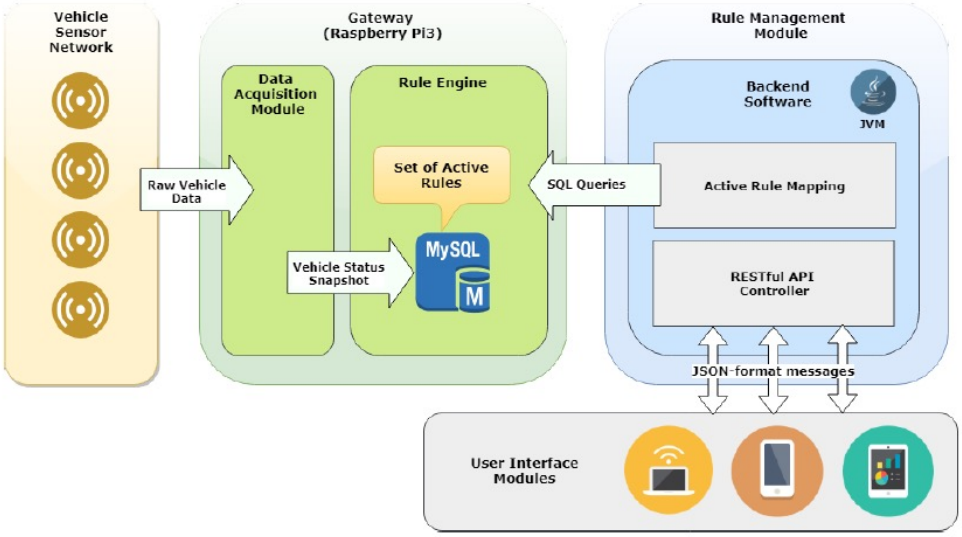

FIGURE 3.2: Architecture of a vehicle node

- **Data Acquisition Module** This module can communicate with vehicle sensor networks and read real-time vehicle data via the SAE J1939 protocol. Besides, it is responsible for decoding raw J1939 sensor data and converting it to parameter-value pairs. Finally, decoded data can be stored in the local database as vehicle status snapshots with the current time-stamp.

- **Active Rules Module** This module provides reactive behaviour for an incoming vehicle snapshot. The expert knowledge used for predicting failure is collected and transformed into active rules. The active database management system has active features such as triggers to store active rules and detect abnormal vehicle snapshots.

- **Rules Management Module** This module consists of a back-end software to manage rules. It allows administrators to map entities and rules into a database through back-end software instead of operating the database directly. In this way, the maintenance work of the database is reduced and SQL queries in back-end

software can search all information stored in the database. In addition, back-end software provides RESTful APIs to transfer JSON messages to web interfaces.

- **User Interface Module** This module is responsible for interacting with users through front-end applications (e.g. desktops or mobile devices). This interface displays the vehicle states and allows experts to create or update rules.

The work of my colleague also laid the foundation for active rules development. He analyzed runtime strategy and proposed two algorithms listed in Figure 3.3 and Figure 3.4 representing rule processing. Detailed information will be illustrated later, in combination with my work.

```
Data: Rule id and current sensor data snapshot
Result: Start of rule monitoring phase
if Rule is active then
    if Rule is monitored then
        if initial condition is satisfied then
            start monitoring Rule;
        end
    else
        monitor Rule for a Deferred condition;
    end
end
```

FIGURE 3.3: Main rule processing

The gateway device is like a black box installed on a bus and carries out several services to satisfy the requirements. Currently, our team has chosen the Raspberry Pi portable computer as the central computational unit in the gateway. For the rule management module installed on the gateway device, we need to consider the possibility of collecting information via J1939 protocol, which requires the J1939 ECU Simulator Board [53] produced by Copperhill Tech. For wireless communication, Raspberry Pi is limited to supporting GSM or LTE directly but provides an extension board called FONA 800 [54] to support GSM connectivity, and a USB-modem to support LTE. Backup power functionality is achieved by connecting a special uninterruptible power supply control-hat [55]. All configurations are listed in table 3.1.

```
Data: Data accumulated during monitoring process
Result: Fire result action
if monitoring termination condition satisfied then
    stop rule monitoring;
    if inner condition for firing result action is satisfied then
        fire result action;
    end
else
    accumulate temporary data;
end
```

FIGURE 3.4: Monitor rule processing

Functionality	**Configuration**
Main computational unit	Raspberry Pi 3 CPU: 1.2GHz 64-bit quad-core ARMv8 RAM: 1GB Connectivity: Ethernet, WiFi, Bluetooth
Cellular connectivity	Adafruit FONA 800 Shield
Backup power unit	UPS Pico - I2C Control Hat
SAE CAN J1939 connectivity	Copperhill SAE J1939 ECU Simulator Board

TABLE 3.1: Configurations of the gateway device

3.4 Requirements for the proposed system

For our purposes, a predictive maintenance system installed on a gateway device consists of an expert system completely encapsulated in ADBMS, a management system for operating ADBMS and a display screen to show the stored information in the database. The typical performance functions that I aim to achieve in this system are as follows:

1. The ability to store the sensor data of a bus into the database in real-time, provided that the sensor network on the bus is well connected with the system.

2. The ability to allow experts to input new failure symptoms based on all sensors shown in the user interface. Expertise should be proposed based on the existing sensors and can be translated into IF-THEN rules which contain events, conditions and actions.

3. The ability to detect failure once it happens and deliver the warning signal to drivers or technicians.

4. The ability to inform users of the real-time condition of the bus through the user interface. Users should also be able to check the logbook to view historical data of the bus's past performance.

According to the analysis of the vehicle node system and the algorithms of rule processing, my work applies the existing structure of the vehicle node system, models entities and operations using the IDEA methodology, and simulates the system in the software to test the performance. The IDEA methodology, as a comprehensive approach, solves various problems in the field of information system design. The most innovative strength of this method is using advanced database features, especially the object-oriented and rule-based paradigms [56], which correctly work for the development of expert systems. In addition, the IDEA methodology supports Chimera active rule language, which can design and implement the components and algorithms of active rules [57].

3.5 Analysis of the expert system

The core component of a gateway is to develop a well-performed expert system to prognose failures. An expert system is comprised of a large number of active rules representing domain knowledge and served by a relational database storing information related to the bus and components of active rules. For our purposes, the expert system must have the following functionalities:

- Storing real-time sensor data and information for each sensor.
- Creating and managing rules with the construction of the events, conditions and actions and representing rules as triggers.
- Dynamic operations on the rules, executed when input sensor data satisfies the conditions of a rule and reported failures for users.

From the perspective of an ECA rule, the type of events, conditions and actions are decided by the actual situation.

Events For primitive rules, the only possible event that can fire them occurs after new sensor data inserted. In contrast, composite rules must consider their sequence of

occurrence. In this situation, besides the first rule, all other rules are fired based on not only real-time sensor data, but the status of the previous rule.

Conditions Conditions are used for evaluating whether the sensor value is abnormal. Therefore, conditions must exist in rules and contain the title of sensors, operators and limited value. According to operators, conditions can divide into immediate conditions with operators GREATER THAN or LESS THAN and accumulated conditions with operators AVG or SUM. Accumulated conditions are responsible for computing data included in previous iterations. Based on the check times, conditions would be checked after a specific time, referred to as deferred conditions.

Actions There are four types of actions for execution. The first is to report failure, the second is to activate the next rule, the third is to start monitor processing and the final action is to return to initial states. For rules with start triggers only: first action: record failure; secondary action: return to the initial state. For rules with start triggers and monitor triggers: first action: start monitor processing; subsequent action: report failure or activate the next rule; third action: return to the initial state.

E-C Coupling The coupling modes of E-C coupling are immediate, deferred and separate. Our events for triggering the rule are new sensor data being inserted, and the previous rule occurring. The coupling modes depend on the classification of conditions. When events occur, conditions may be evaluated as soon as possible or checked over time. Nevertheless, if the check requires accessing of previous states, it is not easy to achieve within the same transaction. In this case, consider creating a new transaction to record the data during the time and evaluate the conditions in the new transaction.

C-A Coupling C-A coupling has the same coupling modes as E-C coupling. The difference between the two is that, in our case, when conditions are confirmed to satisfy, then actions are executed immediately. Therefore, the immediate mode is chosen for C-A coupling.

There are two rule processing algorithms illustrated in Figure 3.3 and Figure 3.4. The monitor trigger is used for carrying out deferred conditions, which require previous records. New incoming sensor data fires the start trigger first, and if this rule has deferred conditions, it is possible for the monitor trigger to be fired. The complexity of different rules enables us to simplify a rule that only consists of one start trigger and one monitor trigger. That is, a complex rule with multiple start and monitor triggers would be decomposed into several simple rules with one start trigger and one monitor trigger followed by a specific order.

Start triggers and monitor triggers are a part of a rule, which means that their components are the same as rules, including events, conditions and actions. Nevertheless, the functions of each component in start and monitor triggers have some differences.

1. *Start Trigger*: Start trigger as the main rule processing is responsible for detecting the occurrence of events. If this rule has a deferred condition, as the start trigger begins to evaluate the deferred condition, an action is executed to fire a monitor trigger. Otherwise, the start trigger acts as a rule.

2. *Monitor Trigger*: The activation of this trigger is due to the satisfaction of a deferred condition. The monitor processing creates a new transaction and relies on monitor meta-data. The results of executing the monitor trigger are the same as the actions of the rule.

3.5.1 Coarse analysis

The coarse analysis starts from the functional specification of the system, and by following the execution path, potential targets of the system can be identified. Therefore, according to each application behaviour mentioned above, a rough view of all possible entities participating in the system can be outlined as in Table 3.2.

Target	*Target Description*
Sensor	A sensor that can detect the value of a specific attribute such as temperature or moisture.
Rule	A rule holds a piece of expert knowledge and can be represented by a start trigger and monitor trigger.
Event	An event is a necessary component of a rule; its occurrence can fire the corresponding rule.
Condition	All conditions of a rule must be evaluated in order to execute corresponding actions.
Actions	An action is the response when conditions are satisfied
Start Trigger	The occurrence of events fire the start trigger to evaluate the conditions.
Monitor Trigger	For rules with deferred conditions, a new transaction is created for further operation over a period of time.
Failure	A failure log means a component of the bus is not working as expected
Log Book	The performance record of the system

TABLE 3.2: Target identification and description of the proposed expert system

From the description of the system, the main functions have been divided into a series of sub-functions established for proving internal coherence:

- *Static Configuration Update:* allows the expert to create and manage the expert system:

    1. When a new sensor is created, a related attribute with the same sensor title must be automatically created for storing sensor data. When a sensor is deleted, the associated attribute to store sensor data must also be deleted immediately.
    2. Rules are created by experts manually, including the construction of the start trigger and monitor trigger. Events typically have two choices for start triggers, either new sensor data incoming or the previous rule occurring. The sensor attribute of conditions must connect to existing sensors.
    3. The logbook automatically records every change of states of rules, including the rule being fired, the start of monitor processing, the end of monitor processing and the action being executed. It has a similar work principle to failure documenting.

- *Dynamic Configuration Update:* allows the system to update the status of rules according to the executed type of action:

  1. The initial state of a rule is enabled, which means this rule is able to activate. The occurrence of corresponding events fires the rule, and its state is automatically changed to fired.
  2. If this rule has a deferred condition and it is satisfied, its monitor processing will start and its state will automatically change to monitored.
  3. If the action event is to report failure, the rule will keep the disabled state until maintenance is completed. Otherwise, the rule will be automatically reset to the initial enabled state.

3.5.2 Schema analysis

By applying object-oriented modelling to design the persistent structure of the system, this subsection analyzes the attributes, operations and relationships of each class based on the review information in the coarse analysis.

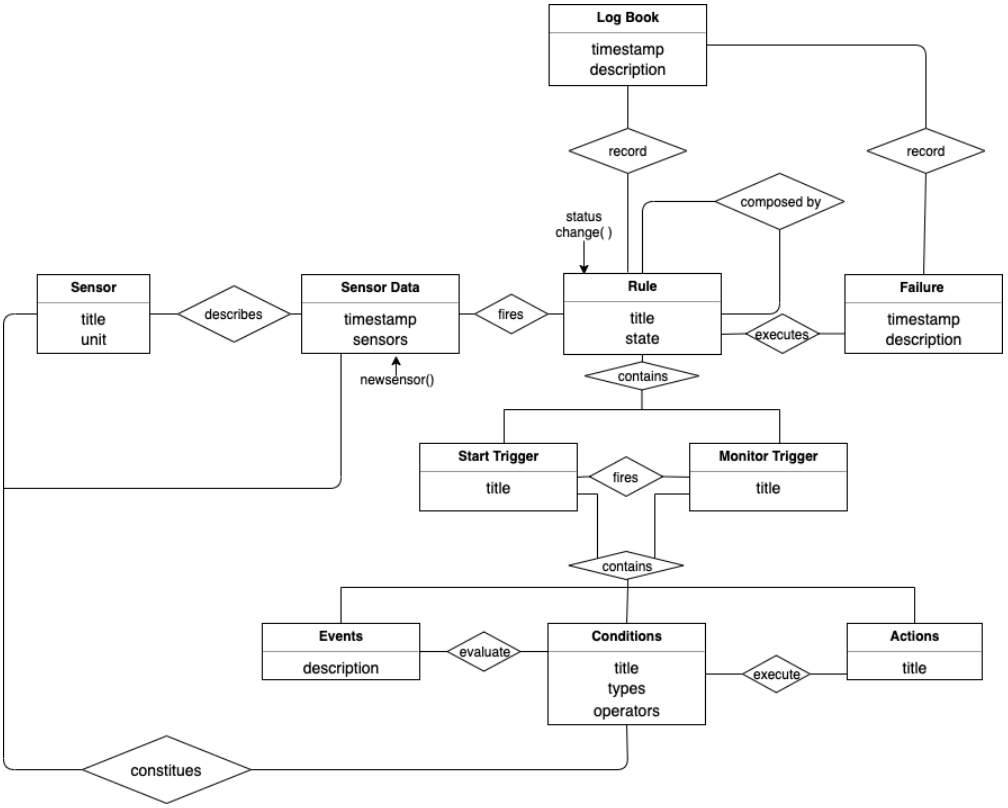

FIGURE 3.5: Object model with attributes, relations and hierarchies

Figure 3.5 builds an object model with all possible entities and relationships described above. Although the structure of the system is enormous and convoluted, there exist

operations and relationships without hierarchies. Combining the target identification and static and dynamic functions analyzed in the coarse analysis, the main entities and relationships include:

- *Sensor* describes sensor information such as temperature and its unit. It could describe the sensor attributes in *sensor data*.
- *Sensor Data* is used for supplying a place to keep old and incoming data over time. Incoming sensor data may trigger the associated rules. The newly created sensor automatically adds an attribute in *Sensor Data*.
- *Rule* can be represented as a start trigger and monitor trigger and used for documenting expert knowledge. The status attribute can help track the current performance of the rule. Composite rules have a particular order for firing.
- *Start Trigger* can be fired with the occurrence of incoming sensor data or the previous triggered rule.
- *Monitor Trigger* is fired because deferred conditions are evaluated.
- *Events* change the status of a rule from non-activation to activation. An event consists of new sensor data coming or previous rule executing.
- *Conditions* are classified into immediate, accumulated and deferred conditions. Among them, immediate conditions are composed of sensors, operators and sensor data. Besides, accumulated conditions and deferred conditions rely on the factor of time. The type of condition (immediate, accumulated or deferred) depends on check times and check methods.
- *Actions* are executed after the evaluation of conditions.
- *Failure* is recorded when a rule has been successfully executed.
- *Log Book* records the performance of the system, such as changes in rule status and failure reports.

The operations illustrated in the figure ignore generally used operations such as constructors and destructors. In the next section, these universal operations are designed and implemented using Chimera methodology with query language and data manipulation approaches.

3.5.3 Knowledge analysis

This step is to specify the integrity constraints of the object model by creating guidelines for checking data stored in the database and ensuring the correctness of the system structure. Two approaches, fixed format and generic format, are used to verify two types of constraints.

Fixed-format integrity constraints This method works similar to database conceptual modelling methods, for example, by using the entity-relationship approach. By presenting the object model, fixed-format integrity constraints need to consider:

1. *Class attribute constraints:* primary keys, unique attributes, Not Null attributes;
2. *Relationship cardinality:* one vs. many, optional vs. mandatory;
3. *Reference integrity:* establishes operations between two linked entities. For instance, when deleting a value of an attribute of an entity, the referred entity must delete relational value to ensure the integrity of data.

Generic integrity constraints Generic constraints consider semantic requirements such as dynamic operations and are expressed by natural language statements (static immediate, static deferred, dynamic immediate, dynamic deferred).

By analyzing the structure and operation in Figure 3.5, we added fixed-format and generic constraints to ensure the integrity of the system in Figure 3.6. Below is a description of these constraints.

1. One rule must have one start trigger, and the start trigger can correspond to many rules. One rule optionally has one monitor trigger, and one monitor trigger can correspond to many rules.
2. Since a rule consists of only one start trigger or one start trigger and one monitor trigger, the relationship between start trigger and monitor trigger is optional. Additionally, one start trigger can fire none or one monitor trigger, and one monitor trigger is fired by one start trigger.
3. Start triggers and monitor triggers have the same cardinality constraints with each component. One trigger includes one or more events, conditions and actions. In turn, one event, one condition and one action also associate with one or more triggers - the same constraints exist between events and conditions, as well as between conditions and actions.

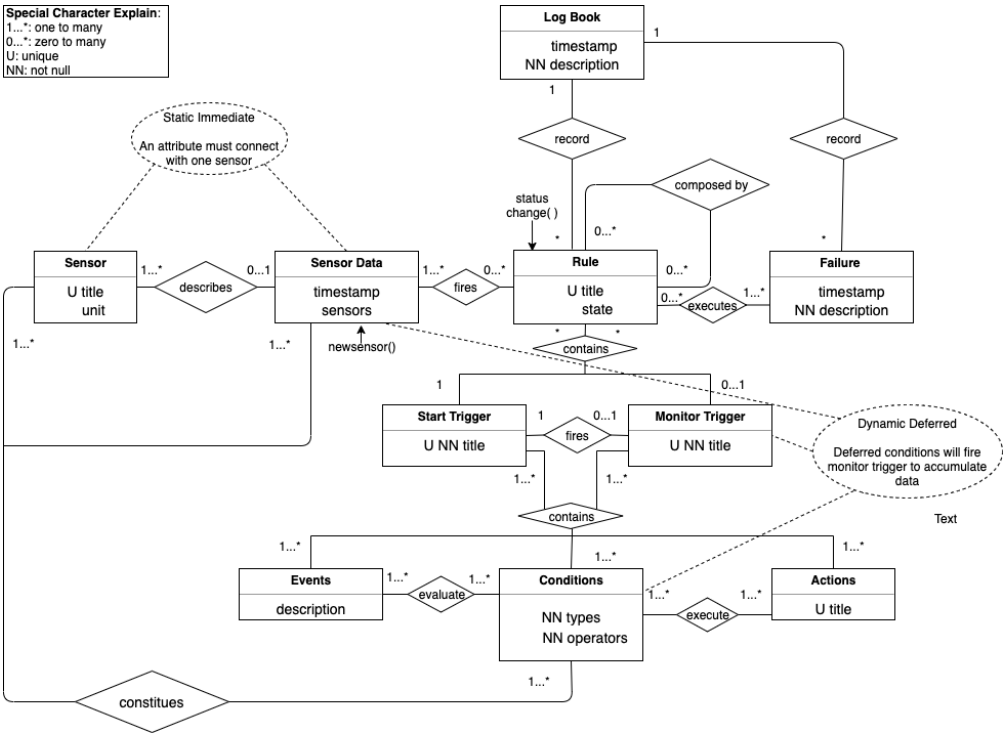

FIGURE 3.6: Schema with integrity constraints

4. At least one sensor and one piece of sensor data comprise the attributes of conditions.

5. The sensor attribute in *Sensor Data* must be linked to the related sensor information. Extra sensors do not connect with any attributes in *Sensor Data*.

6. Every piece of sensor data can fire rules with incoming sensor data as events, but not fire other rules with different events. A rule can also be evaluated with one or many rows of sensor records.

7. Every change of status of a rule and the action being fired including failure information must be recorded in a logbook. This logbook can document these specific performances of the system.

8. Only composite rules have orders to trigger the next rule. One rule may trigger several failure warnings or activate the next rule. One failure warning can be triggered by one or many rules.

Chapter 4

Design And Implementation an Expert System for Predictive Maintenance of a Hybrid Bus

After applying coarse analysis and detailed analysis, we have determined the structure and features of the system. This chapter illustrates the steps to transform the object model and dynamic operations to conceptual schemas using Chimera language and implement the Chimera model into MySQL.

4.1 Design of the expert system

4.1.1 Schema design

The goal of the schema design is to enrich the analysis with textual descriptions and provide a structural and behavioural basis for implementation afterwards. Given the graphical view provided in Figure 3.6 in the previous chapter, hierarchies are not included in the design. We continue to focus on the establishment of classes, operations and relationships.

4.1.1.1 Class design

There are two aspects of developing classes. First, we must determine which classes can be viewed as types and design types for attributes and parameters. Unlike classes that describe objects, types only include values and no identities. Below are some classes and in this instance they can be viewed as types.

1. Consider the classes of *start trigger* and *monitor trigger*, both reference classes of *events*, *conditions* and *actions* and are referenced from class *Rule*.

   - For start triggers, the *event* class only has a static attribute (description) for describing possible trigger activities (sensor data insertion and previous rule execution). For monitor triggers, only the event of evaluating deferred conditions is considered.
   - Similarly, the *condition* class does not have operations, but static attributes (sensors, operators, values, period and types).
   - The description of the final executed results also comprises the class *action*. However, if the class *action* is treated as type, a new class *failure event* must be created as an object which is illustrated in relationship design.

   Overall, according to Chimera language, all classes of *event*, *condition* and *action* can be defined as types and used as attributes in classes *start trigger* and *monitor trigger* written in Chimera below, see Figure 4.1 and Figure 4.2.

```
define value type event
    description of (type:string)
end;

define value type condition
    record of (sensor:sensor, sensor data: float, operator:string, type:string, period:
    integer)
end;

define value type action
    description of (type: string)
end;
```

2. Instead of being programmed as a Boolean value, the dynamic states of rules can be described as *enabled*, *fired*, *monitored* and *disabled*.

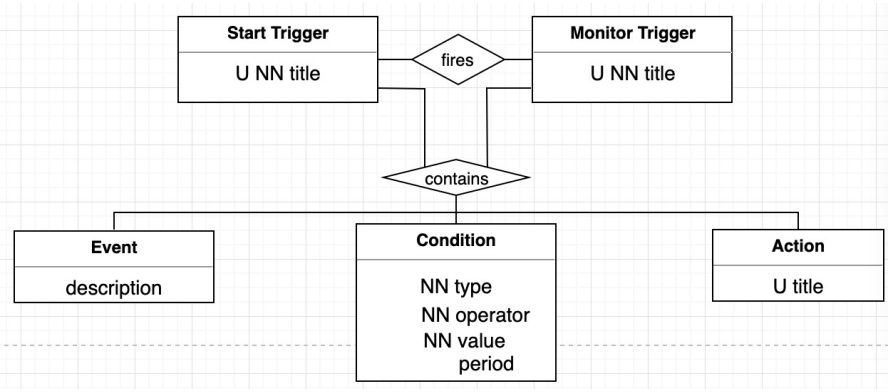

FIGURE 4.1: Original classes structures

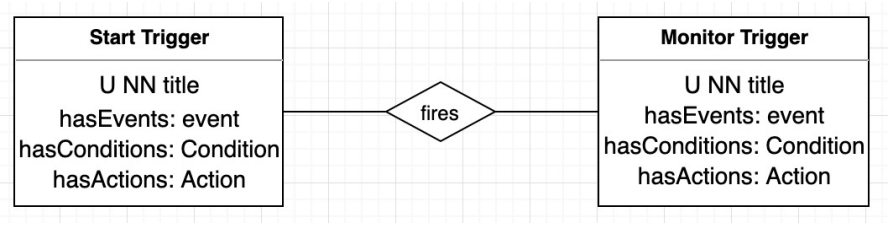

FIGURE 4.2: Transformation to types

```
define value type ruleStatus:
    enabled, fired, monitored, disabled
end;
```

The second aspect of developing classes is to identify the components of objects by combining or splitting classes. This step focuses on classes which are simply components of other classes. Therefore, considering the classes of *start trigger* and *monitor trigger*, most of their attributes contain one or more values, which is hard to implement and maintain. In order to conveniently process them in the future, we must split one start (monitor) trigger into several start (monitor) events that have the same attributes but include only one condition. Since a rule comprises of one start trigger and one monitor trigger, a rule is easily able to handle several start events and monitor events. The components of a start (monitor) event are the same as triggers, but the cardinality constraints have changed from one-to-one to one-to-many (Figure 4.3).

4.1.1.2 Relationship design

Chimera language does not support relationship design directly, like entity-relational design in the database, it can map relationships by adding attributes to relations or

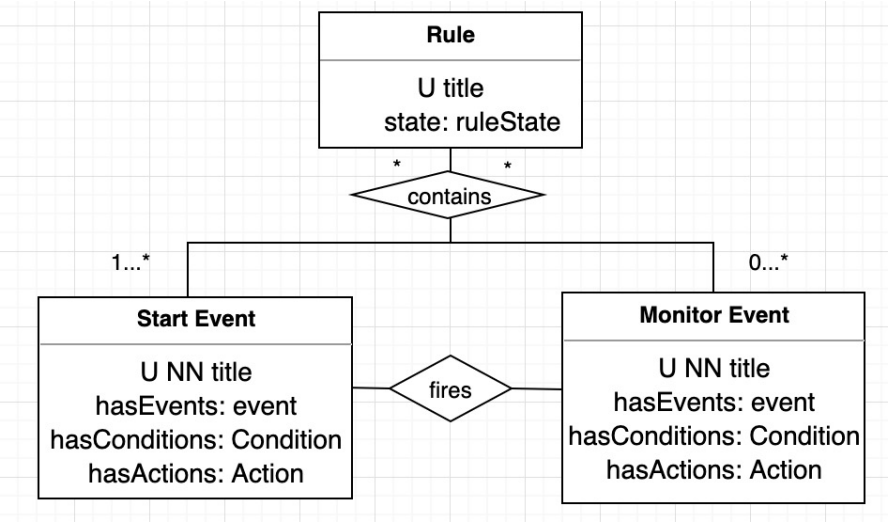

FIGURE 4.3: Triggers partitions

joining new relations.

After the class design, the current structure of the system has some differences in comparison to the original structure. In order to accurately specify the relationships of the system, a new schema is created based on the class design as shown in Figure 4.4.

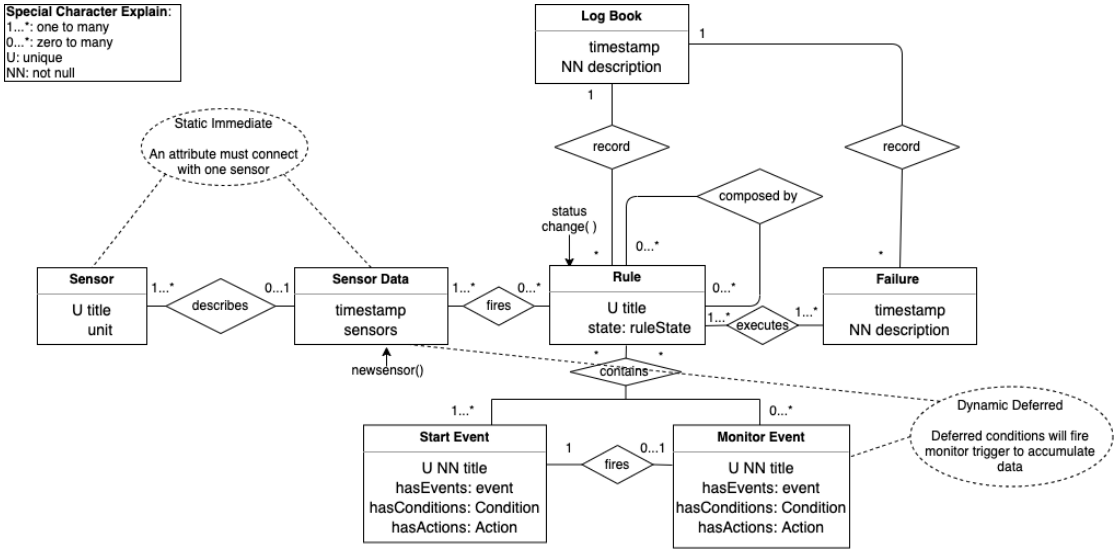

FIGURE 4.4: Simplified object model with attributes and relations

Several reference alternatives are offered depending on the relational directions. A single object-valued attribute as a *single reference* mainly applies to the one-to-many relationship and adds a new attribute related to the class with max-card one in the class with max-card many. This reference omits the trouble of adding a set of attributes and replaces the relationship with a new attribute and a dotted edge pointing. When it comes to one-to-one and many-to-many relationships, *coupled reference* pairs these two

classes and adds a new attribute related to another class in each class. This reference models traversal directions and points with each other using two dotted edges. *Bridge class* is responsible for analyzing the relationships with more than two classes and adding a new class to connect. The new class usually stores the primary key of both classes and has integrity constraints for ensuring the correctness of data flow.

There are eight relationships existing in Figure 4.4. The following statements illustrate the chosen reference for each relationship:

1. Relationship *describes* between sensor and sensor data is not mandatory and modelled as a single reference.

2. A rule must have at least one start event and possibly have some monitor events. Therefore, the relationship *contains* is treated as a coupled reference.

3. The types of events, conditions and actions of start events and monitor events provide linked attributes with other classes. Relationship *fires* between sensor data and rule applies two bridge classes, sensor and start (monitor) event. The insertion of sensor data as an event of start event class may fire rules, and rules need to extract several sensor data to check the conditions of start (monitor) events.

4. In addition, the other *fires* relationship is embodied in the action of a start event class and the event of a monitor event class - same processing as the relationship *composed_by*.

5. If all conditions of a rule are examined satisfactorily, the next procedure is to execute actions and report failure symptoms. For the relationship *executes* between classes rule and failure, a new class is needed for storing titles of all failure events and the value must be unique and not null. The cardinality constraint between failure class and failure event class is one-to-many: one failure event contains several failures, and only one failure is described in the failure event. Class failure uses a Boolean value to determine which failure event has occurred.

6. The two relationships *record* are both single reference. The class logbook is responsible for recording every change of rule status and failure warnings. However, each change is documented by only one piece of data in the class logbook. And the description attribute in the logbook contains the records of the system performance.

4.1.1.3 Operation design

In our analysis, possible activities involved in the system have been defined. With the exception of dynamic operations, four kinds of general-purpose utility operations are considered in the system: *constructors* for creating new objects with associated values, *destructors* for deleting existing object and connections with other classes, *accessors* for allowing the system to load and display required information and *transformers* for translating values as expected. For complex operations, composing of operations are supported by Chimera.

4.1.1.4 Updated schema after deigning using Chimera

Through the design of the above elements, a conceptual level of system structure is constructed in Figure 4.5. Based on the original structure, the current model enriches the definitions and adds more Chimera concepts by class, relationship and operation design. By dividing triggers into a series of start (monitor) events, a condition can be efficiently represented and linked with a sensor instead of a list of sensors. The relationship between class sensor data and class rule has been simplified with two existing classes: start event and monitor event as bridges. A new class called failure event is added to describe possible failure action and connect class failure and class rule as a bridge. These changes during design lead to some new attributes and new operations. The object classes and operations of the final schema are listed below.

```
define object class rule
attributes   title: string (20),
             start_trigger_name: string (20),
             monitor_trigger_name: string (20),
             start_events: startEvent notnull,
             monitor_events: monitorEvent,
             status: ruleStatus,
             strategy: string (20),
             deferred_time: integer,
             failure: failureEvent
key          (title)
operations   enabled(), fire(), monitor(), disabled()
constraints  immediate recordToLogBook
             immediate reportFailure
end;
```

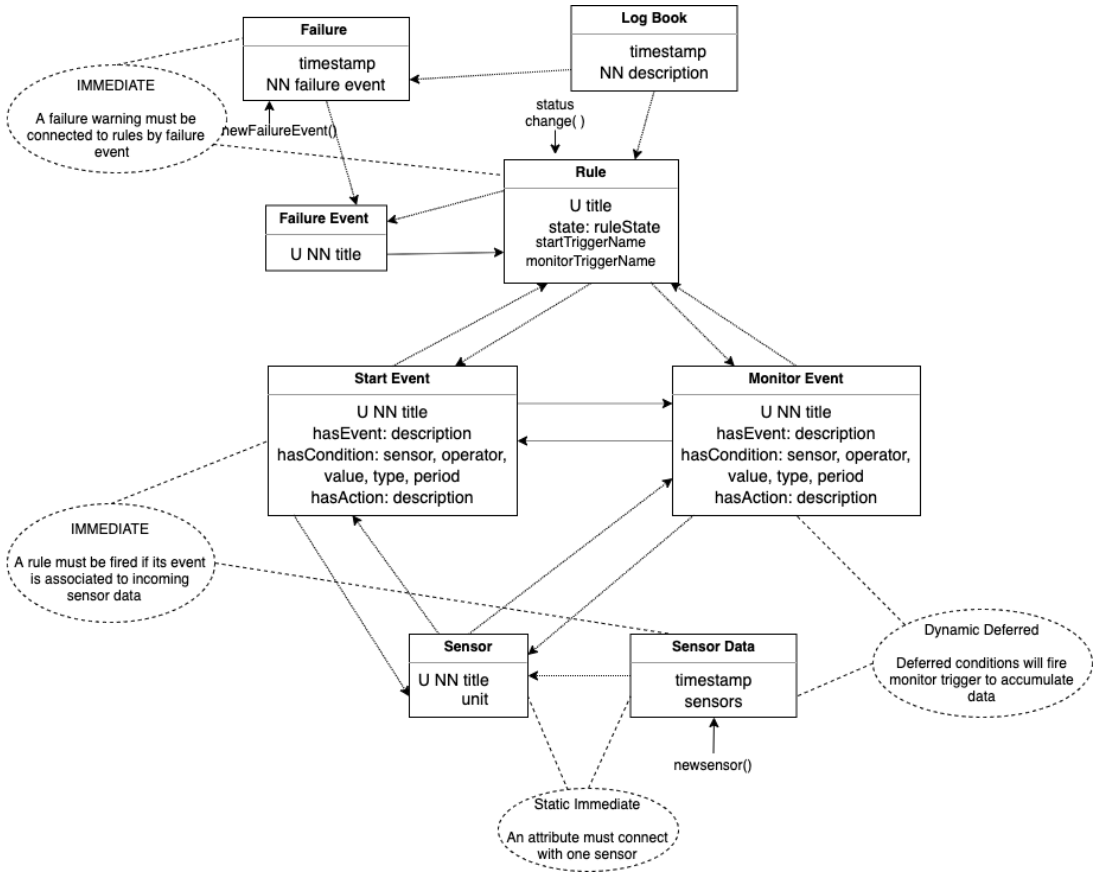

FIGURE 4.5: Object model expressed in the conceptual model by Chimera

```
define object class startEvent
attributes   title: string (20),
             rule: Rule,
             event: string (20) not null,
             sensor: Sensor not null,
             operator: string (20) not null,
             value: float not null,
             type: string (20) not null,
             period: integer,
             action: string (20) not null
key          (title)
operations   checkSensorData(), fireMonitorEvent(), changeRuleStatus()
constraints  immediate matchRule
             immediate fireBySensorData
end;

define object class monitorEvent
```

```
attributes   title: string (20),
             rule: Rule,
             sensor: Sensor not null,
             operator: string (20) not null,
             value: float not null,
             type: string (20) not null,
             period: integer,
             action: string (20) not null
key          (title)
operations   checkSensorData(), changeRuleStatus()
constraints  immediate matchRule
             deferred fireByTime
end;

define object class sensor
attributes   title: string (20),
             unit: string (20),
             start_events: startEvent,
             monitor_events: monitorEvents
key          (title)
end;

define object class sensorData
attributes   timestamp: datetime,
             sensor_title: float
operations   newSensorAttribute(sensor_title: float)
end;

define object class failureEvent
attributes   title: string (20)
key          (title)
end;

define object class failure
attributes   timestamp: datetime not null,
             rule: Rule not null,
             failureEvent_title: boolean not null,
operations   newFailureEvent_title (failureEvent: integer)
end;
```

```
define object class logbook
attributes   timestamp: datetime,
             rule: Rule not null,
             description: string (20)
operations   recordRuleStatusChange(), recordFailure()
constraints  immediate systemPerformance
end;
```

The deferred condition usually relies on the specified time (e.g. after 10 seconds) and does not depend on sensor data. So in a rule, one monitor trigger only needs one deferred time to fire. Therefore, the feature *deferred_time* can be viewed as one attribute of rule class. However, if during deferred time, the check also needs to access sensor data and evaluate the condition in advance, which means that there are two types of termination strategies. To meet this requirement, an attribute *strategy* is defined to display the termination type, either condition or time. As status change of a rule is the primary record in the class logbook, it is necessary to identify which rule has changed and add a new attribute *rule* in the class logbook. The attribute *rule* is also presented in class failure to show related rule information.

4.1.2 Active rule design

Active rules provide reactive behaviour and can be written in triggers using Chimera language. In our system, an active rule refers to reactions to the predefined failure symptoms to achieve the faults prediction.

4.1.2.1 Failure symptoms list

Based on the mechanical components of a hybrid bus shown in Figure 2.3, our failure symptoms could be grouped by 4 functions: engine management system, transport management system, HEV management system and accessory management system. 36 possible rules are organized according to the four functions and listed below. Technicians can follow each function to check if there are missing failure symptoms.

- **Engine Management System**

1. If fuel consumption exceeds the value of 12, and oil pressure falls lower than 35, check for problem:
   - check exhaust leak of engine system
2. If engine manifold intake pressure is below 40 kPa or over 90 kPa, check for problem:
   - check intake manifold gasket
3. If engine oil pressure drops to below 144.2 kPa and engine oil level is under MIN mark, check for problem:
   - check oil pump
4. If engine crankcase pressure is lower than 12.07 kPa or higher than 53.78 kPa, check for problem:
   - check oil leak
5. If aftertreatment exhaust temperature exceeds 455 degree C, check for problem:
   - check the fuel system
6. If engine coolant temperature is out of range 135.77/ 57.55 degree C, check for problem:
   - check the radiator
7. If the aftertreatment diesel exhaust fluid concentration is not in the range of 20% and 45%, check for problem:
   - check the diesel exhaust filter
8. If the diesel exhaust fluid level is less than 5% and above 0%, check for problem:
   - check the DEF system
9. If the diesel exhaust fluid level is at 0% and 'Engine STOP Led' flashes, restart the vehicle, check the engine speed. If the speed is more than 10 km/h, check for problem:
   - check the speed limit system
10. If fan speed is beyond 2860 rpm or under 450 rpm, check for problem:
    - check the fan drive system
11. If hydraulic fan motor pressure is reaching over 34473.79 kPa, check for problem:
    - check the hydraulic system
12. If within 15 seconds after starting the engine oil pressure is detected below 144.2 kPa, check for problem:

   - check the engine wearing

13. If 'coolant pressure' drops by 20% compared to the average of values acquired within last 5 seconds, check for problem:
    - check coolant leak
14. If power take-off oil temperature exceeds 169 degree C, check for problem:
    - check the cooling system
15. If the typical boost provided by a turbocharger is less than 6 pounds per square inch (psi), check for problem:
    - check the air leak
16. If the current value of engine RPM is detected to be 0, check for problem:
    - check the system circuitry

- **Transport Management System**

1. If transmission oil temperature falls lower than 122.5°C or goes over 292.5°C, check for problem:
   - check the transmission system
2. If transmission air supply pressure is detected below 682.581 kPa, check for problem:
   - check the vehicle air supply system
3. If clutch temperature is higher than 170 degree C, check for problem:
   - check the clutch
4. If the vehicle air pressure drops to 414 kPa, and the parking brake pedal position is still at 0%, check for problem:
   - check the parking brake
5. If the vehicle air pressure drops below 552 kPa, and the parking brake pressure falls lower than 552 kPa, check the emergency brake pedal position. If emergency brake pedal position is still at 0%, check for problem:
   - check the emergency brake
6. After 'emergency break' external event, monitor for 10 seconds, if speed reaches 0, check the stopping distance. If it exceeds norm by 20%, check for problem:
   - check the braking system

- **HEV Management System**

1. If DC/AC Accessory Inverter 1 AC Side RMS Voltage exceeds 255 V, check for problem:
   - check the inverter
2. If power input for batteries is out of range 650/280 voltage, check for problem:
   - check the electric wiring
3. If the alternator is putting out too much voltage (15+ volts), check for problem:
   - check the battery
4. If the peak power of battery exceeds 200 kw, check for problem:
   - check the battery
5. If voltage level for 24-volt line is detected to be out of range 24v, after 20 seconds, if average reading of 'battery temperature' exceeds 50c, check for problem:
   - check the battery
6. If the power supply voltage (12VDC) tolerance is out of 5%, check for problem:
   - check the motors

- **Accessory Management System**

1. If within 10 seconds engine coolant temperature exceeds value 180, and if 'transmission warning' signal appears, check for problem:
   - check the belt system
2. If pneumatic supply pressure is outside the range of 552 kPa and 1000 kPa, check for problem:
   - check the air brake system
3. If Cab A/C Refrigerant Compressor Outlet Pressure is over twice the ambient temperature plus 70 PSI, check for problem:
   - check the A/C system
4. If the tire pressure is lower than 193.05kPa, or higher than 627.42 kPa, check for problem:
   - check the tire
5. If the washer fluid level is detected under the MIN mark, check for problem:
   - check the reservoir
6. If the front axle is detected over normal curb height, check for problem:

   - check the Front kneeling system

7. If the vehicle air supply pressure is lower than 552 kPa, check for problem:
   - check the air supply system
8. If the 'KNEELING SWITCH' is set in the UP position and the 'MASTER SWITCH' is in the DOWN position, the vehicle is not being raised, check for problem:
   - check the pneumatic front suspension

The above rule examples follow the 'IF-THEN' pattern and clearly show the three components: events, conditions and actions. Most rules only consider start events without deferred conditions, while some rules must take into account deferred conditions and have monitor events. Therefore, when designing and implementing active rules, deferred condition is a very vital element to consider.

4.1.2.2 Active rules design procedures

The components of an active rule are closely connected with other classes, such as sensor and failure event. In order to acquire the necessary relationships and operations with other objects, in earlier subsections, active rules have been analyzed and designed into object classes start event and monitor event with three components, events, conditions and actions.

However, concrete features of active rules, like how to give a warning and how to trigger the next rule, are unclear. This subsection focuses on the implementation of active rules and introduces the transformation from expert knowledge to active rules.

From earlier analysis and design, the integrity constraints of an active rule could be classified into *immediate* and *deferred*. The main functionality of building active rules is to check if input real-time sensor data is abnormal. There are two events that can fire rules; one is the input sensor data, and the other is the previous rule's execution. Additionally, conditions are responsible for checking the sensor data. Actions either trigger the next rule or report failures. Lastly, the deferred rule contains delay time, and its termination relies on the condition or time. Due to the limitation of database management features, it is difficult to track data in the monitoring process. Therefore, a temporary monitor meta-data is created for storing sensor data during the monitoring process and evaluating the condition after or during the specified time period.

A trigger can be defined as follows:

```
define immediate trigger afterInsertSensorDataImmediate for sensor data
events      insert(S.value), sensorData (S)
condition   occurred(SE.sensor.value SE.operator SE.value),
            S.sensor_title == SE.sensor,
            sensorData (S), Self.startEvent (SE)
action      modify (Self.ruleStatus),
            modify(F.failureEvent == 1), F.failureEvent == SE.action,
            insert(log, L),
            failure(F), logBook (L), self.startEvent (SE)
end;

define deferred trigger afterInsertSensorDataDeferred for sensor data
events      insert(S.value), sensorData (S)
condition   occurred(SE.sensor.value SE.operator SE.value),
            S.sensor_title == SE.sensor,
            sensorData (S), Self.startEvent (SE)
action      modify (Self.ruleStatus),
            fire(Self.monitorTrigger),
            create (monitor meta-data),
            insert(log, L), logBook (L)
end;

define deferred trigger afterFireMonitorTrigger for monitor meta-data
events      execute(Self.startTrigger)
condition   occurred(Self.terminationStrategy),
            occurred(ME.sensor.value ME.operator ME.value),
            S.sensor_title == ME.sensor,
            sensorData (S), Self.monitorEvent (ME)
action      modify (Self.ruleStatus),
            modify(F.failureEvent == 1), F.failureEvent == ME.action,
            insert(log, L),
            failure(F), logBook (L), monitorEvent (ME)
end;
```

Statements above illustrate the operation of a simple rule with other classes. A simple rule contains one start trigger with the possibility of a monitor trigger. This rule does not need to trigger the next rule and does not have predefined orders to decide

the sequences of rules. Triggers rely on the sensors of the conditions. The deletion of sensors will automatically cause that the associated triggers are dropped.

Similarly, triggers of a composite rule can be defined as follows:

```
define immediate trigger afterInsSensorDImmediateComposite for sensor data
events      insert(S.value), sensorData (S)
condition   occurred(SE.sensor.value SE.operator SE.value),
            S.sensor_title == SE.sensor,
            sensorData (S), Self.startEvent (SE)
actions     modify (Self.ruleStatus),
            fire(R.order == Self.order + 1), insert(log, L),
            Rule (R), failure(F), logBook (L)
end;

define deferred trigger afterInsSensorDDeferredComposite for sensor data
events      insert(S.value), sensorData (S)
condition   occurred(SE.sensor.value SE.operator SE.value),
            S.sensor_title == SE.sensor,
            sensorData (S), Self.startEvent (SE)
action      modify (Self.ruleStatus),
            fire(Self.monitorTrigger),
            create (monitor meta-data),
            insert(log, L), logBook (L)
end;

define deferred trigger afterFireMonitorTriggerComposite for monitor meta-data
events      execute(Self.startTrigger)
condition   occurred(Self.terminationStrategy)
action      modify (Self.ruleStatus), fire(R.order == Self.order + 1),
            insert(log, L), failure(F), logBook (L)
end;

define immediate trigger afterPreviousRule for sensor data
events      execute(R), R.order == Self.order - 1, Rule(R)
condition   occurred(SE.sensor.value SE.operator SE.value),
            S.sensor_title == SE.sensor,
            sensorData (S), Self.startEvent (SE)
actions     modify (Self.ruleStatus),
```

```
        fire(R.order == Self.order + 1), insert(log, L),
        Rule (R), failure(F), logBook (L)
end;
```

Conditions for both simple rules and composite rules are the same. The differences between these two rules lie in how events occur and what results are executed. In addition, no matter what triggers are fired, according to 4.6, the status of a rule must be changed. A complete processing event goes from enabled (waiting for activation), fired (sensor data insertion or previous rule execution), monitored (optional, only for rules with deferred conditions) to disabled (reported failure, waiting for maintenance). During processing, fired and monitored states can be potentially interrupted, resulting in rules returning back to the original state due to conditions not being satisfied. The trigger event is usually the incoming sensor data, and this event happens frequently and almost at the same moment the system starts to evaluate the conditions. In this case, ignoring the fired state is allowed and has no impact on the system. Therefore, the fired state does not count as a rule status and is not recorded into class log_event.

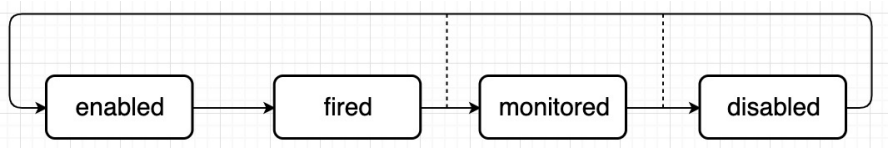

FIGURE 4.6: State charts of a rule

Given the rule set is complex, the transformation from natural language to Chimera language is necessary. In the context of the above faults, the following two examples are chosen to illustrate how to translate rules in triggers.

Example 1:
If fuel consumption exceeds value 12, and oil pressure falls lower than 35, check exhaust leak.

Example 2:
If Engine Coolant Temperature exceeds value 180 then if during 10 seconds 'transmission warning' appears, check belt

After acquiring knowledge, it is vital to recognize its essential components and operations and then design appropriate triggers based on the information. The list below illustrates these components and how to recognize them.

1. To recognize the sensors listed in the knowledge. Our system contains large sensor networks and faults are detected mainly based on the real-time process of sensor

data. Rule 1 mentions two sensors that need to be monitored, fuel_consumption and oil_pressure; rule 2 has sensors: engine_coolant_temperature and transmission_warning (all sensors may not be installed on the bus. The specific analysis for replacing nonexistent sensors is illustrated in Chapter 5). If we want to create the above two rules, the first step is to ensure these sensors are installed on the bus. Only if these sensors are present can incorrect data be detected and relevant rules be created.

2. To recognize the status of a rule. The first example is fairly straightforward since it does not have deferred time, which means it is an *immediate* rule without the monitoring process. The second example includes checking if transmission warning happens within 10 seconds after engine coolant temperature is detected to be above 180, so it is a *deferred* rule and must include the monitoring process.

3. To recognize the components of examples. Input sensor data fire in the case of both examples. For every incoming sensor data, associated sensor data is checked. When sensor data is abnormal, in the case of the first example, actions will be directly executed. As for the second example, it will start to monitor and verify next conditions. With the exception of sensors, conditions are composed of operators and values which have been acquired from experts.

Following the analysis, two rules can be defined as below:

```
define immediate trigger rule1_startTrigger for sensor data
events      insert(S.value), sensor data (S)
condition   occurred(S.fuel_consumption >12)
            AND occurred(S.oil_pressure <35), sensorData (S)
action      modify (Self.ruleStatus == disabled),
            modify (F.exhaust_leak == 1),
            insert(Self, "rule 1 is triggered", L),
            failure(F), logBook (L)
end;

define deferred trigger rule2_startTrigger for sensor data
events      insert(S.value), sensor data (S)
condition   occurred(S. engine_coolant_temperature >180)
action      modify (Self.ruleStatus == monitored),
            fire (Self.monitorTrigger),
            create (rule2_monitor_metadata),
```

```
            insert (Self, "rule 2 starts monitoring", L),
            logBook (L)
end;

define deferred trigger rule2_monitorTrigger for rule2_monitor_metadata
events      execute(Self.startTrigger)
condition   occurred(Self.strategy == condition)
            AND occurred(S.transmission_warning == 1)
            AND occurred(Self.monitor_limit_id == 10),
            sensorData (S)
action      modify (Self.ruleStatus == disabled),
            modify(F.belt == 1),
            insert(Self, "rule 2 is triggered", L),
            failure(F), logBook (L)
end;
```

4.2 Implementation

This section maps the conceptual models of the expert system written in Chimera language into ADBMS. We adopt MySQL as an effective database management system due to its stable and reliable advantages. More importantly, MySQL provides active features such as triggers to support active rules, which is beneficial for developing our expert system. Based on our analysis and design, this section classifies the mapping process into several steps: schema, operations and active rules.

4.2.1 Mapping to schema

Schema mapping is to translate the object model into relational tables, including the objects and relationships. Seven class objects have been defined in the final design and, assuming that each class can be mapped into a table, there are seven tables need to be created. In addition, corresponded object identities and references are important to consider when creating tables. All designed classes include reference classes and operations. Among them, the class *sensor* does not have any operations and is easily mapped into a table with identities and references.

For example, objects of a Chimera class *sensor* have been defined as:

```
define object class sensor
attributes   title: string (20),
             unit: string (20),
             start_events: startEvent,
             monitor_events: monitorEvents
key          (title)
end;
```

And the class now can be represented by MySQL as:

```
CREATE TABLE sensor
      (id INTEGER PRIMARY KEY,
       title VARCHAR (20) NOT NULL,
       unit VARCHAR (20),
       FOREIGN KEY (id)
          REFERENCES start_event ON DELETE CASCADE,
       FOREIGN KEY (id)
          REFERENCES monitor_event ON DELETE CASCADE,
      );
```

In Table sensor, MySQL provides an auto-increment identifier as a foreign key for the class to connect with other classes instead of unique titles in Chimera. In the context of reference classes, one sensor associates with one or more values (also called set-valued) of start_events and monitor_events. The ON DELETE clause is an optional definition of the database foreign key and used for the foreign key table to respond to the deletion of referenced values in the primary key table. There are four possible operations behind the ON DELETE clause: NO ACTION means no operations; SET NULL indicates that the corresponding field is set to null in the foreign key table; SET DEFAULT is similar to the SET NULL clause but set to the default value; CASCADE means cascade operations, that is, if the referenced field in the primary key table is deleted, the foreign key table is also deleted. The referential integrity of our case depicts that if this sensor is deleted, all related sets are deleted automatically by the clause ON DELETE CASCADE. Other classes of the object model can be mapped into corresponding tables similar to the transformation of the class *sensor*.

4.2.2 Mapping operations

The Chimera operations are mapped into a sequence of SQL queries with relevant objects and attributes. For example, the operation of newSensorAttribute of class *sensorData* is that when a new sensor is created in the table *sensor*, a new attribute with the same sensor title is automatically created in the table *sensorData*.

```
define operation newSensorAttribute (newSensor) for sensor
condition   true
action
      create (newSensor.title) for sensorData
end;
```

This statement can be mapped into the database by a trigger:

```
CREATE TRIGGER newSensorAttribute AFTER INSERT ON sensor
FOR EACH ROW
BEGIN
      @column = NEW.title;
      ALTER TABLE sensorData ADD @column float;
END;
```

4.2.3 Mapping active rules

The biggest achievement of our system is the ability to use MySQL triggers to represent Chimera triggers. MySQL triggers are attached to a table where events occur. Generally, a trigger is responsible for checking a new incoming value to ensure its correctness or updating the value by performing some calculations. In our system, using the first example which contains two simple conditions, the Chimera trigger is defined as:

```
define immediate trigger rule1_startTrigger for sensor data
events      insert(S.value), sensor data (S)
condition   occurred(S.fuel_consumption >12)
            AND occurred(S.oil_pressure <35), sensorData (S)
action      modify (Self.ruleStatus == disabled),
            modify (F.exhaust_leak == 1),
            insert(Self, "rule 1 is triggered", L),
```

```
            failure(F), logBook (L)
end;
```

And mapping into MySQL trigger is:

```
CREATE TRIGGER rule1_startTrigger
AFTER INSERT ON sensorData
FOR EACH ROW
BEGIN
    IF NEW.fuel_consumption >12 AND NEW.oil_pressure <35 THEN
        UPDATE rule SET ruleStatus = 'disabled' WHERE rule.id = 1;
        UPDATE failure SET exhaust_leak = 1;
        INSERT INTO logBook VALUES (1, 'rule 1 is triggered');
    END IF;
END;
```

Given that the monitor rules contain a deferred strategy and must build monitor meta-data tables, the action events need to consider more factors than immediate triggers. A trigger is activated depending on every row of new incoming data with a defined event, which is 'AFTER INSERT'. The main function of firing a start trigger is to check the monitoring conditions that have a termination strategy of either time or condition. Therefore, a monitor trigger needs to evaluate both conditions of monitor events and termination strategies.

```
define deferred trigger rule2_startTrigger for sensor data
events      insert(S.value), sensor data (S)
condition   occurred(S. engine_coolant_temperature >180)
action      modify (Self.ruleStatus == monitored),
            fire (Self.monitorTrigger),
            create (rule2_monitor_metadata),
            insert (Self, "rule 2 starts monitoring", L),
            logBook (L)
end;

define deferred trigger rule2_monitorTrigger for rule2_monitor_metadata
events      execute(Self.startTrigger)
condition   occurred(Self.strategy == condition)
            AND occurred(S.transmission_warning == 1)
            AND occurred(Self.monitor_limit_id == 10),
```

```
            sensorData (S)
action      modify (Self.ruleStatus == disabled),
            modify(F.belt == 1),
            insert(Self, "rule 2 is triggered", L),
            failure(F), logBook (L)
end;
```

The start trigger is mapped into MySQL as:

```
CREATE TRIGGER rule2_startTrigger
AFTER INSERT ON sensorData
FOR EACH ROW
BEGIN
    IF NEW.engine_coolant_temperature >180 THEN
        UPDATE rule SET ruleStatus = 'monitored' WHERE rule.id = 2;
        INSERT INTO logBook (ruleId, title) VALUES (2, 'rule 2 is monitored');
        INSERT INTO rule2_monitor_table (transmission_oil_temperature)
         SELECT transmission_oil_temperature FROM sensorData WHERE id = NEW.id;
    END IF;
END;
```

The 'UPDATE ruleStatus' and 'INSERT INTO logbooks' statements of monitor triggers are similar to the statements of start triggers. The difference between rules with monitor triggers and rules without monitor triggers lies in one of the actions, which is to achieve the evaluation of monitor conditions. Since the evaluation is initiated by new data being inserted into the monitor meta-data, it is necessary to extract the new incoming data in the sensorData table and insert the data into the monitor meta-data table. In this case, the monitor trigger is able to activate with the new incoming sensor data in the meta-data table. The statement can be translated into MySQL trigger with 'INSERT INTO...SELECT...'.

The monitor trigger is mapped into MySQL as:

```
CREATE TRIGGER rule2_startTrigger
AFTER INSERT ON rule2_monitor_metadata
FOR EACH ROW
BEGIN
    IF (Self.strategy == condition) THEN
     IF NEW.id <10 AND NEW.transmission_oil_temperature >450
```

```
        THEN
          UPDATE rule SET ruleStatus = 'disabled' WHERE rule.id = 2;
          INSERT INTO logBook (ruleId, title) VALUES (2, 'rule 2 is triggered');
          INSERT INTO logBook (ruleId, title) VALUES (2, 'rule 2 stops monitoring');
          UPDATE failure SET belt = 1;
        ELSE
          INSERT INTO logBook (ruleId, title) VALUES (2, 'rule 2 stops monitoring');
        END IF;
      END IF;
END;
```

The prerequisite of firing monitor trigger is to guarantee that the start trigger is under evaluation within 10 seconds. Therefore, the new id value and transmission_oil_temperature value stored in the rule2_monitor_metadata will be assessed together. Only if the two conditions are satisfied will the result reports that the relevant action event is triggered.

The above statements illustrate the transformation from Chimera language into MySQL queries. The trigger event defined in Chimera is to consider the insertion of every incoming data. This event can be expressed in MySQL as clauses: AFTER INSERT ON and FOR EACH ROW. When the event occurs, the evaluation of conditions begins. Active rules are generated in our system as IF...THEN expression, IF conditions, THEN actions. Rule 1 is responsible for checking the values of two sensors, fuel_consumption and oil_pressure.

The action of Chimera triggers are represented by *modify*, which can be replaced by UPDATE in MySQL, which means that the original value is updated to a new value. The replaced statements such as AFTER INSERT and UPDATE are also applied for rules with monitor triggers. And rules including monitor triggers, such as rule 2, need to analyze the monitoring time and termination strategy. To achieve the functionality of evaluating monitor time, the start trigger can select the new incoming data and insert it into the meta-data table. Also, the number of data records (id) in the meta-data table can measure the monitoring time. The different terminating strategy will cause different types of triggers. When the strategy is condition, the table will continuously evaluate the monitor events during the monitoring time. Conversely, monitor triggers with time as their strategy will evaluate the conditions of monitor events after monitoring time. The meta-data table would check the number of records, and only if the id is outside of the defined monitor time would the table then start to evaluate all conditions of monitor events.

4.3 The establishment of the whole system

Based on the requirements of the expert system, a detailed structure and accurate instructions of mapping active rules into triggers have been illustrated in the previous sections. The next step is to build the whole system, including the front-end system and management system. The active database system is set up as designed, and some operations are expressed as functions in the back-end system to optimize the process of the whole system.

In addition to an attractive page design, most functions of the web interface are achieved, including the display of related information, such as sensors and rules, and basic operations, such as create and delete, for these tables. More importantly, the user interface provides a dynamic platform that allows users to observe the real-time performance of each sensor.

The back-end system built using Python mainly manages the database and achieves some operations in the database like resetting the rule status after repair and automatically adding a column in the table sensor_data when a new sensor is created.

Chapter 5

Experiments and Results

5.1 Empirical models of sensors

Thanks to the STO company's contribution, 662 MB of sensor data has been collected from a hybrid bus during 10 hours of regular running. After decoding, we found that over 1 million sensor values are adopted by 72 different PGN with 616 SPN. Since all sensor values are only measured in normal running mode with no occurrences of failure, the external situations that lead to an accident cannot be analyzed. The active rules are developed with sensors and limited maximum (minimum) values. Although the sensor data acquired during the 10-hours of normal running is not enough to analyze fault patterns, we can extract several features to represent how sensors act under typical driving conditions.

5.1.1 Sensor selection for building empirical models

In my work, sensor selection is accomplished from a mechanical perspective based on the common failures that frequently happen on hybrid buses. To achieve the functionality of the expert system, a few pieces of failure information are acquired from technicians in the STO company and related articles online.

However, the considerable number and complexity of sensor networks makes it challenging to transfer failure information to active rules. Some failure symptoms do not have direct sensor values and need to be represented by indirect sensors. There are two types of sensor values: switch signals and numbers. For example, engine torque mode with PGN 61444 and SPN 899, measures the state signal from 0001b to 111b. These

signals are all discrete values which indicates which engine torque mode is currently generating, limiting, or controlling the torque. Additionally, ambient air temperature with PGN 65269 and SPN 171, calculates the temperature of air surrounding the vehicle. These values are continuous with measurable ranges between -273 °C to 1735 °C.

Among the 616 sensors, a number of useful sensors are investigated in Appendix A for developing active rules based on the failure information.

Tables A.1, A.2, A.3 and A.4 list 25 existing sensors organized by different mechanisms of the hybrid bus and include the importance of studying them. The rule lists mentioned in the last chapter has considered the existing sensors as well as the non-existing sensors on the hybrid bus. The sensor lists provide reference for experts to generate complex rules into the expert system and increase sensors on the bus compared with rule lists.

5.1.2 Modelling for selected sensors

The ECU controls the behaviour of a hybrid bus with the computation of sensor values, such as the air-fuel ratio to start the bus. If a sensor value is detected outside of the normal range, the component where the sensor has located fails and requires maintenance. The empirical models for sensors are built for generating a clear view of how each sensor should behave when the bus is running. Features, including the typical value range, maximum and minimum value of each sensor, are analyzed based on the observed data.

To function optimally, each mechanism is working with numerous components. For example, an engine on a bus requires different components to work together, such as the intake manifold, fuel tank and coolant systems. So, in order to observe whether the diesel engine is in good condition, the system must detect whether the values of critical components of the engine are within a healthy range.

Figure 5.1 shows the data distribution of the sensor engine coolant temperature for 10 hours of run time.

This sensor measures the temperature of the liquid found in the engine cooling system. The X-axis represents the time interval, while the Y-axis is the value of the engine coolant's temperature. The value should be able to increase gradually during the process of starting the cold engine to the hot engine.

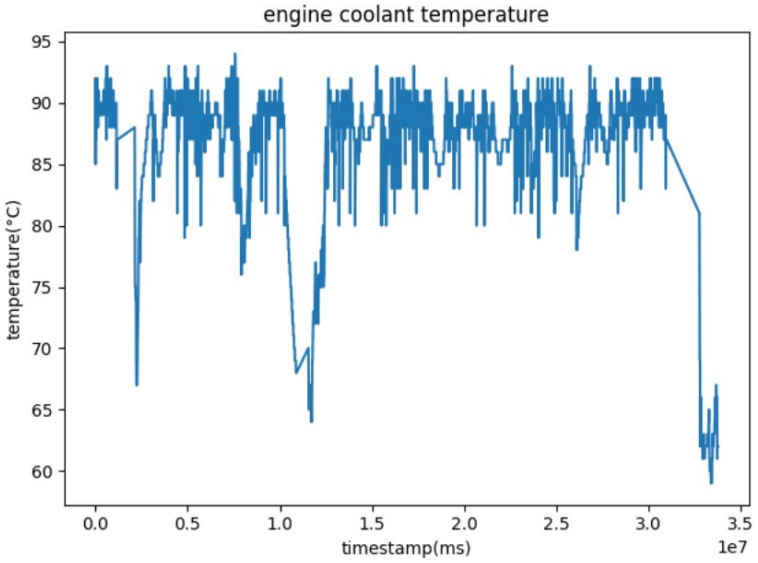

FIGURE 5.1: Data distribution of engine coolant temperature

PGN	SPN	SPN name	Min	Max	Mean	Median	std
65162	110	Engine Coolant Temperature	59	94	86.616659	88.0	5.875229

TABLE 5.1: Features analysis based on 10-hours normal operation data

The feature information of this sensor is described in Table 5.1. During the entire process, the value range of this sensor is from 59 °C to 94 °C and the average temperature value is 86.616659 °C. In this way, we could roughly understand the behavior of this sensor during normal running and compare it with actual fault data in future analysis. In addition, the data features calculated in table 5.1 provide references for experts when setting limited values of rules.

5.2 Expert system simulation

5.2.1 Assumption

Due to functional differences, different types of vehicles have their own failure patterns and standard maintenance policies. The hybrid bus information investigated above only applies to this specific bus type running under the STO. Therefore, when building active rules, failure patterns and maintenance policies for this bus type are adopted by experts in the STO.

The simulation process exists with the following assumptions:

- The system does not consider external events (such as warning lights, driver behaviour, etc.). Rules consist of sensors installed on the bus.
- The limited values of rules have been compared with empirical models of corresponding sensors and it has been ensured that they are not included in the normal value range or detected range.
- Simplify defined rules only containing one start trigger and monitor trigger. One start(monitor) trigger can be divided into several start(monitor) events.
- Assuming the interval time is 1 second, this means that the data is being injected into the database every second.

Based on the type of rules, the simulation of the proposed expert system is divided into two phases:

1. Verify the feasibility of simple rules, which only consist of simple conditions.
2. Verify the feasibility of monitored rules, which consist of simple and monitor conditions

The proposed system is managed by Python, which provides the RESTful API to build web interfaces and connect with MySQL. To ensure the performance of the database, the web interface can show all related information stored in the database (operating steps are illustrated in Appendix B). Through the interface, experts can conduct regular operations such as create, update and delete on sensors, events and rules, and observe the rule status by watching results.

The steps of creating sensors, action events and rules follow the attributes designed in Figure 4.5. The rules tested in the following subsections may not apply for the failure patterns of a STO hybrid bus. The information related to specific failure symptoms requires input by experts in STO.

5.2.2 The simulation of simple rules

Simple rules only contain immediate conditions, and if all conditions are satisfied, the action is executed. Example 1 listed in chapter 4 is the typical simple rule type, and it has been mapped into the MySQL trigger in the last chapter.

If fuel_consumption exceeds value 12, and oil pressure falls lower than 35, check exhaust leak.

The fuel_consumption refers to the fuel_level in the sensor list, which is the ratio of volume of fuel to the total volume of fuel storage container. After decoding and analyzing, useful features are extracted in Table B.1.

PGN	SPN	SPN name	Min	Max	Mean	Median	std
65276	96	Fuel Level	59.6	94.8	75.1308277	73.6	8.490350
65263	100	Engine Oil Pressure	100	380	176.571799	144.0	64.588253

TABLE 5.2: Features analysis of the fuel level sensor and engine oil pressure sensor

Due to the lack of design and maintenance manuals available to us for the hybrid bus, the size of the fuel tank is unknown. To measure fuel consumption, we need to choose a related sensor which indicates the fuel consumption. From the above chart, we can see that the normal value range of fuel levels should be between 59.6% and 94.8%. Therefore, I assume that if the fuel_level drops below 35%, the fuel_consumption exceeds the value 12. The failure symptom is translated into:

If fuel_level falls lower than value 35%, and oil pressure falls lower than 35 kPa, check exhaust leak.

Two simulated experiments have been run with 1000 sensor values to test the performance; this includes 10 abnormal data and 100 abnormal data, respectively. Given the injection time of 1 second, in the meantime, the system observes the new incoming sensor data and displays them in line charts on a web page.

If the sleep time is 1 second, the overall time to inject 1000 values is estimated at 16.67 minutes. The whole process was initiated at 22:36:31 pm and ended at 22:53:18 pm, which was precisely equal to the estimated time. The real-time line chart describing the fuel level sensor data changes is shown in Figure 5.2.

During the simulation, I disabled the rule status change function so that the system could consistently receive a warning with abnormal data. For the first experiment, Figure 5.2 has revealed 10 abnormal data. To verify the feasibility of the expert system, I checked the log_event and action_event tables for observing results, which are shown in Figure 5.3 and Figure 5.4, respectively.

The log_event table in Figure 5.3 mainly describes the rule status change with a specific rule id. The action_event table in Figure 5.4 shows the required maintenance

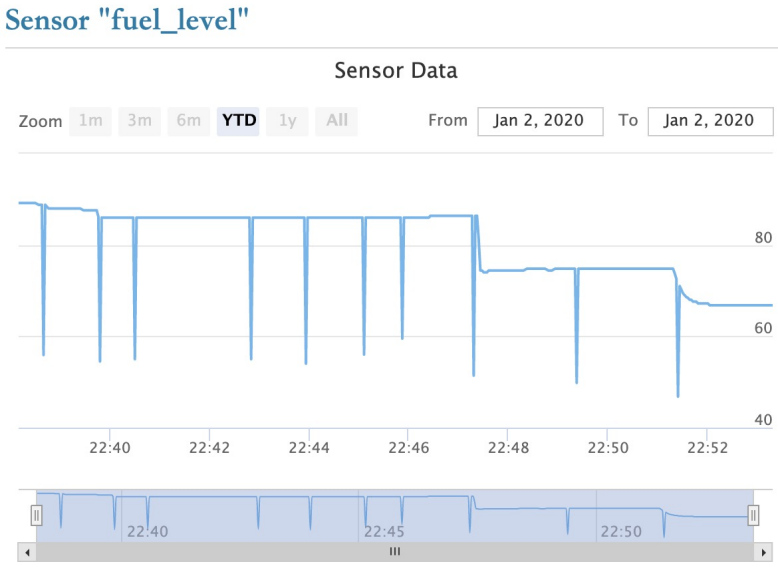

FIGURE 5.2: The fuel_level sensor data

	id	datetime	ruleId	title
▶	1	2020-01-02 22:38:39	1	rule triggered
	2	2020-01-02 22:39:49	1	rule triggered
	3	2020-01-02 22:40:30	1	rule triggered
	4	2020-01-02 22:42:49	1	rule triggered
	5	2020-01-02 22:43:57	1	rule triggered
	6	2020-01-02 22:45:07	1	rule triggered
	7	2020-01-02 22:45:51	1	rule triggered
	8	2020-01-02 22:47:18	1	rule triggered
	9	2020-01-02 22:49:23	1	rule triggered
	10	2020-01-02 22:51:25	1	rule triggered
	NULL	NULL	NULL	NULL

FIGURE 5.3: Results in the log_event table

action of the specific rule with value 1 after relevant rules have been triggered.

Comparing the timestamp between failure occurrence time and results acquired time, it can be determined that rule 1 has been triggered successfully. To check the reliability, the second experiment with the same rule but different amounts of test data has been conducted. 100 normal values have been selected randomly and changed to fit for the failure pattern. The line chart on the web page shows that the rule needed to be triggered 100 times and the results in the two tables proved that the rule had been triggered the same times.

Both experiments prove the 100% correctness of the system when performing simple rules. That is, if a failure symptom is able to be translated into rules only containing simple conditions, the system would detect the faults once they happen.

id	datetime	rule_id	exhaust_leak
1	2020-01-02 22:38:39	1	1
2	2020-01-02 22:39:49	1	1
3	2020-01-02 22:40:30	1	1
4	2020-01-02 22:42:49	1	1
5	2020-01-02 22:43:57	1	1
6	2020-01-02 22:45:07	1	1
7	2020-01-02 22:45:51	1	1
8	2020-01-02 22:47:18	1	1
9	2020-01-02 22:49:23	1	1
10	2020-01-02 22:51:25	1	1
NULL	NULL	NULL	NULL

FIGURE 5.4: Results in the action_event table

5.2.3 The simulation of monitored rules

A monitored rule consists of a start trigger and a monitor trigger. More importantly, a monitoring condition, which includes a specific monitoring time and strategy, connects the start trigger with the monitor trigger. Once the start trigger is fired, the monitoring condition starts to evaluate until the termination strategy is satisfied. Then, after checking the monitor trigger, and if all conditions of the monitor trigger are met, the rule is triggered, and related faults are communicated for maintenance.

The monitored rule applied for the simulation is the second example illustrated in the last chapter.

If Engine Coolant Temperature exceeds value 180°C then if during 10 seconds transmission warning appears, check belt.

The sensor transmission_warning is an external event that cannot be observed in this experiment. Instead, we consider the factor leading to the appearance of the transmission warning by using transmission oil temperature to represent the external event. So, the failure symptom can be replaced as shown below, and associated features are illustrated in Table 5.3.

If engine_coolant_temperature exceeds value 180°C then if during 10 seconds transmission_oil_temperature is over 300°C, check belt.

1000 typical values of engine_coolant_temperature sensor and transmission_oil_temperature sensor have been selected randomly for experimentation. Among them, two different types of test data were generated to compare with each other in four groups. The first data group aims to prove that when the start event is triggered, the rule

PGN	SPN	SPN name	Min	Max	Mean	Median	std
65262	110	Engine Coolant Temperature	59	94	86.616659	88.0	5.875229
65272	177	Transmission Oil Temperature	178.31	225.34	204.667270	206.31	6.907659

TABLE 5.3: Features analysis of the engine coolant temperature sensor and transmission oil temperature sensor

would stop monitoring after 10 seconds without satisfying the conditions of monitor events. The second group follows the failure pattern, which includes 10 abnormal engine_coolant_temperature data and one abnormal transmission_oil_temperature data.

Since the monitoring time (10 seconds) is expressed as the number of records inserted in the monitoring meta-data table (id = 1...10), a timer is set to truncate the meta-data table and avoid missing failures.

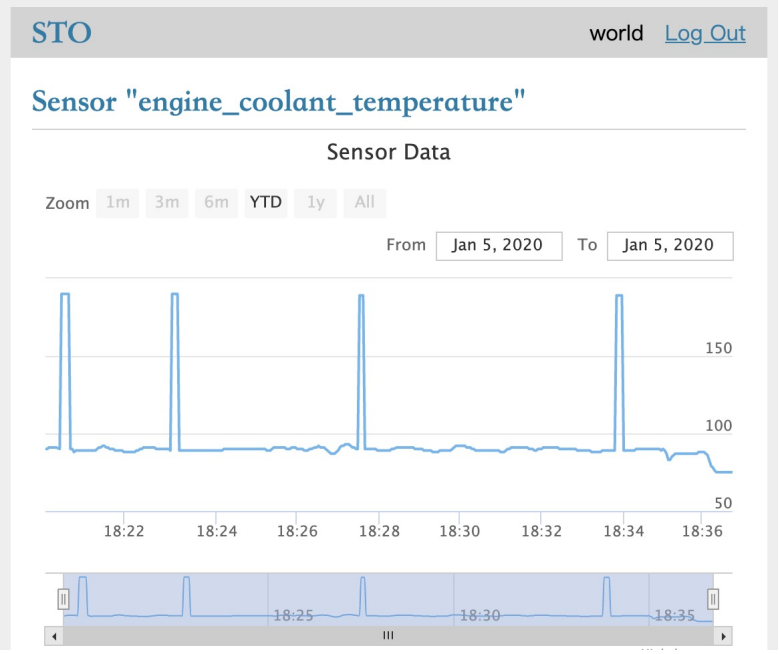

FIGURE 5.5: Sensor data in the engine_coolant_temperature table

Figure 5.5 shows the sensor data change during the experiment. At 18:20:29pm, the first abnormal engine coolant temperature data fired the first trigger and started to check the subsequent transmission oil temperature values for 10 seconds. Relevant results were recorded as Figure 5.6. Compared with the first failure symptom, the second fault happened at 18:22:57 pm. Figure 5.7 demonstrates that after 4 seconds, the transmission oil temperature was detected above 300°C. In the real world, the system would stop monitoring when the rule is fired because the rule status has been changed from monitor to disabled and waits to be reset after the maintenance event is done. Here to avoid the interruption of data being consistently inserted and to ensure we obtain the

complete feedback, the function of rule change status is disabled. Therefore, if a fault does happen within 10 seconds, the 'monitor started' statement would be inserted to show the real-time performance of the system.

id	datetime	ruleId	title
1	2020-01-05 18:20:29	1	monitor started
2	2020-01-05 18:20:30	1	monitor started
3	2020-01-05 18:20:31	1	monitor started
4	2020-01-05 18:20:32	1	monitor started
5	2020-01-05 18:20:33	1	monitor started
6	2020-01-05 18:20:34	1	monitor started
7	2020-01-05 18:20:35	1	monitor started
8	2020-01-05 18:20:36	1	monitor started
9	2020-01-05 18:20:37	1	monitor started
10	2020-01-05 18:20:38	1	monitor started

FIGURE 5.6: Rule changes recorded log #1

11	2020-01-05 18:22:57	1	monitor started
12	2020-01-05 18:22:58	1	monitor started
13	2020-01-05 18:22:59	1	monitor started
14	2020-01-05 18:23:00	1	monitor started
15	2020-01-05 18:23:01	1	monitor started
16	2020-01-05 18:23:01	1	Stop Monitoring!
17	2020-01-05 18:23:01	1	Rule triggered!
18	2020-01-05 18:23:02	1	monitor started
19	2020-01-05 18:23:03	1	monitor started
20	2020-01-05 18:23:04	1	monitor started
21	2020-01-05 18:23:05	1	monitor started
22	2020-01-05 18:23:06	1	monitor started

FIGURE 5.7: Rule changes recorded log #2

There are two more fault data groups with a type identical to the previous two test data groups occurring at 18:27:28 pm and 18:33:47 pm, respectively. Combined with the results shown in table action_event as shown in Figure 5.8, the two DateTime records are the same as the DateTime in the sensor_data table and log_event table. This proves that the system can successfully and correctly detect the failure symptoms by triggering associated rules.

id	datetime	rule_id	belt
1	2020-01-05 18:23:01	1	1
2	2020-01-05 18:27:36	1	1
NULL	NULL	NULL	NULL

FIGURE 5.8: Rule changes recorded log #2

5.2.4 Rule establish procedures through a web interface

The pictures in Appendix B show the procedure for creating a new rule (take rule 2 as an example). Below are some detailed instructions for creating a new rule:

1. To start, users must check if there are existing related sensors and events in the database. If not, users need to create new sensors and events, which are shown in Figure B.3 and Figure B.4. The creation of a new sensor and event is fairly simple since they only have the title as their attributes.

2. The new sensor and the new event will automatically produce a new column built in the sensor_data table and the action_event table, respectively.

3. Next, it is time to create a new rule. After filling out the information about the rule shown in Figure B.5, the system will transfer to the interface to create a new start event. The create procedure of a new rule contains primary information such as thr start trigger name and monitor trigger name.

4. Generally, a new rule is created with a new start event and possibly a new monitor event. If the system perceives that there is a monitor trigger name, it will transfer to create a new monitor event page next to the page of creating a start event. Two forms for filling out the attributes of start event and monitor event are displayed in Figure B.6 and B.7.

5. The current system provides the ability to build one start event and one monitor event directly. When the information of the start event and the monitor event has been filled out, the system will directly lead users back to the rule information page (Figure B.8). On this page, users can find all information related to the rule and add more start events and monitor events.

6. Based on provision of accurate information, the 'CREATE TRIGGERS' button is provided to build triggers in MySQL and then, the trigger statement (Figure B.9) is shown in the interface. When users see this page, it means that the trigger has been built as shown, also allowing users to check the trigger statement to verify correctness.

Chapter 6

Conclusion and Future work

6.1 Conclusion

This thesis aims to achieve predictive maintenance on a hybrid bus against the backdrop of IoT technology and FMS. Each hybrid bus carries a Raspberry Pi portable computer to collect the real-time sensor values via the J1939 SAE protocol. To achieve the requirements of storing data and detecting faults in real-time, an expert system has been developed in MySQL as the database management system. This system has been designed using the IDEA methodology and the Chimera language. Based on the analyzed entities and their relationships, a general architecture of the system has been modelled with rules and elements. The Chimera language updates the schema with enriched definitions and adds more concepts in the design of attributes and operations. The design of active rules relies on three essential elements: events, conditions and actions. The recognition of rule status urges the development of triggers.

MySQL supports completion of the functionalities of active rules such as triggers and stored procedures. It is important to implement the entities, relationships and operations and map to the database using active features. In Chapter 4, detailed procedures related to the design and implementation of the expert system have been illustrated to construct a complete architecture of the system.

The analysis of 10-hours of regularly running sensor data offers us a clear view of each sensor's typical performance and can be used for reference when building the active rules. After setting up the whole system, a series of experiments with two main rules were conducted to verify the feasibility of the system. Both rules are used to check if

simple rules and monitor rules can be triggered by abnormal sensor data. The results confirmed 100% correct reactions when detecting errors with triggers.

6.2 Limitations

There are several notable limitations in this research:

- **Lack of available data:** the data used for building empirical models and simulating the system was collected from a hybrid bus during 10-hours regular running. There exists missing or wrong data, which limits the sample size and is difficult to generate particularly accurate statistical models. The precise statistical features require a larger sample to ensure a representative distribution of the operating performance of a bus. This problem will be fixed when more data is collected and analyzed in future work.
- **Lack of generalized model:** this study only focuses on hybrid buses of STO, and only concerns the mechanisms of hybrid buses. The rule lists, which were developed from the mechanical perspective, might not be suitable to generalize to other vehicle models (e.g. diesel buses and electric buses). Specific breakdowns of different types of vehicles need to be consulted with technicians to acquire additional information.
- **Time and cost investments for developing and maintaining the expert system:** while the expert system can behave like a human with expert knowledge and make decisions based on the facts, it does need a great deal of effort to build and maintain. With more rules storing in the knowledge base, more errors may occur and lead to wrong results. Therefore, a large number of simulations are required to test the performance. Additionally, an expert system is incapable of adapting to changing environments where it is vital to reconstruct the knowledge base.

6.3 Future Work

In the context of this work along with the predictive maintenance work in FMS, some future directions are presented to improve the functionality, stability and performance of the system:

- **Experimentation with more rules:** a sensor list is organized in Appendix A which provides reference for experts in STO to raise various failure symptoms. The system needs to be tested in a real working environment on a hybrid bus with multiple rules encapsulated in MySQL.

- **Designing a general set of canonical classes of rules:** the rules forming the expert system for the running example permeating the IDEA methodology are structured into canonical classes that capture special doc aspects of the semantics of the electricity distribution domain[56]. We need to take a similar approach for structuring the rules of the domain of predictive maintenance for hybrid busses. Such a set of classes can then be transposed to related domains such predictive maintenance for fully electrical busses or fully gaz busses.

- **Accounting for external events:** current rules were developed with sensors installed on the bus, since we could achieve predictive maintenance with sensor values. In the future, the recognition of external events such as lights and driver behaviour can improve the performance of the system.

- **Applying self-learning technologies:** the current rule engine is developed with domain knowledge and it requires experts to continuously update it. Therefore, adapted technologies [58] should be applied to improve the performance of predictive maintenance.

Appendix A

Sensor List based on mechanical principles

Principles	Sensor Name	Significance
Pressure	Engine Intake Manifold 1 Pressure	Provides instantaneous manifold pressure information for the ECU of the engine
	Barometric Pressure	Reacts to sensitive changes in absolute atmospheric pressure
	Engine Oil Pressure	Detects oil pressure in the engine and sends this value to the powertrain control module
	Engine Intake Air Mass Flow Rate	Measures the flow rate of the gas entering the engine; computer calculates the amount of fuel required to maintain the correct fuel mixture
	Engine Crankcase Pressure 1	Obtains crankshaft position by pressure information, which will be used by the drivetrain control module to synchronize fuel
Position	Accelerator Pedal Position 1	Detected and transmitted to the ECU to take action on the opening and closing of the throttle. The accelerator is actuated directly by the driver via the accelerator pedal
	Engine Throttle Valve 1 Position 1	Determines the opening angle of the valve, and then determines the amount of air required by the engine and sends this information to the powertrain control module
	Fuel Level 1 & Fuel Level 2	Used to perform precise fuel level measurement in fuel tanks. It is mainly used to determine the fuel volume to be filled in the vehicle and remotely monitor the fuel tank
Temperature	Engine Intake 1 Air Temperature	The temperature of the gas entering the engine is measured and passed to the ECU to optimize combustion, which can optimize the fuel supply so that the air-fuel ratio enables efficient combustion
	Aftertreatment 1 Exhaust Temperature 1	The sensor detects the aftertreatment exhaust temperature and converts it into a voltage, which is then fed back to the ECU along with the voltage signal to control engine operating conditions and effectively reduce emissions
	Engine Coolant Temperature	Fed back to the ECU, which uses the data to adjust fuel injection and ignition timing; also turns on the electric cooling fan
Exhaust	Engine Exhaust Gas Recirculation 1 Mass Flow Rate	Computer calculates the amount of air required to maintain the correct fuel mixture
	Aftertreatment 1 Diesel Exhaust Fluid Concentration	Measures fuel concentration and quality and tells the system when to change oil to improve efficiency

Table A.1: Engine Management Sensors

Principles	Sensor Name	Significance
Velocity	Front Axle Speed	The average speed of the two front wheels represents the speed of the wheels and is transmitted to the anti-lock brake system, traction control system and electronic stability program control unit, which individually control the braking force of each wheel
	Wheel-Based Vehicle Speed	Speed of the vehicle as calculated from wheel or tailshaft speed, which is used by the ECU to modify engine functions (such as ignition timing, air-fuel ratio and transmission shift point) and to initiate diagnostic routines
	Transmission 1 Output Shaft Speed	Electronic transmission control module uses this information to determine the clutch slip of the torque converter
	Fuel Pump Primer Control	Parameter used to activate or deactivate a priming system on the fuel transfer system. The fuel priming system is a system that purges air in the fuel lines and may assist fuel delivery to a second pump at lower speeds
Temperature	Transmission Oil Temperature 1	Provides input to the transmission control module, which is used by the transmission control module to monitor the temperature of the transmission fluid

Table A.2: Transport Management Sensors

Principles	Sensor Name	Significance
Temperature	Propulsion Motor Oil Pump Control Temperature	Detects the temperature of oil pump which is driven by propulsion motor
	Propulsion Motor Coolant Pump Control Temperature	Detects the temperature of coolant pump which is driven by propulsion motor
	Power Electronics Coolant Pump Control Temperature	Detects the temperature of coolant pump which is driven by power electronics
Power	Propulsion Motor Oil Pump Power	Detects the power of propulsion motor used for driving the oil pump
	Propulsion Motor Coolant Pump Power	Detects the power of propulsion motor used for driving the oil pump

Table A.3: Power Management Sensors

Principles	Sensor Name	Significance
Current	DC/AC Accessory Inverter 1 AC Side RMS Current	Performs charging status and operation status calculation
Voltage	Battery Potential / Power Input 1	The battery monitoring system measures battery voltage and performs battery balancing to monitor and protect the battery

Table A.4: HEV Management Sensors

Appendix B

The operating steps through a web interface

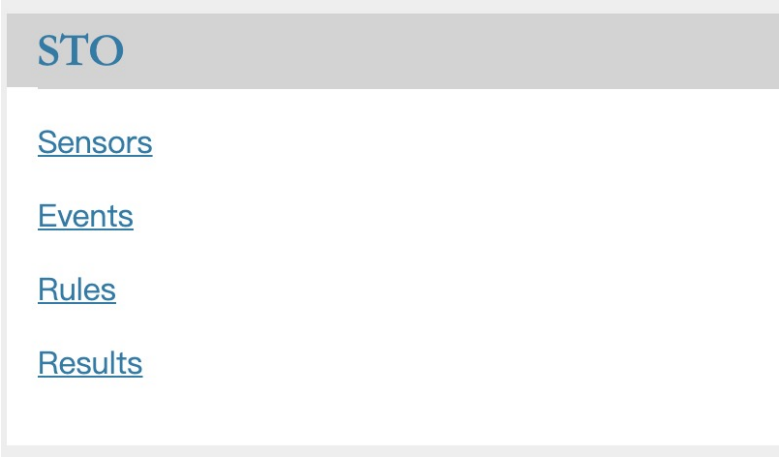

FIGURE B.1: The information list stored in the database

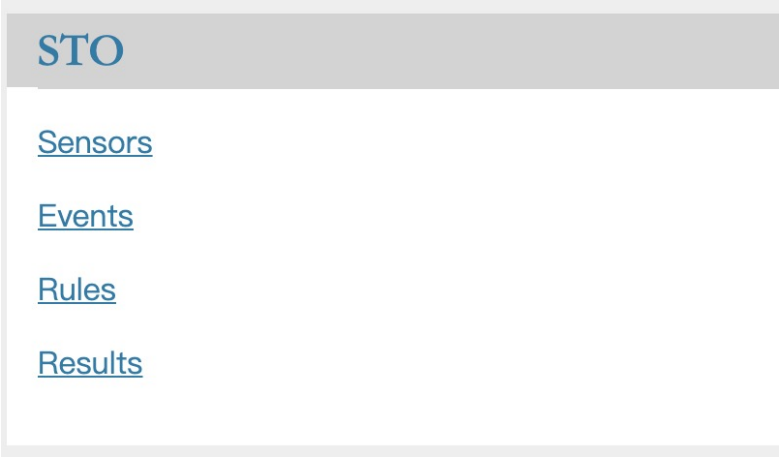

FIGURE B.2: The homepage for showing stored information in the database

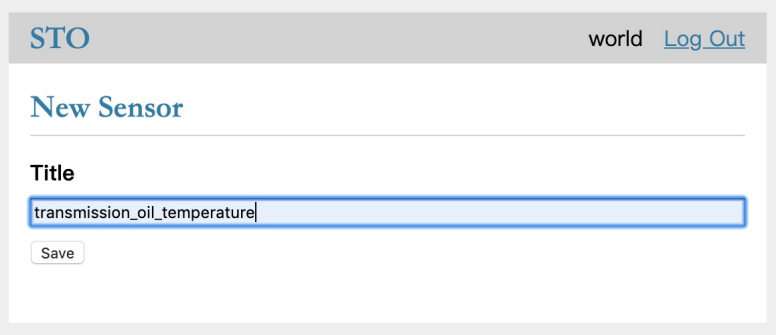

FIGURE B.3: The creation of a new sensor

FIGURE B.4: The creation of a new event

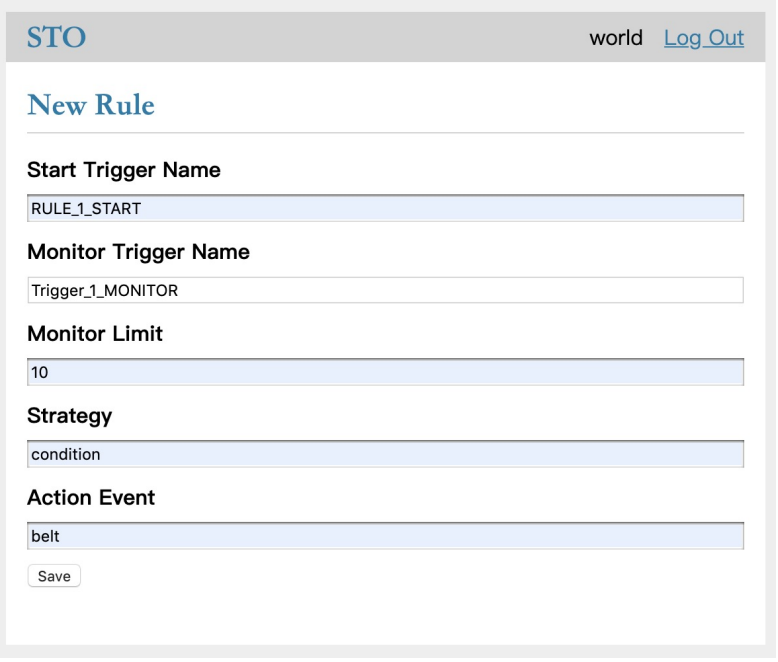

FIGURE B.5: The creation of a new rule

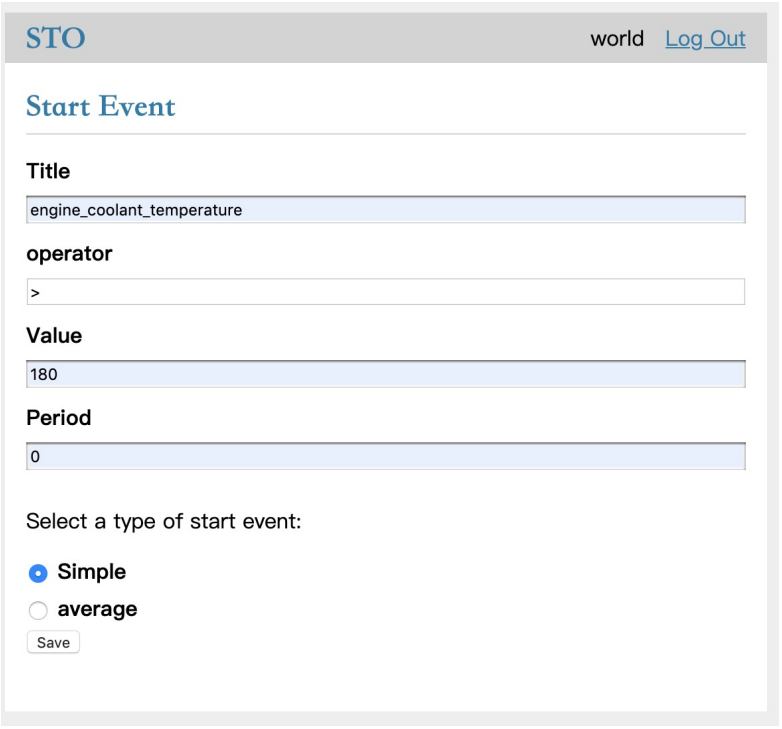

FIGURE B.6: The creation of a new start event of the new rule

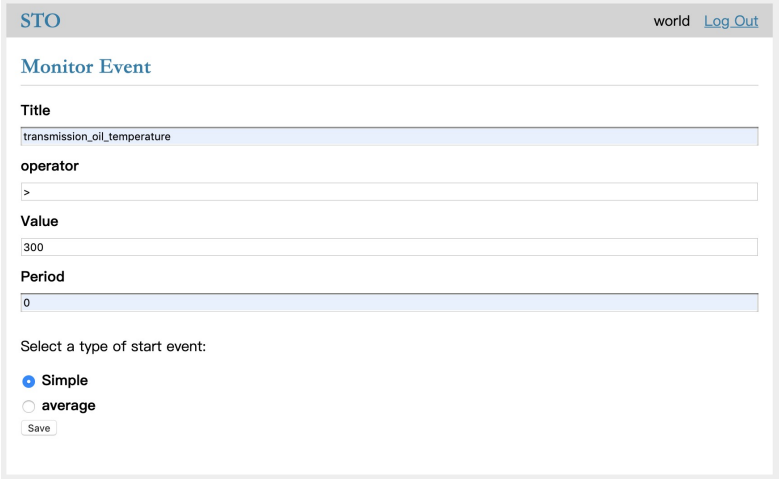

FIGURE B.7: The creation of a new monitor event of the new rule

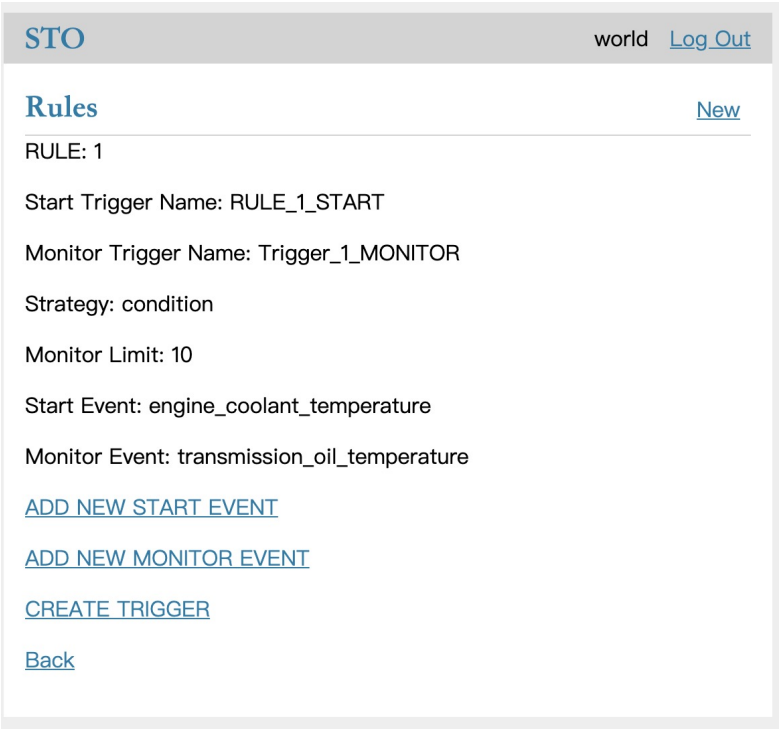

FIGURE B.8: The rule information

```
CREATE TRIGGER `RULE_1_START` AFTER INSERT ON `Sensor_Data` FOR EACH ROW BEGIN SELECT monitored
FROM Rule WHERE id = 1 INTO @monitored; IF(@monitored = 0)THEN IF(NEW.engine_coolant_temperature > 180)
THEN INSERT INTO Log_Event (datetime, ruleId, title) VALUES (CURRENT_TIMESTAMP, 1, 'monitor
started');SELECT `transmission_oil_temperature` FROM sensor_data WHERE id = NEW.id INTO @value; INSERT
INTO RULE_1_MONITOR(`transmission_oil_temperature`) VALUES (@value); END IF; END IF; END;CREATE TRIGGER
`Trigger_1_MONITOR` AFTER INSERT ON `RULE_1_MONITOR` FOR EACH ROW BEGIN IF(NEW.id <= 10) AND
(NEW.transmission_oil_temperature > 300)THEN INSERT INTO LOG_EVENT(datetime, ruleId, title) VALUES
(CURRENT_TIMESTAMP, 1, 'Stop Monitoring!'); INSERT INTO LOG_EVENT(datetime, ruleId, title) VALUES
(CURRENT_TIMESTAMP, 1, 'Rule triggered!'); INSERT INTO Action_Event (datetime, `rule_id`, belt) VALUES
(CURRENT_TIMESTAMP, 1, 1); END IF; END;
```

FIGURE B.9: The trigger information

Bibliography

[1] M. A. J. Jamali, B. Bahrami, A. Heidari, P. Allahverdizadeh and F. Norouzi. Towards the internet of things: Architectures, security, and applications. *IoT architecture*, pages 9–31, June 2019.

[2] M. Wu, T. Lu, F. Ling, H. Du and J. Sun. Research on the architecture of internet of things. *2010 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE)*, 2010.

[3] M. Rouse. What is internet of the things. *IoT Agenda*, 2019. https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT.

[4] K. L. Lueth. Why the internet of things is called internet of things: Definition, history, disambiguation. *IoT Analytics*, December 2014. URL https://iot-analytics.com/internet-of-things-definition/.

[5] R. Mehta, J. Sahni and K. Khanna. Internet of things: Vision, applications and challenges. *International Conference on Computational Intelligence and Data Science (ICCIDS 2018)*, 2018.

[6] N. Bessis and C. Dobre. *Big data and internet of things: A roadmap for smart environments*, volume 546. Springer, 2014.

[7] Fact sheet - hybrid buses: Costs and benefits. *Environmental and Energy Study Institute*, 2018. URL https://www.eesi.org/papers/view/fact-sheet-hybrid-buses-costs-and-benefits.

[8] J. Peng, H. He and R. Xiong. Rule based energy management strategy for a series–parallel plug-in hybrid electric bus optimized by dynamic programming. *Applied Energy*, 185:1633–1643, January 2017.

[9] Wikipedia. Hybrid vehicle. *Wikipedia*, 2019. URL https://en.wikipedia.org/wiki/Hybrid_vehicle.

[10] M. Voss. A comprehensible guide to controller area network. *Copperhill Media*, 2010.

[11] R. P. Walter and E. P. Walter. Data acquisition from hd vehicles using j1939 can bus. *Society of Automotive Engineers*, 2016.

[12] Y. Burakova, B. Hass, L. Millar and A. Weimerskirch. Truck hacking: An experimental analysis of the sae j1939 standard. *WOOT 2016*, August 2016.

[13] S. Selcuk. Predictive maintenance, its implementation and latest trends. *Journal of Engineering Manufacture*, 231:1670–1679, 2017.

[14] M. Johnston. Five steps to applying predictive maintenance. *Plant Engineering*, 69:43–45, 2015.

[15] H. M. Hashemian and W. C. Bean. State-of-the-art predictive maintenance techniques. *IEEE Transactions on Instrumentation and Measurement*, 60:3480–3492, October 2011.

[16] S. Ranger. What is the iot? everything you need to know about the internet of things right now. *Cybersecurity in an IoT and Mobile World*, August 2018.

[17] H. Opocenska and M. Hammer. Use of technical diagnostics in predictive maintenance. *2016 17th International Conference on Mechatronics - Mechatronika (ME)*, December 2016.

[18] P. Poór, D. Ženíšek and J. Basl. Historical overview of maintenance management strategies: Development from breakdown maintenance to predictive maintenance in accordance with four industrial revolutions. *Proceedings of the International Conference on Industrial Engineering and Operations Management*, July 2019.

[19] H. Yin, G. Zhang, H. Zhua, Y. Deng and F. He. An integrated model of statistical process control and maintenance based on the delayed monitoring. *Reliable Engineering & System Safety*, 133:323–333, January 2015.

[20] F. Francis and M. Mohan. Arima model based real time trend analysis for predictive maintenance. *2019 3rd International conference on Electronics, Communication and Aerospace Technology (ICECA)*, June 2019.

[21] P. Cesar, L. Gerum, A. Altay and M. B. Gursoy. Data-driven predictive maintenance scheduling policies for railways. *Transportation Research Part C: Emerging Technologies*, 107:137–154, July 2019.

[22] A. J. P. Ibáñeza, J. M. M. Bernala, M. J. C. Diegob and F. J. A. Sánchezy. Expert system for predicting buildings service life under iso 31000 standard. application in architectural heritage. *Journal of Cultural Heritage*, 18:137–154, March 2016.

[23] H. Luttenberg, C. Bartelheimer and D. Beverungen. Designing predictive maintenance for agricultural machines. *ECIS 2018 Proceedings at AIS Electronic Library (AISeL)*, November 2018.

[24] S. Kiran, P. V. R. VijayaramKumar, S. Vijayakumar, P. S. Varakhedi and V. Upendranath. Application of kalman filter to prognostic method for estimating the rul of a bridge rectifier. *Emerging Trends in Communication Control Signal Processing & Computing Applications (C2SPCA) 2013 International Conference*, pages 1–11, October 2013.

[25] C. S. Byington, M. Watson and D. Edwards. Data-driven neural network methodology to remaining life predictions for aircraft actuator components. *Aerospace Conference 2004*, 6:3581–3589, March 2004.

[26] J. Kang, X. Zhang, J. Zhao and D. Cao. Gearbox fault prognosis based on chmm and svm. *Quality Reliability Risk Maintenance and Safety Engineering (ICQR2MSE) 2012 International Conference*, pages 703–708, June 2012.

[27] P. Zschech. A taxonomy of recurring data analysis problems in maintenance analytics. *Twenty-Sixth European Conference on Information Systems (ECIS 2018)*, June 2018.

[28] Y. Peng, M. Dong and M. J. Zuo. Current status of machine prognostics in condition-based maintenance: a review. *The International Journal of Advanced Manufacturing Technology*, 50:297–313, 2010.

[29] J. Furch, T. Turo, Z. Krobot and J. Stastny. Using telemetry for maintenance of special military vehicles. *International Conference on Modelling and Simulation for Autonomous Systems*, pages 392–401, 2017.

[30] M. Svensson, S. Byttner and T. Rognvaldsson. Self-organizing maps for automatic fault detection in a vehicle cooling system. *2008 4th International IEEE Conference Intelligent Systems*, November 2008.

[31] H. M. Elattar, H. K. Elminir and A. M. Riad. Conception and implementation of a data-driven prognostics algorithm for safety–critical systems. *Soft Comput 23*, pages 3365–3382, 2019.

[32] K. Martin and M. Marzi. Diagnostics of a coolant system via neural networks. *Journal of Systems and Control Engineering*, 213:229–242, 1999.

[33] R. Golini and M. Kalchschmidt. Designing an expert system to support competitiveness through global sourcing. *International Journal of Production Research*, 53: 3836–3855, 2015.

[34] C.S. Krishnamoorthy and S. Rajeev. *Artificial Intelligence and Expert Systems for Engineers*. CRC Press, 1996.

[35] D. S. Maylawati1, W. Darmalaksana and M. A. Ramdhani1. Systematic design of expert system using unified modelling language. *The 2nd Annual Applied Science and Engineering Conference (AASEC 2017)*, 2018.

[36] M. Agarwal and S. Gael. Expert system and it's requirement engineering process. *IEEE International Conference on Recent Advances and Innovations in Engineering (ICRAIE-2014)*, 2014.

[37] K. Kozhakhmet L. Atymtayeva and G. Bortsova. *Soft Computing in Artificial Intelligence*. Springer, 2014.

[38] N. W. Paton. *Active rules in database systems*. Springer, 1999.

[39] A. Alasgarova and L. Muradkhanli. Expert system for decision-making problem in economics. *International Journal of Information Technologies and Knowledge*, 2, 2008.

[40] M. Bahrami and S. Kaviani. A new method for knowledge representation in expert system's (xmlkr). *Proceedings - 1st International Conference on Emerging Trends in Engineering and Technology*, pages 326–331, 2008.

[41] A. Paszek and R. Knosala. The method of the knowledge representation in an expert system for metal cutting engineering. *Journal of Materials Processing Tech*, pages 319–326, 1997.

[42] B. Sergei. Predictive maintenance framework for a vehicular iot gateway node using active database rules, 2018.

[43] S. Ceri and P. Fraternali. *Designing database applications with objects and rules*. Addison Wesley, 1997.

[44] Y. Qiao, K. Zhong, H. Wang and X. Li. Developing event-condition-action rules in real-time active database. *Symposium on Applied Computing: Proceedings of the 2007 ACM Symposium on Applied Computing*, pages 511–516, March 2007.

[45] J. Pavon. A rule meta-model for business rules. *The 16th IASTED International Conference on Applied Simulation and Modelling*, pages 511–516, August 2007.

[46] S. Viana, J. R. Almeida and J. Pavón. A rule repository for active database systems. *CLEI ELECTRONIC JOURNAL*, 10, December 2007.

[47] D. R. McCarthy and U. Dayal. The architecture of an active database management system. *SIGMOD '89 Proceedings of the 1989 ACM SIGMOD international conference on Management of data*, 18:215–224, June 1989.

[48] P. Killeen, B. Ding, I. Kiringa and T. Yeap. Iot-based predictive maintenance for fleet management. *Procedia Computer Science*, 151:607–613, 2019.

[49] P. Killen. Knowledge-based predictive maintenance for fleet management, 2020.

[50] E. Fumeo, L. Oneto and D. Anguita. Condition based maintenance in railway transportation systems based on big data streaming analysis. *Procedia Computer Science*, 53:437–446, 2015.

[51] H. Wang, O. L. Osen, G. Li, W. Li, H. Dai and W. Zeng. Big data and industrial internet of things for the maritime industry in northwestern norway. *TENCON 2015 - 2015 IEEE Region 10 Conference*, 2015.

[52] T. Lee and M. Tso. A universal sensor data platform modelled for real-time asset condition surveillance and big data analytics for railway systems: Developing a "smart railway" mastermind for the betterment of reliability, availability, maintainability and safety of railway systems and passenger service. *2016 IEEE SENSORS*, 2016.

[53] Copperhill Technologies. Sae j1939 ecu simulator board with usb port, 2019. https://copperhilltech.com/sae-j1939-ecu-simulator-board-with-usb-port/.

[54] Adafruit. Adafruit fona 800 shield - voice/data cellular gsm, 2019. https://www.adafruit.com/product/2468.

[55] ModMyPi. Ups pico - uninterruptible power supply & i2c control hatups pico, 2019. https://www.buyapi.ca/product/ups-pico-uninterruptible-power-supply-i2c-control-hat/.

[56] S. Ceri and P. Fraternali. *Designing database applications with objects and rules: The IDEA methodology*. Springer, 1997.

[57] G. Guerrinia and D. Montesi. Design and implementation of chimera active rule language. *Data & Knowledge Engineering*, 24:39–67, October 1997.

[58] T. Rögnvaldsson, S. Nowaczyk, S. Byttner, R. Prytz and M. Svensson. Self-monitoring for maintenance of vehicle fleets. *Data Mining and Knowledge Discovery*, 32:344–384, March 2018.