Received September 1, 2020, accepted September 15, 2020, date of publication September 22, 2020,
date of current version October 2, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3025933

A Model-Driven Alarms Framework (MAF) With Mobile Clients Support for Wide-Ranging Industrial Control Systems

HANNY TUFAIL[1], FAROOQUE AZAM[1], MUHAMMAD WASEEM ANWAR[1], MUHAMMAD NOUMAN ZAFAR[1], ABDUL WAHAB MUZAFFAR[2], AND WASI HAIDER BUTT[1]

[1]Department of Computer and Software Engineering, College of Electrical and Mechanical Engineering, National University of Sciences and Technology (NUST), Islamabad 44000, Pakistan

[2]Department of Information Technology, College of Computing and Informatics, Saudi Electronic University, Riyadh 13323, Saudi Arabia

Corresponding author: Muhammad Waseem Anwar (waseemanwar@ceme.nust.edu.pk)

ABSTRACT An efficient and robust *alarms management* is an extremely desirable feature to prevent critical failures in modern Industrial Control Systems. Generally, *alarm systems* are categorized into two major components; i.e., *Alarm Server* and *Alarm Client*. *Alarm server* is responsible to process alarms on real time values, while *alarm clients* display the generated alarms. With the advent of modern technologies, like, Internet of Thing (IoT), the need for mobile alarm clients has significantly been increased, as alarms need to be displayed on different gadgets instead of traditional big screens. Owing to significance of alarms in the industry, several state-of-the-art approaches have been introduced, suggesting various improvements. However, such approaches are only meant for *specific industry* and cannot be applied across-the-board. Furthermore, full support for mobile clients is not available. On the other hand, certain industrial software products like, SIMATIC WinCC, Genesis64 provide complete *alarm solution* including *mobile client features*, as well. However, such solutions are proprietary with higher costs, i.e., *pay per tag* model. Considering such limitations, this article proposes a novel **M**odel-driven **A**larms **F**ramework (**MAF**) with mobile clients (*Android* and *iOS*) support. Particularly, MAF comprises an **A**larm **P**rofile for **I**ndustrial **C**ontrol **S**ystems (**APICS**) for the modeling of *server*, *mobile clients* and *configuration requirements* of alarms with remarkable simplicity. Furthermore, a complete open source Alarms Profile Transformation Engine (**APTE**) has been implemented to automatically generate *alarm server* (Kotlin), native Android (Kotlin) and iOS (Swift) *alarm clients* from APICS-compliant models. The feasibility of proposed framework is demonstrated through two case studies (*flour mill* and *home automation*). The results prove that MAF not only provides *complete alarm server solution* to generate bulk of alarms simultaneously, but also supports *android and iOS alarm clients* for efficient visualization. Besides that, MAF is *easy to use*, *cost-effective* and can be *applied in wide-ranging industrial applications.*

INDEX TERMS Alarm systems, control systems, industry automation, industrial alarms, model driven engineering (MDE), process control alarms, Unified Modeling Language (UML).

I. INTRODUCTION

Industrial Control Systems (ICS), *a.k.a.* Process Control Systems (PCS) are employed along the production line to monitor and control the processes [1]. The ICS is relatively a general term used for hardware and software integration with network connectivity to support critical industrial processes. Typically, an ICS is the combination of software and hardware to handle physical processes in the industry even from a remote location. They can acquire, process and log real-time data for particular industrial process. Furthermore, they allow direct and instant monitoring of industrial processes through Human Machine Interface (HMI) [12]. The basic architecture of industrial control systems is shown in **Figure 1.**

The associate editor coordinating the review of this manuscript and approving it for publication was Giambattista Gruosso.

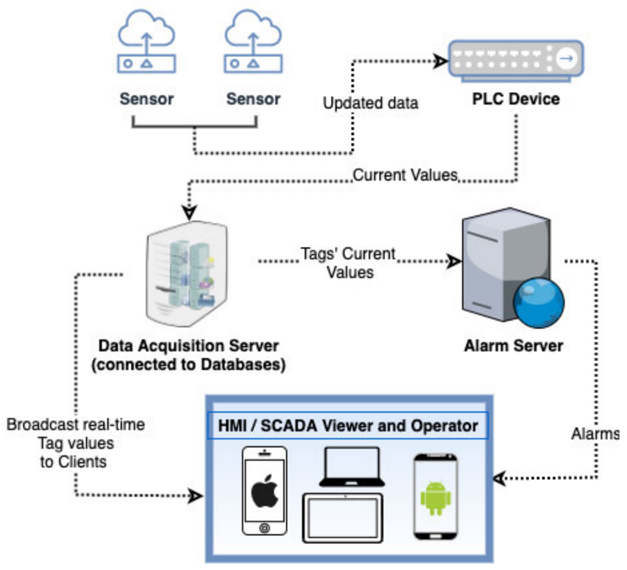

Sensor
Sensor
Updated data
PLC Device
Current Values
Tags' Current Values
Data Acquisition Server (connected to Databases)
Alarm Server
Broadcast real-time Tag values to Clients
HMI / SCADA Viewer and Operator
Alarms

FIGURE 1. **Basic architecture of industrial control system.**

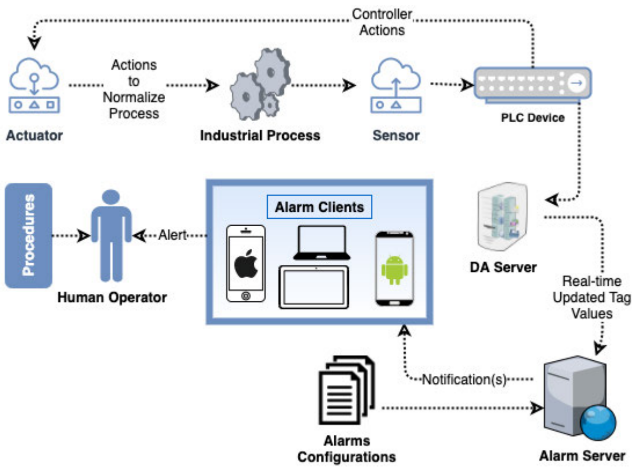


FIGURE 2. **Architecture of alarm system in industrial control system.**

ICS usually employ Programmable Logic Controllers (PLCs) to fetch real-time values from sensors which are attached to the industry machines. Data Acquisition (DA) server is a central component of ICS which is responsible to fetch real-time values from multiple PLCs. Subsequently, the fetched values are further processed to display real-time data to operators via HMIs for efficient monitoring and control as shown in **Figure 1.** Similarly, other components of ICS like alarm server etc., also take real-time values from DA server as needed.

The importance of ICS is significantly increased during last decade due to several technological advancements like introduction of Internet of Thing (IoT) [12]. Nowadays, the support for advanced data acquisition and HMI features is mandatory in modern ICS. Moreover, the safety critical nature of industrial processes requires robust and reliable notification mechanism in case of malfunctioning. Consequently, *alarm* is highly desirable component in modern ICS. Particularly, alarms are intrusive alert to the human operator. An alarm notifies the operator about the behavior of controlled process. Human operators monitor and control industrial processes through HMI screens. However, a large amount of information, displayed on their system, could potentially lead an operator to miss important information in HMI about critical process state. It could eventually lead to serious damages. To manage such situations, alarms component of ICS plays a vital role.

A typical architecture of alarm system particularly in ICS is shown in **Figure 2**. Generally, alarm component is comprised of three sub-components i.e. *Alarm Configurations, Alarm Server* and *Alarm Client*. Particularly, the basic configurations of alarms like *tags* and *type of alarm* etc. are performed in alarm configuration sub-component. Alarm server is a major sub-component which is responsible for evaluation and generation of alarms as per given configurations. Particularly, it takes configuration settings of alarms from alarm configuration sub-component. Subsequently, it fetches real-time values of configured tags from DA server. Afterwards, it evaluates real-time values with respect to given configurations and generate alarm accordingly. Alarm client sub-component is responsible to display / notify the generated alarms from alarm server to human operators and higher management through HMI screens. The alarm clients can be a desktop or mobile screens as shown in **Figure 2.**

In ICS, alarm server and client sub-components are of high importance. Particularly, there are four major types of alarms i.e. *Limit, Deviation, Rate-of-Change (RoC)* and *Digital Alarms* [37]. In this regard, alarm server should be able to support all four types of alarms. Furthermore, it should popup alarms in timely manner based on real-time values. On the other hand, alarm client should be able to display / notify the generated alarms to human operators with minimum delay. Furthermore, it should support modern and sophisticated mobile HMI screens (e.g. Android, iOS etc.) along with desktop screens for the notification of alarms.

As alarm component, being intrusive alert to the human operator, serves as an integral part of ICS. It has been researched intensely over the last decade. For instance, there exist state-of-the-art approaches deployed at industrial level to monitor: 1) air quality in a room [5], 2) power usage by motor conveyers in a production unit [6], 3) communal water supply system to reduce drinking water losses [7], and 4) day-to-day railway operations [13].

However, these approaches [5]–[7] [13] are limited to a particular domain and are difficult to apply in other industries. In addition to this, several machine learning techniques are also employed for the generation of alarms based on the historical trends [20], [21], [23]. However, the reliability of such approaches is questionable, therefore, their actual application in industries is difficult. Moreover, there are studies [8], [30], [34]–[36] that attempt to enhance the capabilities

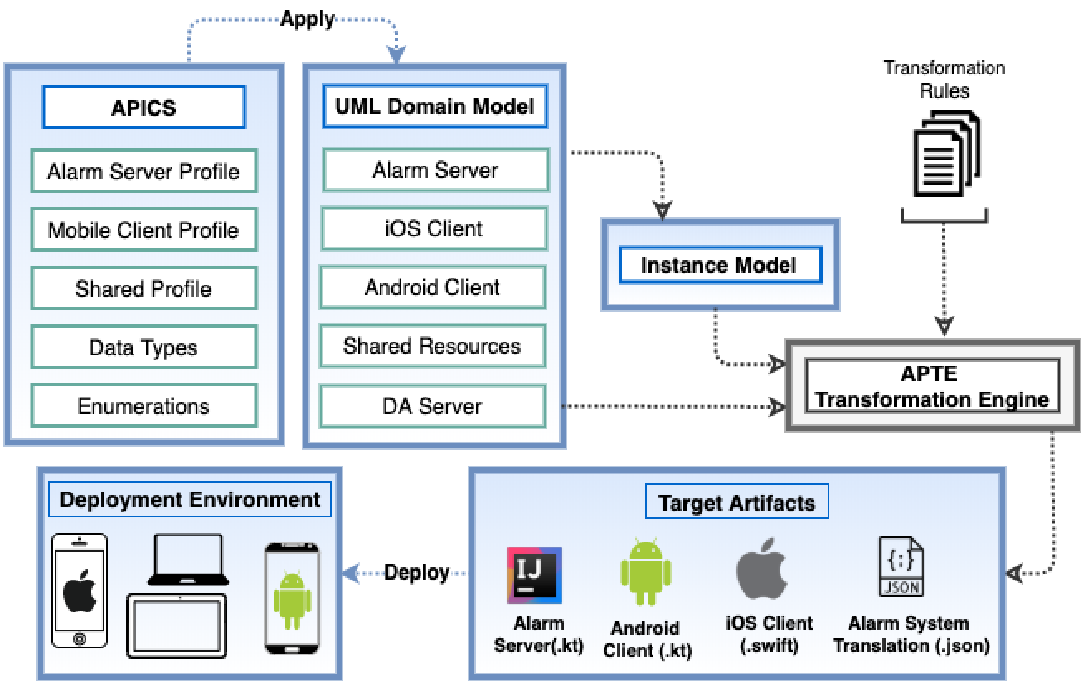


FIGURE 3. Overview of Model-driven Alarm Framework (MAF).

of alarm server. However, such studies are only meant for limited area and their application is not wide-ranging. Furthermore, there are studies [15]–[19] particularly dealing with the ergonomics of notification for alarm client; to enhance the capabilities of mobile HMI display for alarm notifications. However, such studies are unable to simultaneously support both iOS and Android platforms. On the other hand, there exist complete and reliable alarms solutions in industry automation suites [37] (e.g. Genesis64 etc.). However, such industry-oriented solutions are propriety and lead to higher operational costs (e.g. pay per tag). Furthermore, the underlying development techniques for such industrial solutions are not publicly available.

Owning to aforementioned limitations of state-of-the-art approaches, there is a need of simple, cost effective and open source framework to provide a complete alarms solution for wide variety of industrial control systems. Particularly, it should be able to support alarm server where several alarms can be processed simultaneously in real time, to support wide-ranging industrial processes. Moreover, it should also support both android and iOS mobile alarm clients for the notification of alarms to human operators. Furthermore, it should be available publicly without involving any licensing and operational cost. To achieve these objectives, this article presents a Model-driven Alarms Framework (**MAF**) for industrial control systems. The overview of MAF is shown in **Figure 3**. The contributions of this article are as follows: -

- Firstly, an Alarm Profile for Industrial Control Systems (**APICS**) has been developed by extending the standard Unified Modeling Language (UML) concepts. APICS allows the modeling of server, mobile clients and configuration requirements of alarms with remarkable simplicity.
- Secondly, a fully open source Alarms Profile Transformation Engine (**APTE**) is implemented to automatically generate: 1) Kotlin code for alarm server, 2) native Android and iOS code for alarm clients in Kotlin and Swift respectively, and 3) alarms configurations in JSON from APICS-compliant models. This allows to instantly deploy alarm server and clients (i.e. Android and iOS) for the monitoring of industrial processes.
- Finally, the applicability of MAF is demonstrated through *Flour Mill and Home Automation* case studies. The experimental results confirmed that MAF is easy to use and cost-effective solution which provides simultaneous support for both android and iOS clients with historian alarms feature. Furthermore, it is also established that alarms server can process several alarms with optimum response time. Consequently, it can be concluded that MAF is a highly efficient and widely applicable solution to meet the control and monitoring demands of modern industrial processes.

The remainder of this article is structured as: The state-of-the-art and industrial solutions are discussed in Section II. This allows to define a realistic research gap for the contributions of this article. In Section III, the details of Alarm Profile for Industrial Control Systems (APICS) are given. In Section IV, the implementation details of Alarms Profile Transformation Engine (APTE) are provided. In Section V, the viability of MAF is established through *Flour Mill* and *Home Automation* case studies. The important aspects of this research are discussed in Section VI. Finally, article is concluded in Section VII.

II. LITERATURE REVIEW

For an industrial plant to run smoothly, efficiently, with safety while keeping the complete control over the operations, alarm systems play a crucial role [4]. In US alone, a medium sized petrochemical company with separate five to six sites can easily accumulate US$50,000,000 to $100,000,000 yearly losses due to plant setbacks and other production mishaps [3]. The study [4] compiled a list of 12 major accidental occurrences that resulted in fatalities and loss of millions of dollars. The reasons were later diagnosed as related to alarm systems' impairments. Due to importance of alarms systems in industries, researchers frequently target this area. In this section, the literature review regrading current alarms solutions for industrial control systems is summarized. Particularly, the existing state-of-the-art approaches (Section A) and solutions provided in the industry (Section B) are analyzed and highlighted in this section. We categorized the selected researches as per their taxonomies.

A. STATE OF THE ART APPROACHES

There are numerous applications of alarm systems in different industries as witnessed in the literature. For example, in [11], authors have developed an android-based app, fall guard, for elderly people to get instant intervention from the caregivers. This application generates alarms and notifies the concerned party in real-time about the fall of elderly person. In another study [9], authors debated on the importance of alarms in vital signs monitors and elevated them to be life-threatening, if not presented timely to care-givers. Subsequently, they proposed an alarm system for vital sign monitors using Qt creator environment. Similarly, in [10], authors have designed an alarm management system for patient monitoring system.

In [13], authors have proposed an effective method to fetch alarm data, using TCP/IP, from railway control room. The method comprises power supervisory system to ensure smooth and stable running of day-to-day railway operations. They validated their proposed methodology on a Lanzhou-Chongqing railway station.

In [34], authors have deployed penalty concept and Markov Chain approach to develop a generalized delay timer for designing flexible alarm to monitor industrial processes. They derived *False Alarm Rate* (FAR*), Missed Alarm Rate* (MAR) and *Average Alarm Delay* (AAD) for an alarm system using the above-mentioned approaches and then introduced another index i.e. Mean Time to alarm for accurate analysis. The proposal is validated through two case studies. In another study [35], authors have utilized Dempster-Shafer theory of evidence and claim to reduce the uncertainties of the monitored industrial process. While optimizing an alarm system, they suggest that AAD should be considered as well in addition to FAR and MAR. Furthermore, authors discussed the tradeoffs among these indices and validated their approach using numerical experiments case studies. In [36], authors have proposed a new generalized delay-timer technique to avoid false and nuisance alarms. Authors computed the common three performance indicators i.e. MAR, FAR and AAD to generalize their proposed methodology and validated it using two case studies.

One of the major components for an alarm's integrity is its meaningful presentation to the human operator [2]. There have been some studies carried out to emphasize the importance of designing an alarm system ergonomically as per the industrial standards. For example, in [15] and [16], the authors debate about designing an alarm system according to EEMUA standards and its configuration as per ISA norms. They analyzed the data from 5 different process systems. Similarly, in [17], authors studied the investigation reports of the accidents, since 1970, occurred in industry. These reports declared poorly designed alarm systems or alarm management culprit to some degree. The causal factors registered related to alarm systems are alarm floods, non-responsive alarms, lack of training of human operator, poor representation of alarms data and their prioritization. Therefore, authors developed a checklist to gauge ergonomic design quality of any alarm management system with ease, consistency and reliability. In another study [18], authors considered alarm systems in fifteen different process control systems. They analyzed them for their ergonomic design against a checklist of 148 items in 11 design areas like alarm presentation, alarm prioritization, documentation, training etc. Aforementioned studies [15]–[18] revealed that certain improvements are required in alarms system to meet the growing demands and standard of modern ICS.

1) PREDICTION OF ALARMS USING MACHINE LEARNING TECHNIQUES

Due to importance of alarm systems, several machine learning approaches are also targeted in literature to predict the alarms based on their historical trends. For example, in [20], authors used a machine learning approach and proposed a probabilistic model derived from n-gram model to predict the occurrence of alarms. This approach uses stored historical alarms raw data and transforms it to alarm sequence in a specific format and then trains their model with that sequence for prediction. However, the given model is strictly dependent on the strong correlations among alarms for accuracy. Furthermore, it doesn't cater the long-distance correlations. This limitation restricts the deployment of this technique over a large scale in multidimensional industrial domains. It is a complex way of prediction and no automated tool is developed. Furthermore, the proposed approach is not fault tolerant. In another study [21], the authors primarily studied the correlation variables among the alarms that effect the occurrence of alarm or cause abnormalities. They took the alarms logs of past three years from three identical Natural Gas Processing Plants and applied correlation methods i.e. Cluster analysis, Principal Component analysis and Correlation analysis to reduce number of alarms generation. They claim to reduce alarms occurrences by 80% by implementing their alarm management protocol.

In [22], authors studied the correlation variables in both process and alarm data and proposed a hierarchical cause-and-effect based alarm model. Using correlation delay analysis technique and considering alarm data, they devised a similarity coefficient. For process data, authors recommend Granger causality approach to help operator to efficiently understand the rationale for an alarm origin. In another study [23], authors have proposed an approach to inform human operators, at early stage, about the aberrant behavior of a process using machine learning approaches. i.e. training two neural networks on SCADA alarms log from gear box component of a Wind Turbine in Sweden.

In [24], authors have deployed IoT based machine learning approach at Slitting Machine to improve the overall manufacturing process. They fetched data from sensors, analyzed it, deployed clustering and supervised learning methods. Subsequently, author deployed Autoregressive Integrated Moving Average to predict aberrance and defects in quality. Similarly, in [25], authors proposed an IoT based alarm system using machine learning algorithms to detect threats and notify the operators in real time. These threats are based on environment variables collected from SCADA components. In another study [26], authors deployed 5 supervised machine learning techniques to classify data sets retrieved from continuous monitoring of ICS using MATLAB statistics and machine learning Toolboxes. In another study [31], authors have provided an approach to improve the performance of an existing alarm system using techniques like delay-timer and majority voting. They deployed delay-time to mitigate chattering alarms. And majority voting technique to handle duplicate alarms problem. They took 10 days of alarms logs data from a power plant and applied these techniques iteratively and claim to have reduced the alarms up to 93%.

Aforementioned approaches [20]–[26] certainly automate the different aspects of alarms systems without human intervention. However, such approaches are only meant for academia. Their application in real industrial processes is not feasible due the safety critical nature of such processes.

2) CURRENT CHALLENGES IN ALARM SYSTEMS

Alarms notifications can cause confusions by running abnormally or flooding the operators screen with false alarms. This can also result in hiding a critical alarm behind the flooded false alarms [27]. There have been some studies that proposed solutions for such problems. For example, in [27], authors addressed the problem of detecting the alarm floods. Typically, the number of alarm occurrences and alarm variables in the alarm state are the two criteria used to detect alarm floods. Authors introduced another variable in the algorithm i.e. number of alarm variables newly appeared in alarm state. The proposed algorithm not only detects occurring alarm floods but its presence in the historical data as well. In another study [28], authors proposed a method to rank and reorder the display of alarms during alarm floods based on the criticality index.

In [29], authors have proposed a resolution to the problem of alarm redundancy in alarm system, when process or device runs abnormally, of a petrochemical process. They introduced adaptive delay timer and adaptive deadband techniques to mitigate the problem. Authors claim that for uniform frequent alarms, adaptive delay timer is effective to reduce the number of alarms generated. In case of uneven alarms frequency, adaptive deadband technique works. In [49], another nuisance alarms reduction method called deadband is discussed and improved. The authors first discussed the suitability of the usage of deadband as a nuisance reduction technique and recommended 45 degrees deadband index. Furthermore, authors examined that both deadband and delay-timer are suitable for nuisance reduction when deadband index's statistical value is 45 degrees. Finally, authors proposed a design method to obtain optimal deadband width for general types of signals, and not just for the independent and identically distributed ones.

Considering above-mentioned literature, it is analyzed that existing studies only target some partial problem in alarm systems. Therefore, application of such approaches in real industries is infeasible. However, before concluding the research gap, it is necessary to evaluate the existing industry automation suites in the context of alarm. This really helps to identify genuine and realistic research gap. Therefore, we investigated the industry suites in the context of alarms in subsequent section.

B. PROCESS CONTROL ALARMS IN INDUSTRY AUTOMATION SOFTWARE (IAS)

In this section, we analyze leading Industry Automation Software (IAS) in the context of alarms. Particularly, we selected Genesis64, SIMATIC WinCC, LabView Wonderware and Open Automation Software (OAS) from ICONICS, Siemens, National Instruments, AVEVA and OAS vendors respectively. It is important to note that we performed the detailed comparison of aforementioned IAS in [37]. Here, we are only summarizing the key findings as given in **Table 1.**

It can be observed from **Table 1** that most of the IAS are proprietary with pay per tag policy. This leads to higher development and maintenance costs. Another issue with given IAS is that underlying development methodology is not publicly available, therefore, the customization of such tools for particular alarm conditions and environments is out of the question. In other words, extension and interoperability of existing IAS is biggest issue while managing modern industry automation demands.

C. RESEARCH GAPS AND PROPOSED SOLUTION

From the literature review, it has been analyzed that researchers frequently target industrial alarms to enhance particular functionality as per requirements. However, most of the studies are only beneficial for the sake of academic research and their application in real and wide-ranging industries is infeasible. On the other hand, there are sophisticated

TABLE 1. Comparison of leading industry automation software in the context of alarms.

Features	OAS	LabVIEW	Genesis64	SIMATIC WinCC	Wonderware
Alarms types	i. Limit ii. Deviation iii. Digital iv. ROC	i. Limit ii. Deviation iii. Digital iv. ROC	i. Limit ii. Deviation iii. Digital iv. ROC v. Rate Limit vi. Trigger Limit	i. Limit ii. Deviation iii. Digital iv. ROC	i. Limit ii. Deviation iii. Digital iv. ROC
OPC Compatible	YES	YES	YES	YES	YES
Open Source	NO	NO	NO	NO	NO
Alarm Viewer Client	YES	YES	AlarmWorx64 Viewer	YES (TIA Portal)	Wonderware Alarm Adviser
Mobile Client Support	YES	YES	YES	YES	YES
Supported Scripting Languages	i. VB ii. C#	i. Python ii. C# iii. G	i. SupportWorX ii. VB	I. VB II. Mendix	Event Driven Scripting using QuickScripts
HMI Available	YES	LVDSC HMI	GraphWorX64 HMI	YES (TIA Portal)	InTouch
Historical Alarms Support	YES	YES	YES	YES	YES
Licensing	YES with 30 days Trial period.	YES, get a quote	YES with 30 days Trial period.	YES with 21 days Trial period.	YES, but details not mentioned.

IAS that support advanced alarms features. However, such software solutions are costly.

Moreover, the internal development methodologies are not publicly available. Therefore, extension of such solutions for particular alarms requirements is not possible. Based on the aforementioned literature review and investigation of IAS, following research gaps are identified: -

- Existing studies are only targeting some particular alarm problem. Therefore, the application of such studies in broader industries is infeasible due to limited scope.
- Current advancements in mobile technologies require the deployment of alarm clients in multiple gadgets. For this, modern alarm systems should support both iOS and Android platforms along with their alarm server. However, current studies do not support both iOS and Android alarm clients. Although few IAS do provide support but are very costly i.e. Pay per tag policy.
- Existing IAS are costly whereas small and medium sized industries require cost-effective alarms systems.
- The development details of current IAS are not publicly available. Therefore, researchers and industry practitioners cannot customize such solutions for future requirements.
- Interoperability issues – Alarm systems are designed to support only prescribed server and databases.

To address aforementioned gaps, this article presents an open-source Model-driven Alarms Framework (**MAF**) that provides a complete alarms solution for wide-ranging ICS. It is not only compliant with OPC-based data acquisition server but also supports mobile clients (i.e. Android and iOS). Further details about the framework are discussed in the subsequent section.

III. MODEL-DRIVEN ALARMS FRAMEWORK (MAF)

MAF is developed by utilizing the concepts of Model Driven Architecture (MDA) as it provides a higher level of abstraction with automated end-to-end implementation [48]. MDA simplifies the development process and reduces the complexity in terms of time and cost [14]. The two key concepts of MDA are; modeling and model transformation. Modeling allows the representation of particular requirements of a given system at higher abstraction level. Subsequently, transformation process is executed to automatically generate desired low-level implementations from models. In this context, MAF is also based on modeling and model transformation activities. Particularly, modeling in MAF is accomplished through the Alarm Profile for Industrial Control Systems (**APICS**). APICS is developed by extending the standard Unified Modeling Language (UML) concepts. Furthermore, a fully open source Alarms Profile Transformation Engine (**APTE**) is employed to automatically generate target artifacts i.e. Kotlin code for alarm server, both Android and iOS clients code, in Kotlin and Swift respectively, and alarm configurations in JSON, from APICS-compliant models. This allows to instantly deploy alarm server and clients (i.e. Android and iOS) for the monitoring of industrial process.

It can be seen from **Figure 4** that APICS is developed by considering alarms server, mobile clients, PLC devices and Modbus communication protocols requirements.

Particularly, alarm server, and mobile client's concepts allow the modeling of respective requirements. On the other hand, PLC devices concepts are also required for tags address space, so that, the real time values of required tags can be fetched from Data Acquisition (DA) server.

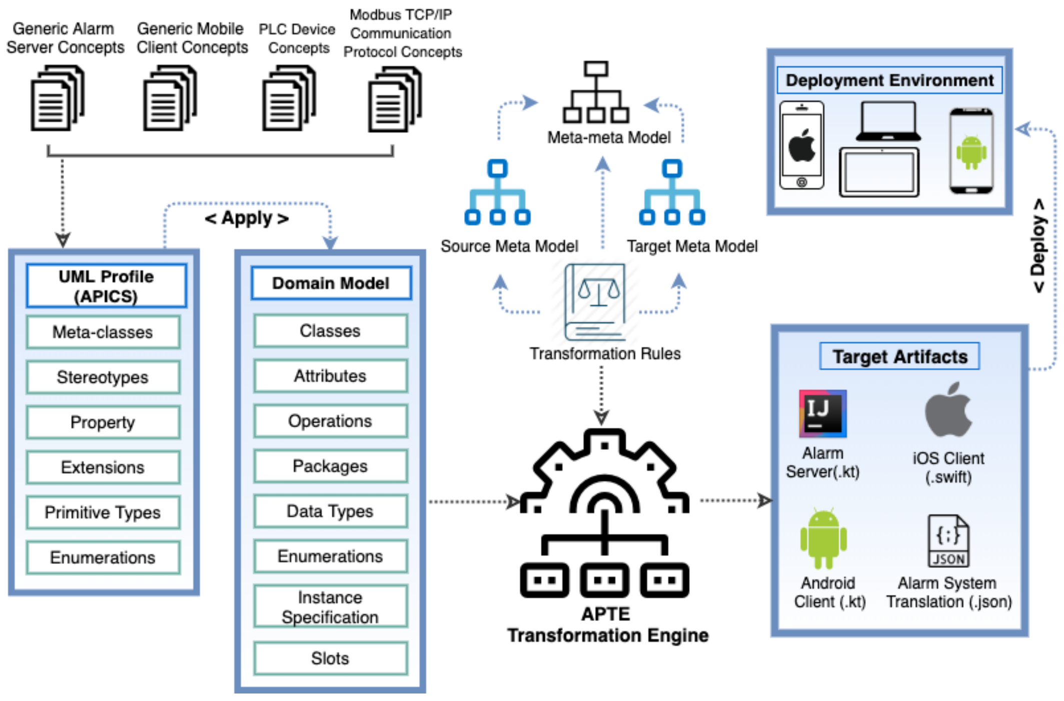


FIGURE 4. **Model-driven Alarm Framework (MAF) components.**

Finally, communication protocol concepts are necessary to communicate with DA server.

APICS enables the development of domain models. To automatically generate target artifacts, Alarms Profile Transformation Engine (APTE) is developed. Particularly, APTE takes APICS models and automatically generates alarm server, clients (i.e. Android and iOS) and configurations low level code. This allows instant deployment of alarm server and mobile clients as shown in **Figure 4.**

A. MAF TOOLS AND TECHNIQUES ARCHITECTURE

The architecture of tools and techniques used in the development of MAF can be seen in **Figure 5.** It comprises of four layers i.e. Metamodeling environment, Modeling editor, Model to Text Transformation (M2T) Tool and Simulation Tools. For metamodeling, Eclipse Oxygen– an open-source tool– is used [40]. It serves as a dashboard application with a vast variety of plugins for different intended purposes. For modeling, Papyrus– an Eclipse plugin for modeling – is used to develop domain model as well as UML Profile [41].

For transformation engine (APTE), model-to-text approach is employed through Acceleo plugin in Eclipse [42]. The fourth layer contains a list of tools required to test and deploy the automatically generated alarm server and mobile clients' code. For alarm server's code, IntelliJ IDEA CE is used for testing and deployment purposes [43]. Similarly, for android alarm client, Android Studio and Android Emulator are used [44], [45]. For iOS alarm clients, Xcode IDE and iOS Simulator are used [32]. For real time data acquisition, FATEK PLCs and open source Modbus DA server are used [38].

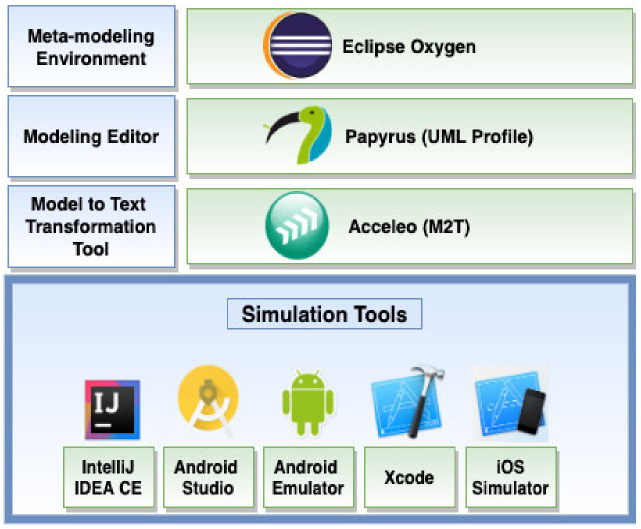


FIGURE 5. **Tools and techniques utilized in MAF.**

B. ALARM PROFILE FOR INDUSTRIAL CONTROL SYSTEMS (APICS)

To accomplish the modeling in MAF, APICS is proposed using UML concepts. UML is a general-purpose modeling language. UML can be extended through Profile mechanism to make UML a domain specific modeling language (DSML) [46]. UML profile is like a package in UML and uses same notation as well with just the addition of "Profile" guillemet that precedes the name of a profile. UML profile mainly consists of stereotypes, tagged values, constraints and extensions from meta classes. UML profiling concept is frequently utilized in different domains to accomplish particular modeling objective [47]. Therefore, the concepts of UML profiling are utilized for the development of APICS.

In MAF, an UML Meta-model extension concept termed as "Profile", is used to specialize the general constructs of a metamodel (i.e. UML) and refine them according to alarms domain (i.e. APICS with server and mobile clients' support). The APICS is modelled using Papyrus – a Java-based Eclipse plugin for modeling [41]. The **Figure 6** provides the complete overview of APICS Profile. To increase the understandability of APICS Profile, it is categorized into three sub-profiles as shown in **Figure 6.** This categorization is followed by the separation of client-server architecture components.

- Sub profile for Alarm Server
- Sub profile for Mobile Clients
- Sub profile for Shared concepts between Server and Mobile Client

1) SUB PROFILE FOR ALARM SERVER

The first sub profile, named as "AlarmServerProfile" in **Figure 6a**, represents alarm server concepts. It allows to model server related concepts in MAF. The concepts of alarm server sub-profile are later transformed to cross-platform desktop application in Kotlin. Basically, alarm server acts as a client for Data Acquisition (DA) to fetch real-time data from a particular PLC device located on a pre-defined IP address using TCP/IP communication protocol. Concurrently, it processes the fetched information from DA Server, as per instructions defined during Alarm configurations and broadcasts the real-time values along with messages to Mobile Clients (i.e. Android and iOS) in case of alarm. It contains the following elements i.e. Data Acquisition Server information, PLC Tag, Device, Destination and Input. In this context, following stereotypes are proposed in **Figure 6a.**

≪DA_Server_Info≫

This stereotype is extended from UML meta-class named Class. It is applied on a class in the domain model to document the location of Data Acquisition Server over the network using *DAServerIPAddress* and its physical endpoint to communicate using *DAServerPor*t. The Data Acquisition Server is connected with the industrial devices via network.

≪PLC_Tag≫

A tag is basically a link or connection string between a sensor and its associated address in the PLC. For each instance of an alarm, set to monitor a device in industry, an associated tag (i.e. *PLC_Tag*) is also created. The number of tags composed by a particular alarm, minimum one and maximum two, is derived from the type of alarm. An alarm can be of four types i.e. *Digital Alarm, RoC Alarm, Limit Alarm and Deviation Alarm*, and is expressed using an enumeration *AlarmType* **in Figure 6e**. Only alarm type – Deviation Alarm – can have two associated tags with it to compare the acceptable difference between two devices monitoring same thing e.g. Oxygen level in a room. Each tag is uniquely identified using *tagId* and the attribute *tagName* is used to keep track of an owning alarm. Whereas the *tagValue* attribute stores the data fetched at runtime from a device and serves as an input for performing logical operations according to the current readings.

≪Device≫

This stereotype is extended from UML meta-class named Class, Property and Slot. It represents the physical existence of a PLC device with a unique identification as deviceID and its relation to the process control equipment can be documented as deviceName. Since the communication of the PLC devices is via TCP network protocol, therefore its location over the network is denoted by ipAddress and the physical end point to communicate with the device is expressed as port.

≪Destination≫

This stereotype is extended from UML meta-class named Class, Property and Slot. It represents the memory location of the targeted device in the system, therefore denoted as location. These memory locations are also called registers and are used to read and write data on device memory. In case of Modbus protocol, the location attribute can be of four types i.e. *Coils, Holding_Registers, Input_Registers* and *Inputs*. It represents the memory type within a device and is expressed using an enumeration *LocationType* as shown in **Figure 6e.**

≪Input≫

This stereotype is extended from UML meta-class named Class, Property and Slot. It has two important attributes and collectively specify the physical address of the device. The *startingAddress* attribute represents starting memory address location and *numOfInput* is used to express the consecutive locations of address / register. The three stereotypes i.e. *Device, Destination* and *Input* combine to make a *tagAddress*, which is used to fetch required real times values from devices for alarm server as shown in **Figure 6a.**

2) SUB PROFILE FOR MOBILE CLIENTS

The second sub profile, named as "MobileClientProfile" in **Figure 6b**, is used to represent the requirements of mobile alarm clients. It comprises of client-side concepts that are required to present the alarms on mobile devices fetched from Server. It contains the following elements i.e. Alarm Server information, Alarm Severity color coding, Display factors, Alarm acknowledgement comments.

≪Alarm_Server_Info≫

This stereotype is extended from UML meta-class named Class. The alarm server serves as a client for DA Server to fetch real-time data from intended industrial devices. On the other hand, Alarm server acts as a "Server" for mobile clients. In this context, *Alarm_Server_IP_Address* provides location of alarm server to mobile clients and *Alarm_Server_Port* poses as a physical endpoint for communication.

≪Alarm_Severity_Color_Code≫

This stereotype is extended from UML meta-class named Class. This stereotype set the display color of a particular alarm in mobile clients as per severity. In alarm clients,

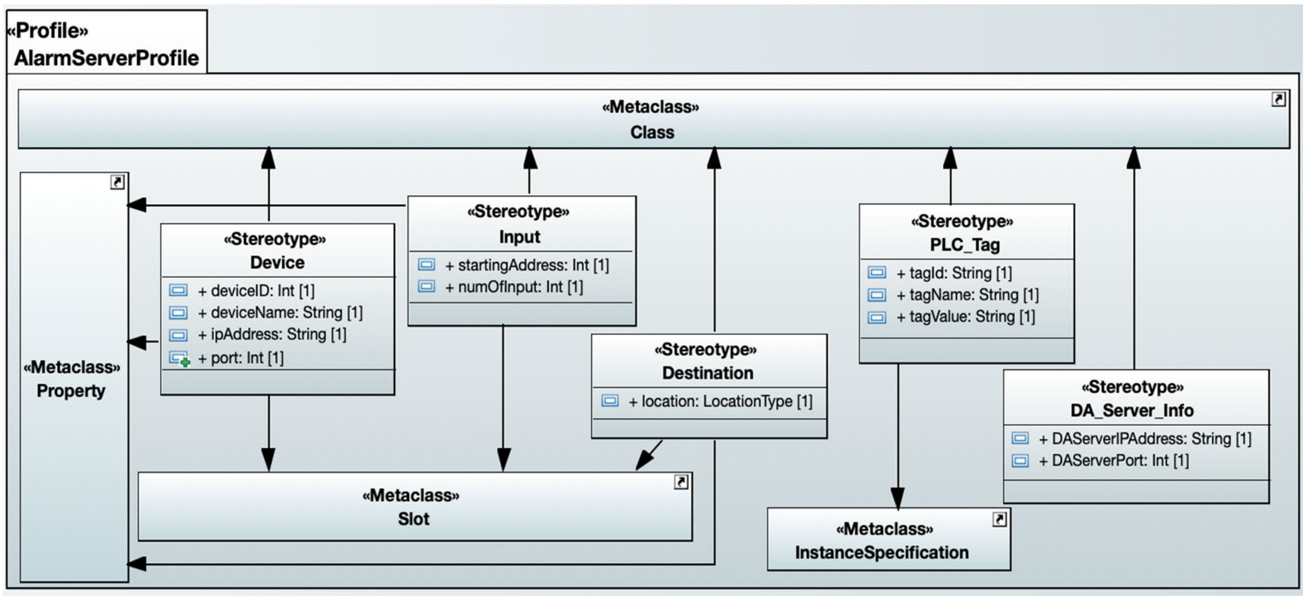


FIGURE 6 a. APICS Profile – Alarm Server Concepts

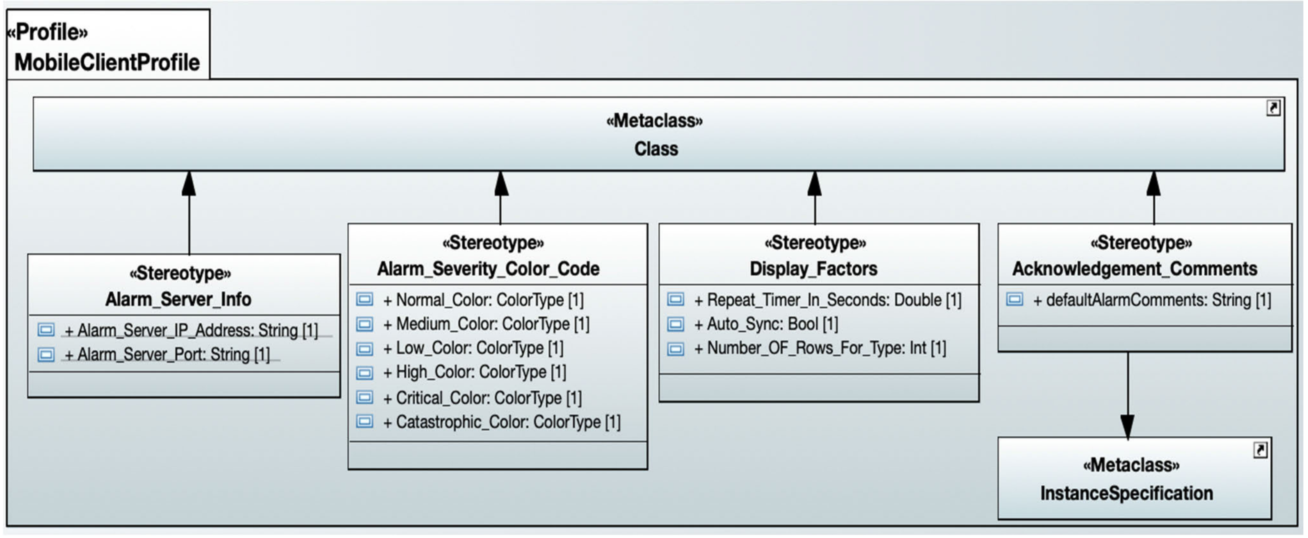

«Profile»
MobileClientProfile
«Metaclass»
Class
«Stereotype»
Alarm_Server_Info
+ Alarm_Server_IP_Address: String [1]
+ Alarm_Server_Port: String [1]
«Stereotype»
Alarm_Severity_Color_Code
+ Normal_Color: ColorType [1]
+ Medium_Color: ColorType [1]
+ Low_Color: ColorType [1]
+ High_Color: ColorType [1]
+ Critical_Color: ColorType [1]
+ Catastrophic_Color: ColorType [1]
«Stereotype»
Display_Factors
+ Repeat_Timer_In_Seconds: Double [1]
+ Auto_Sync: Bool [1]
+ Number_OF_Rows_For_Type: Int [1]
«Stereotype»
Acknowledgement_Comments
+ defaultAlarmComments: String [1]
«Metaclass»
InstanceSpecification

FIGURE 6 b. APICS Profile – Mobile Clients Concepts

FIGURE 6. a. APICS profile – alarm server concepts. b. APICS profile – mobile clients concepts. c. APICS profile – shared concepts. d. APICS profile – data types. e. APICS profile – enumerations.

an alarm serves as a notification agent for the human operators to take effective measures accordingly. The window for an operator to take a reactive measure may not be big in industries. Therefore, timely and accurate information is required for a human operator in a presentable form. The color-coding technique is highly effective as it speeds the recognition process. The severe effect of an alarm's current readings from a device is presented to an operator using color codes. The appropriate values can be set for each alarm severity state using these attributes *Normal_Color, Low_Color, Medium_Color, High_Color, Critical_Color, Catastrophic_Color*. All the color attributes can have few possible values that are expressed using an enumeration *ColorType* in **Figure 6e.**

≪Display_Factors≫

This stereotype is extended from UML meta-class named Class. On launch of a mobile client application, it automatically establishes a connection with Alarm Server and requests a list of alarms to show. However, to keep the mobile client up to date, it needs to continuously keep in touch with server and present real-time information to the human operator. In this context, time interval can be set i.e. *Repeat_Timer_In_Seconds* to send request to Alarm Server for updates and value for *Auto_Sync* to update the mobile

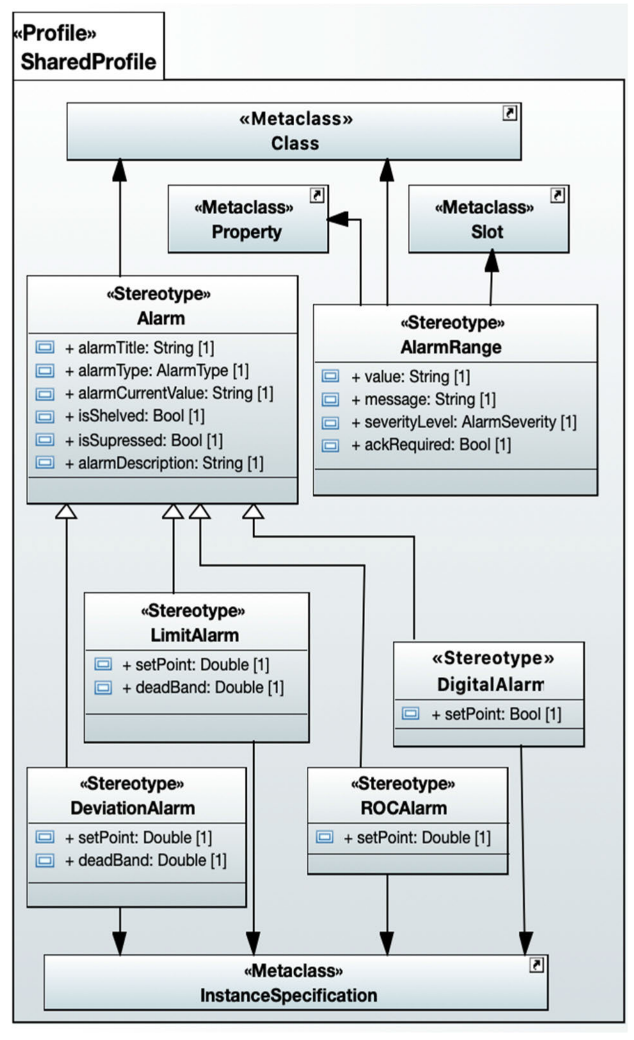


FIGURE 6 c. APICS Profile – Shared Concepts

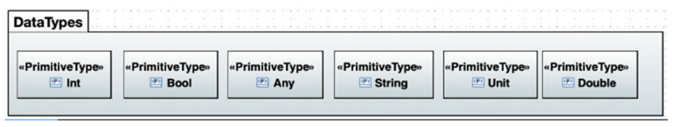


FIGURE 6 d. APICS Profile – Data Types

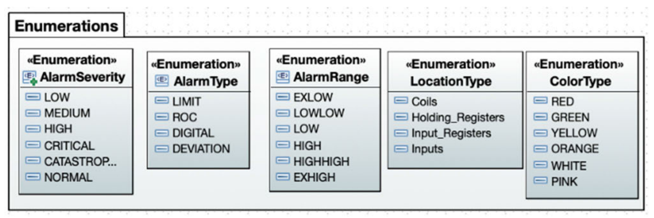


FIGURE 6 e. APICS Profile – Enumerations

FIGURE 6. ***(Continued.)*** **a. APICS profile – alarm server concepts. b. APICS profile – mobile clients concepts. c. APICS profile – shared concepts. d. APICS profile – data types. e. APICS profile – enumerations.**

interface without explicitly refreshing the screen. There can be tens of alarms in a single industry during particular instance. For this, number of records for a particular type of alarm to be shown on the screen can be set through *Number_OF_Rows_For_Type*.

≪Acknowledgement_Comments≫

This stereotype is extended from UML meta-classes named Class and Instance Specification. This stereotype enables human operator to acknowledge particular alarms and specify comments if required. The *defaultAlarmComments* may be configured using this stereotype.

3) SUB PROFILE FOR SHARED CONCEPTS BETWEEN SERVER AND CLIENTS

The third sub profile, named as "SharedProfile" in **Figure 6c**, is used to represent the shared requirements between the Alarm Server and Alarm Viewers (i.e. mobile clients). It contains the following elements i.e. Alarm, Alarm Range, and the four major types of alarms i.e. Limit Alarm, Deviation Alarm, RoC Alarm and Digital Alarm.

≪Alarm≫

This stereotype is extended from UML meta-classes named Class and Instance Specification. The attributes of this stereotype are quite self-explanatory. The title of an alarm is denoted by *alarmTitle* and to document the type of a particular alarm, *alarmType* is used. The real-time value fetched from an Alarm Server is stored in *alarmCurrentValue* to show on the mobile client's screen. *isShelved* depicts whether an alarm is to be activated or displayed on the mobile client. Whereas *isSupressed*, if set to true, portrays that this alarm should be silent and shouldn't alert the human operator until set otherwise. The default description of an alarm is stored in *alarmDescription* attribute.

≪AlarmRange≫

This stereotype is extended from UML meta-classes named Class, Property and Slot. It allows to specify a range of normal operations and a flexibility to set extremely high and low values and to refine the limit by specifying different levels of operations i.e. *ExLow, LowLow, Low, High, HighHigh, ExHigh.* For each range instance, its corresponding severity can be set from the enumeration *AlarmSeverity* (**Figure 6e**). The attribute *ackRequired*, if set to true shows that alarm needs to be acknowledged. Finally, the *message* provides a brief description of necessary reactive measures to be displayed on client side.

Each Alarm type has a *setpoint* – an activation point that triggers the alarm into the active state. Configuring an alarm of any specialized type requires defining a range of normal operations that are associated, using AlarmRange attributes. LimitAlarm and DeviationAlarm compose six additional attributes of type AlarmRange (i.e. ExLow, LowLow, Low, High, HighHigh, ExHigh) whereas ROCAlarm and DigitalAlarm compose one AlarmRange attribute in our domain model as shown in **Figure 10**. These attributes contain information about abnormal range value, its severity and message containing reactive measures for human operator. Deviation Alarm and Limit Alarm contain an additional attribute, *deadband* that indicates the change required in the process signal to either trigger the alarm into active state or neutral one. Each alarm type requires single PLC tag to function except Deviation Alarm which requires two associated tags.

Limit Alarm accepts analog inputs and provides discrete output indications for each Alarm Range (i.e. ExLow, LowLow, Low, Medium, High, HighHigh, ExHigh) depending upon the input value. For instance, Limit Alarm can be used to monitor voltage or current input signals.

Rate of Change (ROC) alarm type is primarily used to track the rapid changing conditions, exceeding setpoint, to trigger an alarm. For instance, to monitor rate of water flow in a turbine or flour mill etc. It isn't applied on values that are not changing fast enough. It is considered one of the most precise method of setting alarms as it can also be used to track the wear generation rate. The collected data over time is used to make better estimation of machine's condition. On the other hand, Digital Alarm operates on digital value (true or false).

≪LimitAlarm≫

This stereotype is extended from UML meta-classes named Class and Instance Specification. The attribute *setPoint* in this stereotype is used to set the default value at the time of creating a LimitAlarm instance. Whereas *deadBand* portrays the bare minimum value of Limit Alarm.

≪DeviationAlarm≫

This stereotype is extended from UML meta-classes named Class and Instance Specification. The attribute *setPoint* is used to set the default value at the time of creating a Deviation Alarm instance. Whereas *deadBand* portrays the bare minimum value of DeviationAlarm.

≪ROCAlarm≫

This stereotype is extended from UML meta-classes named Class and Instance Specification. The attribute *setPoint* is used to set the default value at the time of creating a ROCAlarm instance.

≪DigitalAlarm≫

This stereotype is extended from UML meta-classes named Class and Instance Specification. The attribute setPoint is used to set the default value at the time of creating a DigitalAlarm instance.

4) DATA TYPES AND ENUMERATIONS

Figure 6e also shows all the enumerations involved while developing the entire APICS. These enumerations represent the possible values for concepts such as AlarmType, AlarmRange, AlarmSeverity and ColorType. Available AlarmRange is to specify abnormal values, severity of each abnormal value is specified using AlarmSeverity, and to color code each severity level ColorType is used. These enumerations enhance available options while configuring an alarm. Furthermore, the data types involved in the APICS concepts are shown in **Figure 6d.**

IV. IMPLEMENTATION

So far, the details about the development of APICS along with purpose of each stereotype are described. In this section, the implementation details regarding Alarms Profile Transformation Engine (APTE) are provided.

A. ARCHITECTURE OF APTE

The architecture of APTE is graphically presented in **Figure 7.** There are two major components of APTE i.e. User Interface (UI) and Transformer.

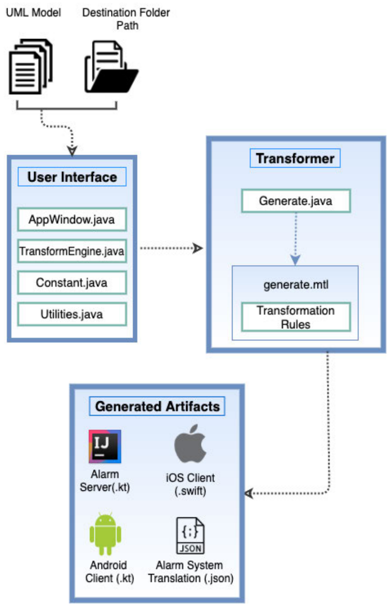
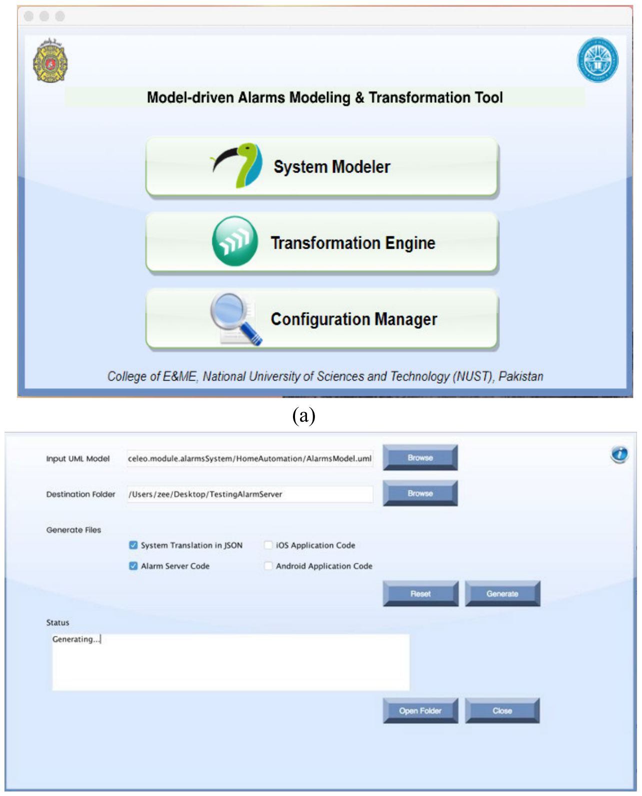


FIGURE 7. Architecture of APTE.

The transformation is carried out using Acceleo – an open source Eclipse plugin for Model-to-Text (M2T) transformation [42]. The languages used for transformation is JAVA and Acceleo transformation language.

1) USER INTERFACE

The main interface of APTE is shown in **Figure 8a**. It comprises of three features i.e. System Modeler, Alarms Configuration Manager and Transformation Engine. System Modeler facilitates in modeling the APICS in Papyrus – an Eclipse plugin for modeling domain specific concepts. The Alarms Configuration Manager provides the UI editor to modify the generated document (through APTE) containing tags and alarms in JSON format. Lastly, transformation engine transforms the input models to actual deployable code.


Model-driven Alarms Modeling & Transformation Tool
System Modeler
Transformation Engine
Configuration Manager
College of E&ME, National University of Sciences and Technology (NUST), Pakistan
Browse
Browse
Reset
Generate
Open Folder
Close

(a)

Input UML Model: celeo.module.alarmsSystem/HomeAutomation/AlarmsModel.uml

Destination Folder: /Users/zee/Desktop/TestingAlarmServer

Generate Files

[✓ System Translation in JSON] iOS Application Code

[✓ Alarm Server Code] Android Application Code

Status

Generating...

(b)

FIGURE 8. a. Main interface of APTE. b. Transformation engine interface.

The interface of transformation engine is shown in **Figure 8b**. It takes the input of APICS-compliant UML model, desired destination folder and information about target models check boxes. The input fields can be reset using Reset button. The status about transformation process is displayed on status bar as shown in **Figure 8b**. Finally, the Open Folder button directs to the destination folder where required files are generated.

It is important to note that we have also developed alarm configuration component to update the configuration of alarms which are generated through transformation engine in JSON format. However, we are not including the details of alarm configuration component here due to space limitations. In this regard, the source code of APTE along with user manual, and sample case studies can be found for further evaluation at [39].

2) TRANSFORMER

Transformer is responsible to generate the low-level alarm server, clients and configuration implementations from APICS models. The input APICS-compliant model is passed to generate.mtl file that holds the Acceleo code to transform the given model into target artifacts as shown in Figure 7. This component contains two files Generate (Generate.java) and Template (generate.mtl). The Template file contains reference of sub-templates and other.mtl files. It is responsible for fetching the input UML model for processing and passes it to sub templates. Each sub template implements transformation rules on different UML elements to generate the output. The generated code in the target folder consists of JSON translation of the Alarm system, deployable Alarm server code, iOS application client code and Android application client code.

B. TRANSFORMATION RULES

This section narrates the mapping rules used to transform the APICS models into deployable alarm server, clients and configuration artifacts. Particularly, we have developed three types of transformation rules in order to correctly transform high level models into required target artifacts. The description of transformation rules is given in **Table 2, Table 3 and Table 4.**

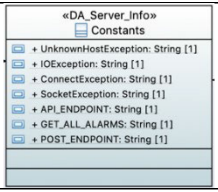
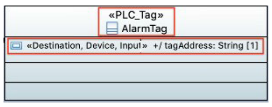
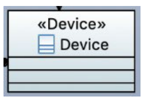
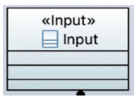
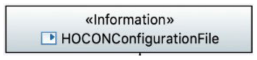
In first step, the transformation rules are developed to generate alarm server implementations from models as given in **Table 2.** Particularly, server concepts like *Constants* – comprising of DAServer ipAddress and port, endpoints, exception messages etc. *AlarmTag* – comprising tagAddress – is associated with each alarm instance to specify physical location of sensors over the network. *Device, Destination* and *Input* collectively make a tagAddress of an AlarmTag.

The abovementioned concepts are mapped to their equivalent in Kotlin language such as for modeling concept AlarmTag; model class is mapped to *data class AlarmTag* having file name AlarmTag.kt. The model class properties and applied stereotypes on model class are mapped as data class member variables. Whereas applied stereotypes on model class properties are mapped as derived member variables of data class.

Additionally, few concepts regarding files are used which are required by the alarm server to compile and perform required functionality. The details are: *Alarms.json* – file comprising of alarms instances in JSON format. *Build.Gradle* – dependency management file to download and configure required libraries. *Manifest* – file holding metadata about JAR file and the classes that application contains. *HOCON-ConfigurationFile* – file comprising of alarm server configurations. It is important to note that few shared concepts are also required for the generation of alarm server code as given in **Table 4.** These details are given in subsequent paragraph.

In second step, the transformation rules are developed to generate both iOS and Android alarm clients as given

TABLE 2. **Transformation rules for alarm server.**

Model Artifacts <<Applied Stereotype>>		**Code Artifacts (in Kotlin)**	**Mapping**
Transformation Rules for Alarm Server (Kotlin)			
Data Acquisition Server Information	 «DA_Server_Info» Constants + UnknownHostException: String [1] + IOException: String [1] + ConnectException: String [1] + SocketException: String [1] + API_ENDPOINT: String [1] + GET_ALL_ALARMS: String [1] + POST_ENDPOINT: String [1]	object Constants (filename: Constants.kt)	Model Class—Name → object Name Model Class—Properties → object static member variable Model Class—Applied Stereotype Properties → object static member variables
PLC Tag Information	 «PLC_Tag» AlarmTag «Destination, Device, Input» +/ tagAddress: String [1]	data class AlarmTag (filename: AlarmTag.kt)	Model Class—Name → data class Name Model Class—Properties → data class member variables Model Class—Applied Stereotype Properties → data class member variables Model Class—Property—Applied Stereotypes →data class—derived member variable
PLC Tag's Device Information		data class Device (filename: Device.kt)	Model Class—Name → data class Name Model Class—Applied Stereotype Properties → data class member variables
PLC Tag's location Information	«Destination» Destination	data class Destination (filename: Destination.kt)	Model Class—Name → data class Name Model Class—Applied Stereotype Properties → data class member variables
PLC Tag's starting address Information		data class Input (filename: Input.kt)	Model Class—Name → data class Name Model Class—Applied Stereotype Properties → data class member variables
JSON Translation of Alarm Instances	«Information» Alarms.json	Alarms instances in JSON format (filename: Alarms.json)	Alarm Instance → JSON object
Supporting Information Files for Alarm Server			
Dependency Manager	«Information» Build.Gradle	HOCON format	To automatically download and configure required Libraries and dependencies
Manifest File	«Information» Manifest	XML format	To hold metadata about JAR file and the classes that application contains.
App configuration Information		HOCON format	Alarm Server configuration file.

in Table 3. It primarily comprises of mapping of UML modeling constructs to programming language constructs for both iOS and Android clients in Swift and Kotlin language respectively. For example, the client modeling concept like *CommentsViewController* is a view concept in iOS and Android clients respectively to enter acknowledgement comments against a triggered alarm. The model class properties and applied stereotypes on model class are mapped as data class member variables. It is important to note that the transformation of few shared concepts is also required for the accurate transformation of alarm clients. The details are given in subsequent section.

As mentioned earlier, shared concepts of APICS need to be transformed in alarm server and alarm client's transformations respectively. The shared concepts transformation details are given in **Table 4.** Particularly, *Alarm* specialized types (i.e. *LimitAlarm, DeviationAlarm, ROCAlarm and DigitalAlarm), AlarmRange and Constants* are required by alarm server and both iOS and Android clients. Therefore, these concepts are mapped to their respective equivalents for Alarm

TABLE 3. Transformation rules for alarm clients.

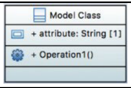
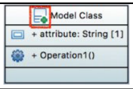
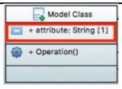
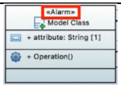
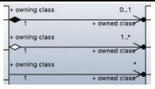
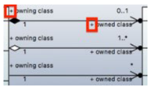
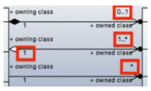
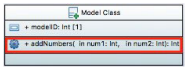
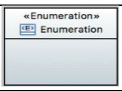
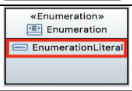
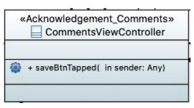
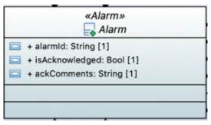
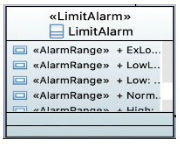
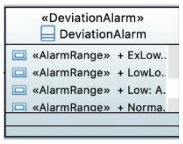
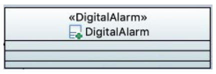
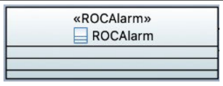
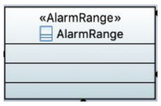
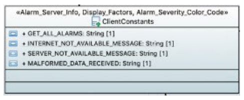
Model Artifacts		Code Artifacts		Mapping
		iOS Client (Swift)	Android Client / Alarm Server (Kotlin)	
Package		Folder	Folder	Package—Name → Folder Name
		Model Class		
Name		class Name	data class Name	Model Class—Name → class/data class Name
Visibility				Model Class—Visibility → Access specifiers class/data class
Attributes				Model Class—Property → Owned Attributes
Applied Stereotype	 «Alarm» / Model Class / + attribute: String [1] / + Operation()			Model Class—Applied Stereotype—Property → Owned Attributes
		Association		
Member End		Class Instance	Class Instance	Association—Member end → Owning Class/ Owned Class.
Member End Visibility		public/ private/ protected/ derived	public/ private/ protected/ derived	Association—Member end visibility → Owned Instance Access specifier.
Member End Cardinality		Array <Type> Instance Name OR Single Class Instance	List <Type> Instance Name OR Single Class Instance	Association—Member end Multiplicity/ Cardinality → Collection or single instance.
		Class Operation		
Operation Signatures	 Model Class / + modelID: Int [1] / + addNumbers(in num1: Int, in num2: Int): Int	function operation name and signature	function operation name and signature	Operation—Name → fun/func Method Name. Operation—Owned Parameter → Method (In, Out) Parameter, Return Type of Method.
Operation Opaque Behavior		Method Body implementation	Method Body implementation	Opaque Behavior—Description → Body of Method.
		Enumeration		
Enumeration		enum Enumeration	enum class Enumeration	Enumeration—Name → enum/enum class Name
Enumeration Literal		case Enumeration Literal	Enumeration Literal	Enumeration—Owned Literal → Enum Values.
		Views		
Alarm acknowledgement Information	 «Acknowledgement_Comments» / CommentsViewController / + saveBtnTapped(in sender: Any)	class CommentsViewController: UIViewController (filename: CommentsViewController.swift	class CommentActivity: AppCompatActivity (), View.OnClickListener (filename: CommentActivity)	Model Class—Name-> class Name Model Class—Applied Stereotype Properties → struct/class member variables Model Class—Operation → struct/class method

TABLE 4. Transformation rules for shared concepts b/w alarm server and clients.

Model Artifacts <<Applied Stereotype>>		**Code Artifacts**		**Mapping**
		iOS Client (in Swift)	**Android Client / Alarm Server (Kotlin)**	
Alarm Information	 «Alarm» Alarm + alarmId: String [1] + isAcknowledged: Bool [1] + ackComments: String [1]	class Alarm: Codable (filename: Alarm.swift)	data class Alarm (filename: Alarm.kt)	Model Class—Name → class Name Model Class—Properties → class member variables Model Class—Applied Stereotype Properties → class member variables
Limit Alarm Information	 «LimitAlarm» LimitAlarm «AlarmRange» + ExLo... «AlarmRange» + LowL... «AlarmRange» + Low: .. «AlarmRange» + Norm.. «AlarmRange» + High:	class LimitAlarm: Alarm (filename: LimitAlarm.swift)	data class LimitAlarm: Alarm (filename: LimitAlarm.kt)	Model Class—Name → class Name Model Class—Properties → class member variables Model Class—Applied Stereotype Properties → class member variables Model Class—Property—Applied Stereotypes → class—derived member variable
Deviation Alarm Information	 «DeviationAlarm» DeviationAlarm «AlarmRange» + ExLow.. «AlarmRange» + LowLo.. «AlarmRange» + Low: A.. «AlarmRange» + Norma..	class DeviationAlarm: Alarm (filename: DeviationAlarm.swift)	data class DeviationAlarm: Alarm (filename: DeviationAlarm.kt)	Model Class—Name → class Name Model Class—Properties → class member variables Model Class—Applied Stereotype Properties → class member variables Model Class—Property—Applied Stereotypes → class—derived member variable
Digital Alarm Information	 «DigitalAlarm» DigitalAlarm	class DigitalAlarm: Alarm (filename: DigitalAlarm.swift)	data class DigitalAlarm: Alarm (filename: DigitalAlarm.kt)	Model Class—Name → class Name Model Class—Applied Stereotype Properties → class member variables
ROC Alarm Information	 «ROCAlarm» ROCAlarm	class ROCAlarm: Alarm (filename: ROCAlarm.swift)	data class ROCAlarm: Alarm (filename: ROCAlarm.kt)	Model Class—Name → class Name Model Class—Applied Stereotype Properties → class member variables
Alarm Range Information	 «AlarmRange» AlarmRange	class AlarmRange: Codable (filename: AlarmRange.swift)	data class AlarmRange (filename: AlarmRange.kt)	Model Class—Name → class Name Model Class—Applied Stereotype Properties → class member variables
Alarm Server & Display Settings information	 «Alarm_Server_Info, Display_Factors, Alarm_Severity_Color_Code» ClientConstants + GET_ALL_ALARMS: String [1] + INTERNET_NOT_AVAILABLE_MESSAGE: String [1] + SERVER_NOT_AVAILABLE_MESSAGE: String [1] + MALFORMED_DATA_RECEIVED: String [1]	struct Constants (filename: Constants.swift)	object Constants (filename: Constants.kt)	Model Class—Name → struct/class Name Model Class—Properties → struct/class member variables Model Class—Applied Stereotype Properties → struct/class member variables

Server and Android client in Kotlin language. Whereas for iOS client, these concepts are mapped in Swift language as given in **Table 4.** For example, regarding Alarm Server and Android client, model class is mapped to *data class Alarm* having file name as Alarm.kt. Whereas for iOS client, model class is mapped as *class Alarm* with filename as Alarm.swift. Similarly, the model class properties and applied stereotypes on model class *Alarm* are mapped as data class member variables. Whereas applied stereotypes on model class properties are mapped as derived member variables of data class.

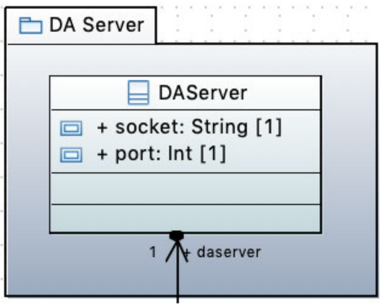
Here, the brief summary of major transformation rules is provided. For further information, the working source code of APTE along with user manual, and other relevant documents can be found at [39].

V. VALIDATION

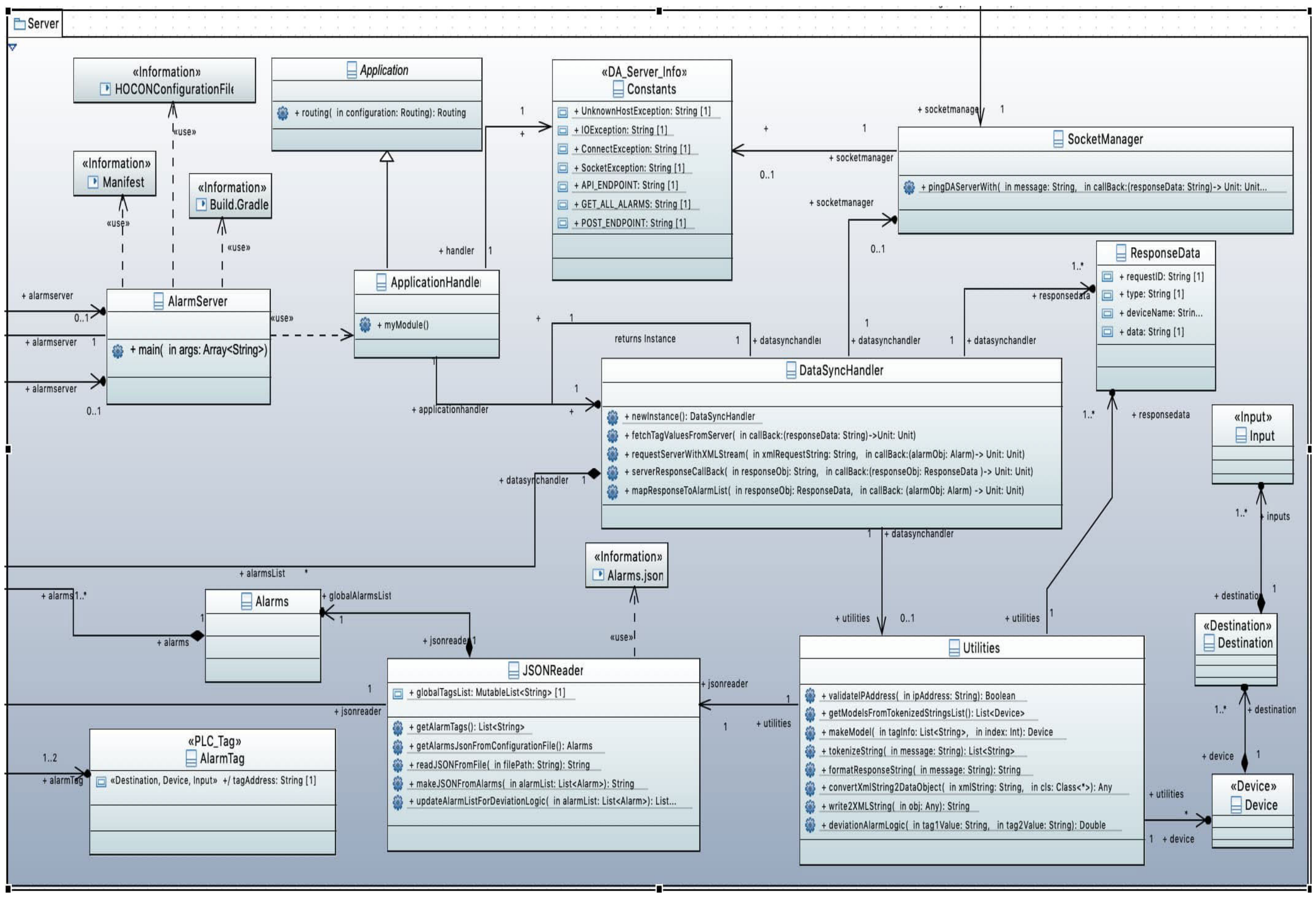
In this section, the applicability of MAF is demonstrated through Flour Mill (Section V-B) and Home Automation (Section V-C) case studies.

A. EXPERIMENTAL SETUP

The validation of MAF is performed in our institution's control lab. An open source Modbus-based Data Acquisition (DA) server is used for the validation of MAF [12]. To develop realistic environment, two Fatek PLCs [38] are utilized where 300 tags are configured to fetch real time values from DA server in order to process alarms accordingly. The distribution of configured tags is as follows: 100 tags belong to *holding registers,* 50 tags belong to *coils,* 100 tags



(a)


Server
«Information» HOCONConfigurationFile
Application
+ routing(in configuration: Routing): Routing
«DA_Server_Info» Constants
+ UnknownHostException: String [1]
+ IOException: String [1]
+ ConnectException: String [1]
+ SocketException: String [1]
+ API_ENDPOINT: String [1]
+ GET_ALL_ALARMS: String [1]
+ POST_ENDPOINT: String [1]
SocketManager
+ pingDAServerWith(in message: String, in callBack:(responseData: String)-> Unit: Unit...
«Information» Manifest
«Information» Build.Gradle
«use»
AlarmServer
+ main(in args: Array<String>)
ApplicationHandler
+ myModule()
ResponseData
+ requestID: String [1]
+ type: String [1]
+ deviceName: Strin...
+ data: String [1]
DataSyncHandler
+ newInstance(): DataSyncHandler
+ fetchTagValuesFromServer(in callBack:(responseData: String)->Unit: Unit)
+ requestServerWithXMLStream(in xmlRequestString: String, in callBack:(alarmObj: Alarm)-> Unit: Unit)
+ serverResponseCallBack(in responseObj: String, in callBack:(responseObj: ResponseData)-> Unit: Unit)
+ mapResponseToAlarmList(in responseObj: ResponseData, in callBack: (alarmObj: Alarm) -> Unit: Unit)
«Input» Input
«Information» Alarms.json
Alarms
JSONReader
+ globalTagsList: MutableList<String> [1]
+ getAlarmTags(): List<String>
+ getAlarmsJsonFromConfigurationFile(): Alarms
+ readJSONFromFile(in filePath: String): String
+ makeJSONFromAlarms(in alarmList: List<Alarm>): String
+ updateAlarmListForDeviationLogic(in alarmList: List<Alarm>): List...
Utilities
+ validateIPAddress(in ipAddress: String): Boolean
+ getModelsFromTokenizedStringsList(): List<Device>
+ makeModel(in tagInfo: List<String>, in index: Int): Device
+ tokenizeString(in message: String): List<String>
+ formatResponseString(in message: String): String
+ convertXmlString2DataObject(in xmlString: String, in cls: Class<*>): Any
+ write2XMLString(in obj: Any): String
+ deviationAlarmLogic(in tag1Value: String, in tag2Value: String): Double
«Destination» Destination
«Device» Device
«PLC_Tag» AlarmTag
«Destination, Device, Input» +/ tagAddress: String [1]
+ alarmserver
+ handler
+ socketmanager
+ datasynchandler
+ responsedata
+ applicationhandler
+ alarmsList
+ alarms
+ globalAlarmsList
+ jsonreader
+ utilities
+ inputs
+ destination
+ device
+ alarmTag
returns Instance

(b)

FIGURE 9. **a. Domain model of data acquisition server concept. b. Domain model of alarm server concepts.**

belong to *input registers and* 50 tags belong to *discrete inputs*. Several alarms are constructed and processed through MAF by utilizing configured tags. Subsequently, the correctness of alarm server and mobile clients, designed and transformed through MAF, is evaluated. The actual alarms situations of two case studies (i.e. Flour Mill and Home Automation) have been considered for the experimentation. It is important to note that although we consider real PLCs for experimentation, the actual sensors are not attached with PLCs due to limited resources. In this regard, we alter the values of PLCs tags through DA server using *write* operation in order to confirm the desired execution of alarms and mobile clients. We utilize Android Studio IDE and Xcode IDE for the validation of Android and iOS alarm clients respectively.

B. FLOUR MILL CASE STUDY

1) REQUIREMENTS

In flour mill, there are many sensors involved such as smoke detectors, fire sensors, solid flow detection sensors, their speed, dust sensors, concentration of oxygen on the floor, liquid level monitoring sensors etc. These sensors are usually

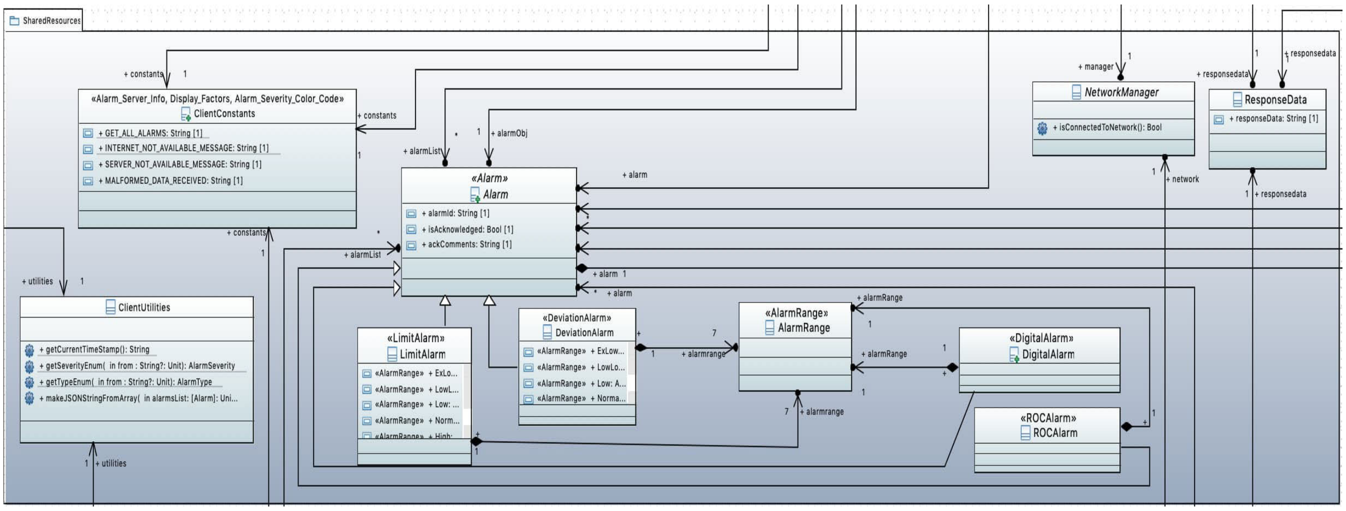

SharedResources
«Alarm_Server_Info, Display_Factors, Alarm_Severity_Color_Code»
ClientConstants
+ GET_ALL_ALARMS: String [1]
+ INTERNET_NOT_AVAILABLE_MESSAGE: String [1]
+ SERVER_NOT_AVAILABLE_MESSAGE: String [1]
+ MALFORMED_DATA_RECEIVED: String [1]
«Alarm»
Alarm
+ alarmId: String [1]
+ isAcknowledged: Bool [1]
+ ackComments: String [1]
ClientUtilities
+ getCurrentTimeStamp(): String
+ getSeverityEnum(in from : String?: Unit): AlarmSeverity
+ getTypeEnum(in from : String?: Unit): AlarmType
+ makeJSONStringFromArray(in alarmsList: [Alarm]: Uni...
«LimitAlarm»
LimitAlarm
«DeviationAlarm»
DeviationAlarm
«AlarmRange»
AlarmRange
«DigitalAlarm»
DigitalAlarm
«ROCAlarm»
ROCAlarm
NetworkManager
+ isConnectedToNetwork(): Bool
ResponseData
+ responseData: String [1]

FIGURE 10. Domain model of shared concepts between server and clients.

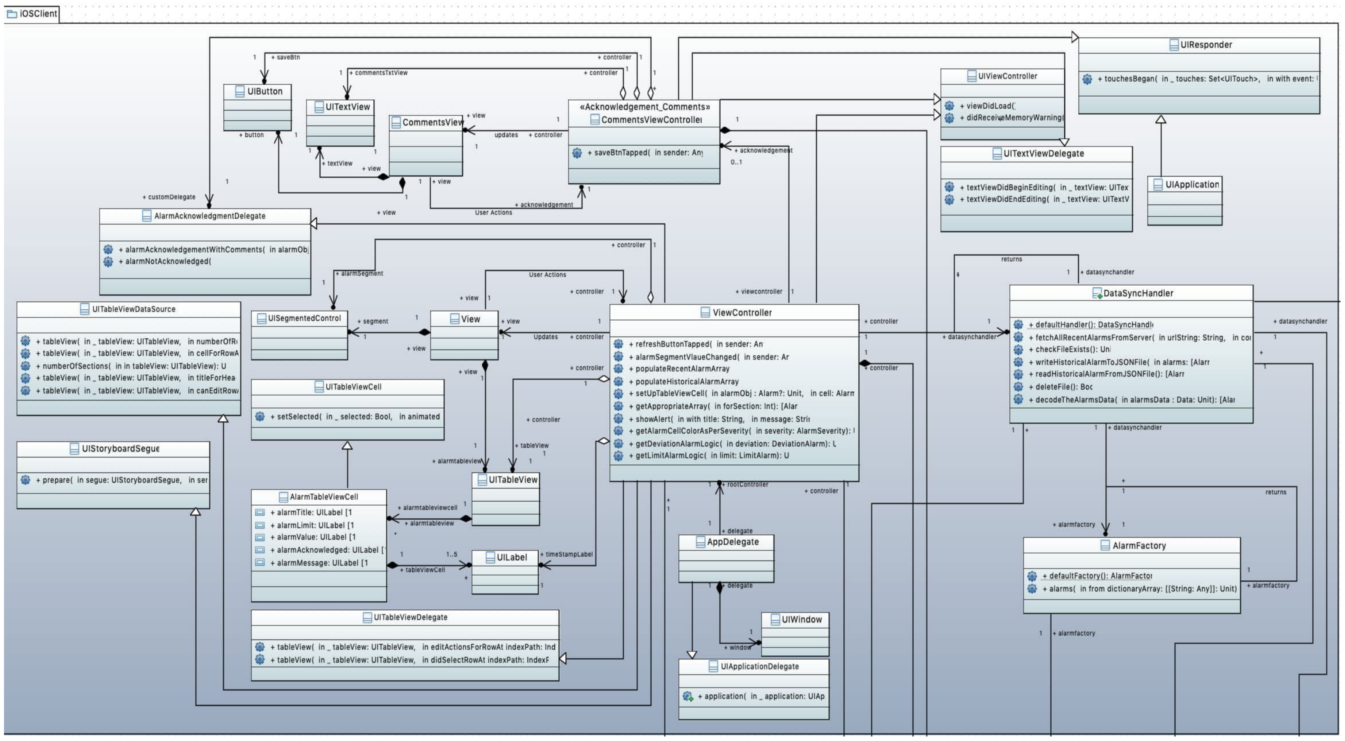

iOSClient
UIButton
UITextView
CommentsView
«Acknowledgement_Comments»
CommentsViewController
+ saveBtnTapped(in sender: Any
UIViewController
+ viewDidLoad(
+ didReceiveMemoryWarning
UIResponder
+ touchesBegan(in _ touches: Set<UITouch>, in with event:
UITextViewDelegate
+ textViewDidBeginEditing(in _ textView: UITex
+ textViewDidEndEditing(in _ textView: UITextV
UIApplication
AlarmAcknowledgmentDelegate
+ alarmAcknowledgementWithComments(in alarmObj
+ alarmNotAcknowledged(
UITableViewDataSource
UISegmentedControl
View
ViewController
DataSyncHandler
UITableViewCell
UIStoryboardSegue
+ prepare(in segue: UIStoryboardSegue, in ser
AlarmTableViewCell
UITableView
UILabel
AppDelegate
AlarmFactory
+ defaultFactory(): AlarmFactor
+ alarms(in from dictionaryArray: [[String: Any]]: Unit)
UIWindow
UITableViewDelegate
UIApplicationDelegate
+ application(in _ application: UIAp

FIGURE 11. Domain model of iOS alarm client.

connected as 'tags'' with PLCs for control and monitoring. Different alarming conditions are monitored for particular purposes. For example, one requirement is to generate alarm if pollution exceeds the allowed limit depending upon the value of dust sensor. For this case study, we utilized different PLCs tags to configure and execute several alarms accordingly through MAF. The details of utilized tags and configured alarms are as follows:

No of Tags

30 holding / input registers are utilized to represent different sensors in PLCs like dust sensor, liquid level monitoring sensor etc. Furthermore, 15 coils / discrete input registers are utilized to represent sensors with Boolean values e.g. switches, buttons etc.

No of Alarms

Overall, the requirements of 21 alarms are modeled through MAF i.e. Limit (5), Digital (11), RoC (4) and Deviation (1). Subsequently, transformation process is executed to automatically generate alarm server and mobile clients. Six alarms are modeled for the control room and remaining fifteen alarms are modeled for factory floor as shown in **Figure 13.**

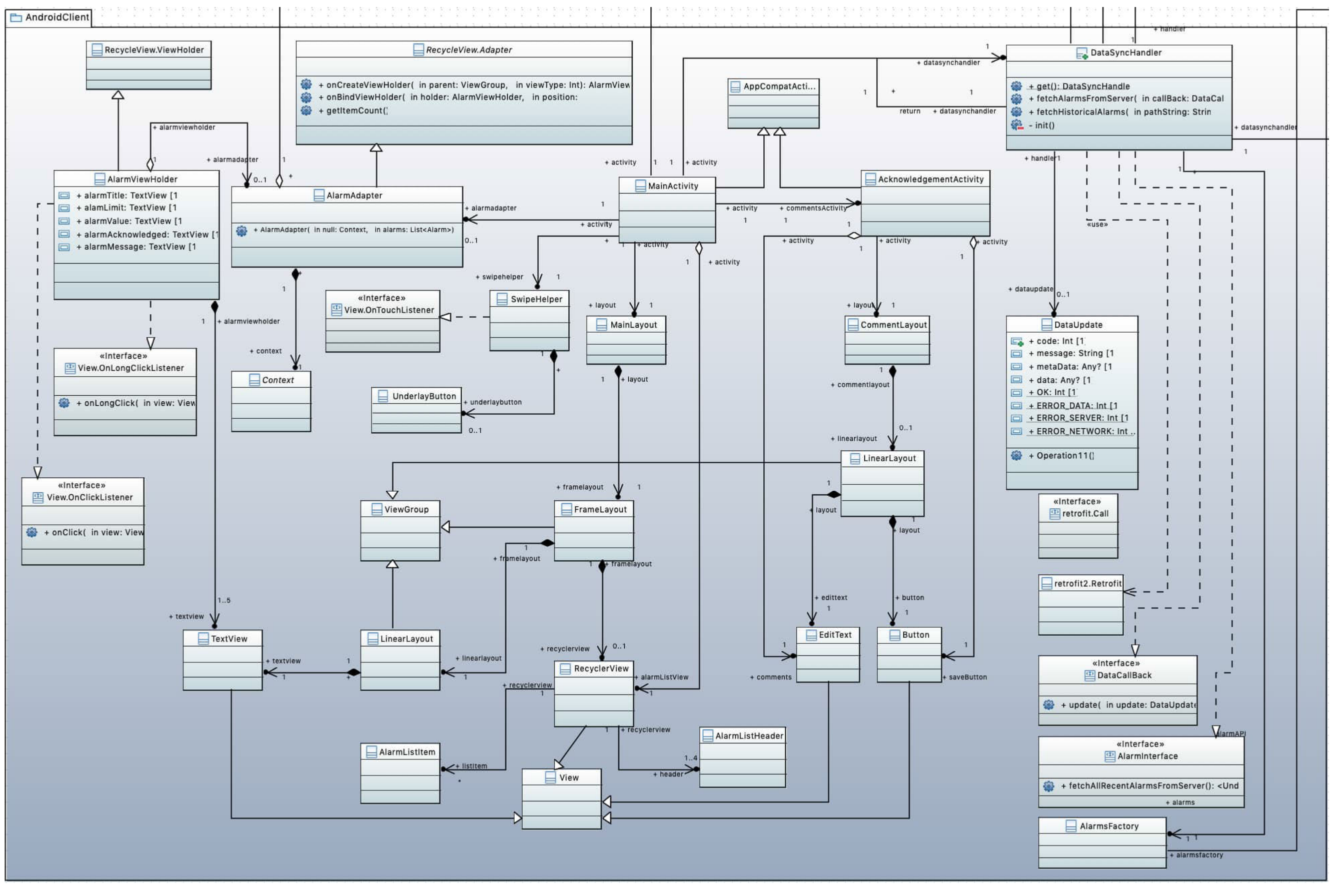


FIGURE 12. Domain model of android alarm client.

2) MODELING

The complete domain model of alarms for flour mill is shown in Figure 9a, Figure 9b, Figure 10, Figure 11 and Figure 12. It contains four types of domain concepts i.e. Data Acquisition (DA) Server, Alarm Server, native iOS Client and native Android Client. The designed models contain attributes, operations along with their applied stereotypes that are required for proper execution. The Figure 13 shows the instance specification of APICS domain model for Flour Mill system. It contains multiple instances of alarm of different types. Each alarm is assigned a tag(s) value depending upon its type e.g. Deviation type alarms must have two tags assigned to them whereas the rest of the types require only a single tag. Each tag contains complete information about the intended PLC device like name, IP address, port, memory location etc. Figure 14 shows assignment of values to an alarm instance of Flour Mill.

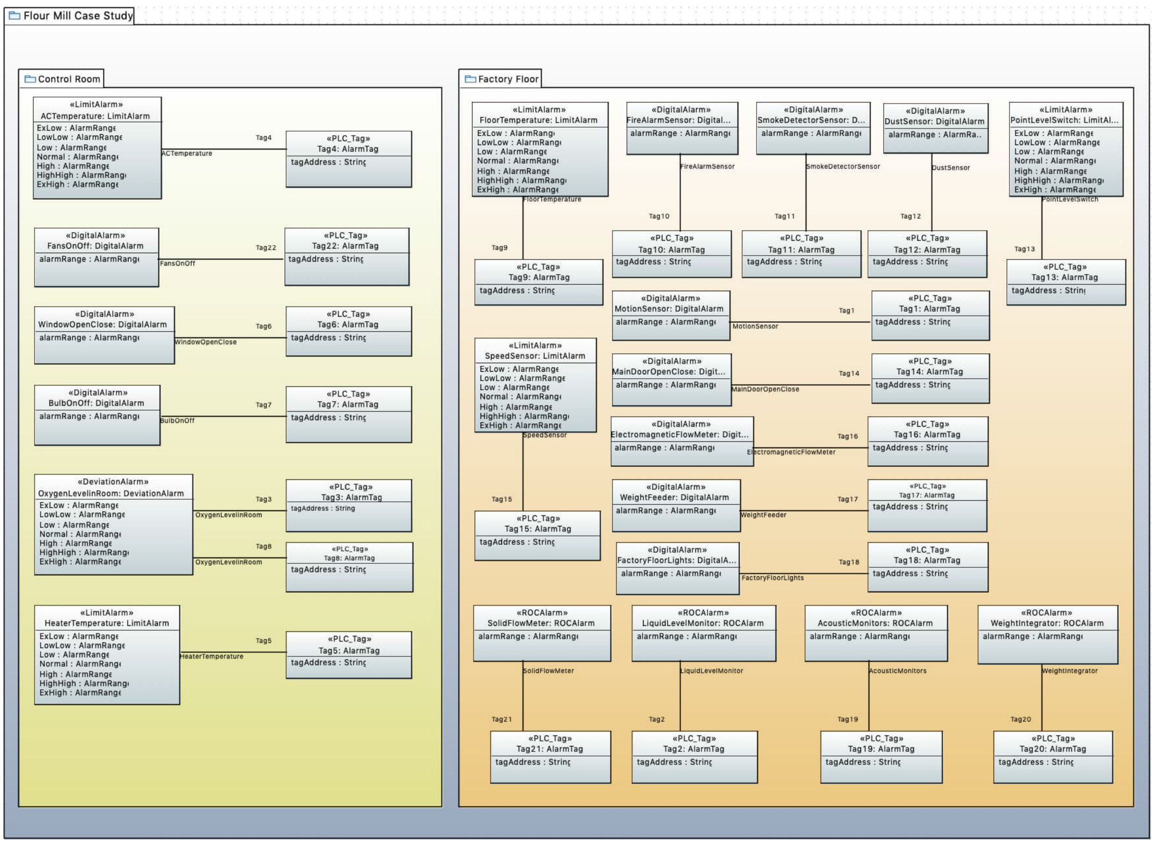
3) CODE GENERATION

APTE takes the domain model (.uml file) of Flour Mill as an input and automatically transforms it into target code artifacts i.e. Alarms Configuration (JSON file), Alarm Server, Android client and iOS client. For further evaluation, the generated source code of the target artifacts along with the screenshots of instance model of Flour Mill case study can be found at [39].

4) DEPLOYMENT AND VERIFICATION

The generated code artifacts i.e. Alarms JSON, Alarm Server, iOS client and Android client are ready to be deployed using their respective IDEs. For Alarm Server, we used IntelliJ IDEA CE. Firstly, we create an empty project with Gradle and then copy all the generated Alarm Server files in it. Additionally, with Alarm Server code, the generated Alarms JSON file is also required. Subsequently, the compilation of source code is performed for syntax error(s) and execution as shown in **Figure 15**. Similarly, for Android client, we utilize Android Studio IDE. Firstly, an empty project is created with Gradle and then copy / paste the generated Android client code. Subsequently, the android client is executed through emulator to show real time and historical alarms as shown in **Figure 17a** an **d Figure 17b** respectively. For iOS client, we utilize Xcode IDE where an empty project is created and then copy /paste the generated code files in the project. Subsequently, iOS client is executed through iOS simulator for real time monitoring of alarms as shown **in Figure 16a** and **Figure 16b.**

Here, we provide the brief description regarding modeling, transformation and execution of Flour Mill alarms to establish



Flour Mill Case Study
Control Room
Factory Floor
«LimitAlarm» ACTemperature: LimitAlarm
«PLC_Tag» Tag4: AlarmTag
«DigitalAlarm» FansOnOff: DigitalAlarm
«PLC_Tag» Tag22: AlarmTag
«DigitalAlarm» WindowOpenClose: DigitalAlarm
«PLC_Tag» Tag6: AlarmTag
«DigitalAlarm» BulbOnOff: DigitalAlarm
«PLC_Tag» Tag7: AlarmTag
«DeviationAlarm» OxygenLevelinRoom: DeviationAlarm
«PLC_Tag» Tag3: AlarmTag
«PLC_Tag» Tag8: AlarmTag
«LimitAlarm» HeaterTemperature: LimitAlarm
«PLC_Tag» Tag5: AlarmTag
«LimitAlarm» FloorTemperature: LimitAlarm
«DigitalAlarm» FireAlarmSensor: Digital...
«DigitalAlarm» SmokeDetectorSensor: D...
«DigitalAlarm» DustSensor: DigitalAlarm
«LimitAlarm» PointLevelSwitch: LimitAl...
«PLC_Tag» Tag9: AlarmTag
«PLC_Tag» Tag10: AlarmTag
«PLC_Tag» Tag11: AlarmTag
«PLC_Tag» Tag12: AlarmTag
«PLC_Tag» Tag13: AlarmTag
«DigitalAlarm» MotionSensor: DigitalAlarm
«PLC_Tag» Tag1: AlarmTag
«LimitAlarm» SpeedSensor: LimitAlarm
«DigitalAlarm» MainDoorOpenClose: Digit...
«PLC_Tag» Tag14: AlarmTag
«DigitalAlarm» ElectromagneticFlowMeter: Digit...
«PLC_Tag» Tag16: AlarmTag
«DigitalAlarm» WeightFeeder: DigitalAlarm
«PLC_Tag» Tag17: AlarmTag
«PLC_Tag» Tag15: AlarmTag
«DigitalAlarm» FactoryFloorLights: DigitalA...
«PLC_Tag» Tag18: AlarmTag
«ROCAlarm» SolidFlowMeter: ROCAlarm
«ROCAlarm» LiquidLevelMonitor: ROCAlarm
«ROCAlarm» AcousticMonitors: ROCAlarm
«ROCAlarm» WeightIntegrator: ROCAlarm
«PLC_Tag» Tag21: AlarmTag
«PLC_Tag» Tag2: AlarmTag
«PLC_Tag» Tag19: AlarmTag
«PLC_Tag» Tag20: AlarmTag
ExLow : AlarmRange
LowLow : AlarmRange
Low : AlarmRange
Normal : AlarmRange
High : AlarmRange
HighHigh : AlarmRange
ExHigh : AlarmRange
alarmRange : AlarmRange
tagAddress : String

FIGURE 13. Instance model containing alarms and devices information for flour mill.

the effectiveness of MAF. For further evaluation, the source code of APTE and APICS models of flour mill alarms can be found at [39].

C. *HOME AUTOMATION CASE STUDY*

Smart homes are also quite common concept that enables the constant monitoring of home appliances e.g. fans, lights switch buttons, temperature of a room, and even oxygen level in bedroom etc. In this case study, MAF is applied to model and transform several alarms in home automation situations.

1) REQUIREMENTS

In this case study, overall, 20 tags are considered where 10 holding / input registers are utilized to represent different sensors for the monitoring home appliances e.g. sensor for room temperature etc. Furthermore, 10 coils / discrete input registers are considered to represent sensors with Boolean values e.g. switch / button of fans and lights etc.

a: NO OF ALARMS

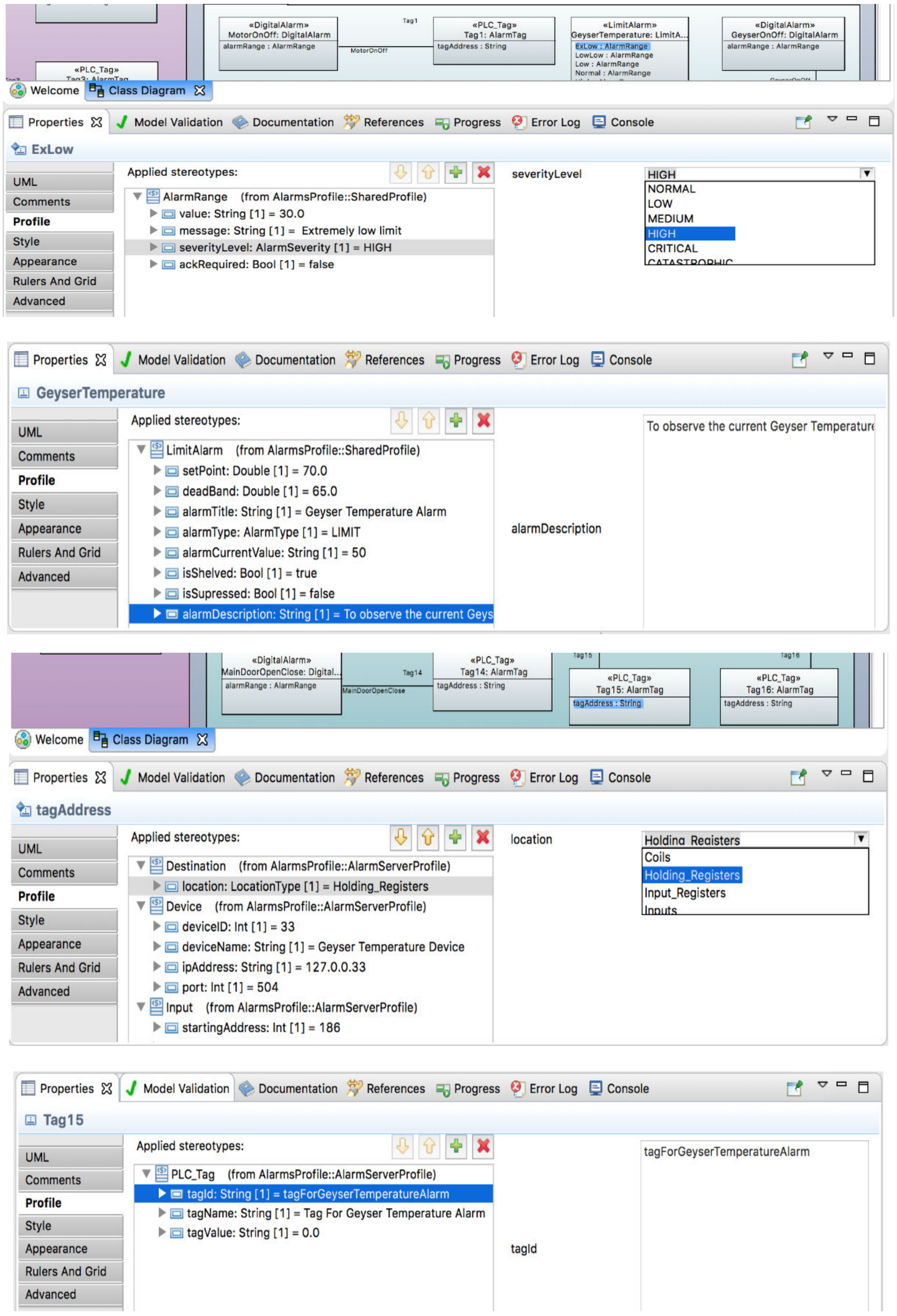
The modeling and transformation of 15 alarms is performed through MAF for home automation situations. Particularly, Limit (5), RoC (1), Digital (8) and Deviation (1) alarms are modeled through APICS. The five alarms are modeled for bedroom and ten alarms are modeled for kitchen and other areas as shown in **Figure 18.**

2) MODELING

The domain model of home automation alarms is modeled in first step. Subsequently, the instance specification of alarms for Home Automation system is modeled as shown in **Figure 18**. It contains multiples instances of alarm of different types. Each alarm is assigned a tag(s) value depending upon its type e.g. Deviation type alarms must have two tags assigned to them whereas the rest of the types require only a single tag. Each tag contains complete information about the intended PLC device such as name, IP address, port, memory location etc. It is important to note that we are not including the details of home automation domain model because such details are comprehensively explained in Section V-B-2 for flour mill case study.

3) CODE GENERATION, DEPLOYMENT AND VERIFICATION

After successful modeling of home automation alarms, APTE is executed to automatically generate alarms configuration (JSON), alarms server, android client and iOS client. The generated code artifacts for Home Automation case study can be found at [39]. This process is exactly similar as given in section V-B-3 for flour mill case study, therefore, we are not including such details here. After seamless generation and compilation of alarm server and mobile clients, the actual execution is performed. Particularly, the emulation of generated android client is performed to visualize, and control different



ExLow

AlarmRange (from AlarmsProfile::SharedProfile)
- value: String [1] = 30.0
- message: String [1] = Extremely low limit
- severityLevel: AlarmSeverity [1] = HIGH
- ackRequired: Bool [1] = false

severityLevel: HIGH
NORMAL
LOW
MEDIUM
HIGH
CRITICAL
CATASTROPHIC

GeyserTemperature

LimitAlarm (from AlarmsProfile::SharedProfile)
- setPoint: Double [1] = 70.0
- deadBand: Double [1] = 65.0
- alarmTitle: String [1] = Geyser Temperature Alarm
- alarmType: AlarmType [1] = LIMIT
- alarmCurrentValue: String [1] = 50
- isShelved: Bool [1] = true
- isSupressed: Bool [1] = false
- alarmDescription: String [1] = To observe the current Geys

alarmDescription: To observe the current Geyser Temperature

tagAddress

Destination (from AlarmsProfile::AlarmServerProfile)
- location: LocationType [1] = Holding_Registers

Device (from AlarmsProfile::AlarmServerProfile)
- deviceID: Int [1] = 33
- deviceName: String [1] = Geyser Temperature Device
- ipAddress: String [1] = 127.0.0.33
- port: Int [1] = 504

Input (from AlarmsProfile::AlarmServerProfile)
- startingAddress: Int [1] = 186

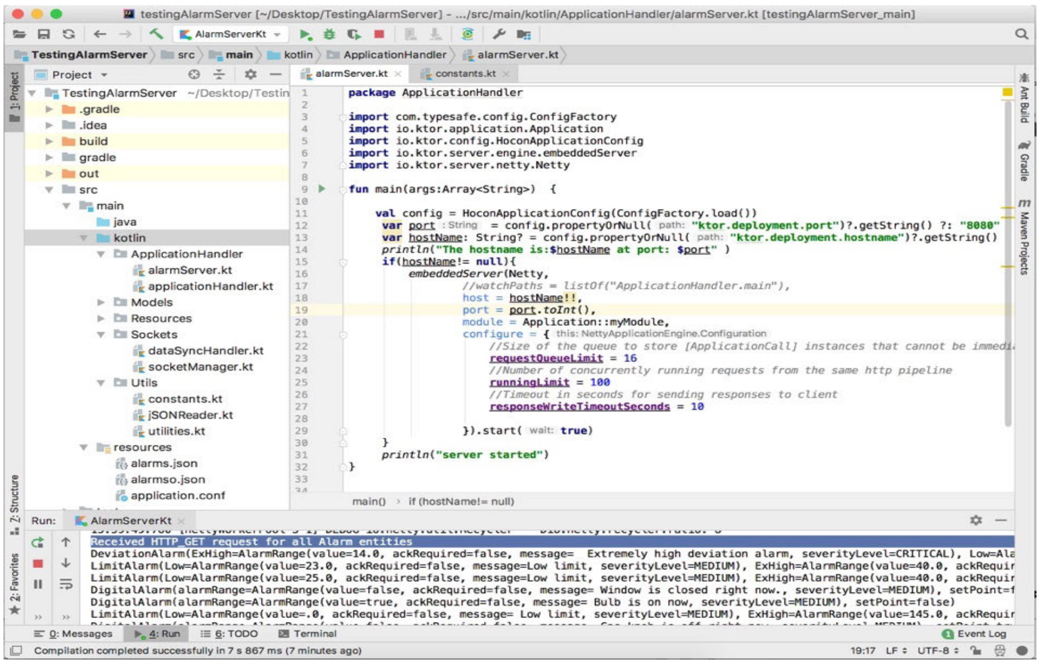
location: Holding_Registers
Coils
Holding_Registers
Input_Registers
Inputs

Tag15

PLC_Tag (from AlarmsProfile::AlarmServerProfile)
- tagId: String [1] = tagForGeyserTemperatureAlarm
- tagName: String [1] = Tag For Geyser Temperature Alarm
- tagValue: String [1] = 0.0

tagId: tagForGeyserTemperatureAlarm

FIGURE 14. **Assigning values to the instances of alarms in flour mill case study.**

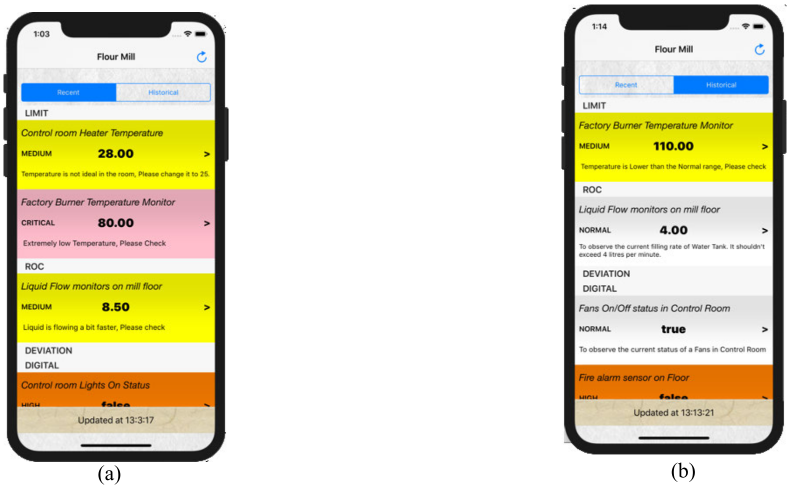


```
package ApplicationHandler

import com.typesafe.config.ConfigFactory
import io.ktor.application.Application
import io.ktor.config.HoconApplicationConfig
import io.ktor.server.engine.embeddedServer
import io.ktor.server.netty.Netty

fun main(args:Array<String>)  {

    val config = HoconApplicationConfig(ConfigFactory.load())
     var port :String  = config.propertyOrNull( path: "ktor.deployment.port")?.getString() ?: "8080"
     var hostName: String? = config.propertyOrNull( path: "ktor.deployment.hostname")?.getString()
     println("The hostname is:$hostName at port: $port" )
     if(hostName!= null){
         embeddedServer(Netty,
                 //watchPaths = listOf("ApplicationHandler.main"),
                 host = hostName!!,
                 port = port.toInt(),
                 module = Application::myModule,
                 configure = { this: NettyApplicationEngine.Configuration
                     //Size of the queue to store [ApplicationCall] instances that cannot be immedi
                     requestQueueLimit = 16
                     //Number of concurrently running requests from the same http pipeline
                     runningLimit = 100
                     //Timeout in seconds for sending responses to client
                     responseWriteTimeoutSeconds = 10

                 }).start( wait: true)
     }
     println("server started")
}
```

```
Received HTTP_GET request for all Alarm entities
DeviationAlarm(ExHigh=AlarmRange(value=14.0, ackRequired=false, message=  Extremely high deviation alarm, severityLevel=CRITICAL), Low=Ala
LimitAlarm(Low=AlarmRange(value=23.0, ackRequired=false, message=Low limit, severityLevel=MEDIUM), ExHigh=AlarmRange(value=40.0, ackRequir
LimitAlarm(Low=AlarmRange(value=25.0, ackRequired=false, message=Low limit, severityLevel=MEDIUM), ExHigh=AlarmRange(value=40.0, ackRequir
DigitalAlarm(alarmRange=AlarmRange(value=false, ackRequired=false, message= Window is closed right now., severityLevel=MEDIUM), setPoint=f
DigitalAlarm(alarmRange=AlarmRange(value=true, ackRequired=false, message= Bulb is on now, severityLevel=MEDIUM), setPoint=false)
LimitAlarm(Low=AlarmRange(value=.0, ackRequired=false, message= Low limit, severityLevel=MEDIUM), ExHigh=AlarmRange(value=145.0, ackRequir
```

```
Compilation completed successfully in 7 s 867 ms (7 minutes ago)
```

FIGURE 15. Deployment of automatically generated alarm server code for Flour Mill case study.



LIMIT
Control room Heater Temperature
MEDIUM 28.00 >
Temperature is not ideal in the room, Please change it to 25.
Factory Burner Temperature Monitor
CRITICAL 80.00 >
Extremely low Temperature, Please Check
ROC
Liquid Flow monitors on mill floor
MEDIUM 8.50 >
Liquid is flowing a bit faster, Please check
DEVIATION
DIGITAL
Control room Lights On Status
HIGH false >
Updated at 13:3:17

LIMIT
Factory Burner Temperature Monitor
MEDIUM 110.00 >
Temperature is Lower than the Normal range, Please check
ROC
Liquid Flow monitors on mill floor
NORMAL 4.00 >
To observe the current filling rate of Water Tank. It shouldn't exceed 4 litres per minute.
DEVIATION
DIGITAL
Fans On/Off status in Control Room
NORMAL true >
To observe the current status of a Fans in Control Room
Fire alarm sensor on Floor
HIGH false >
Updated at 13:13:21

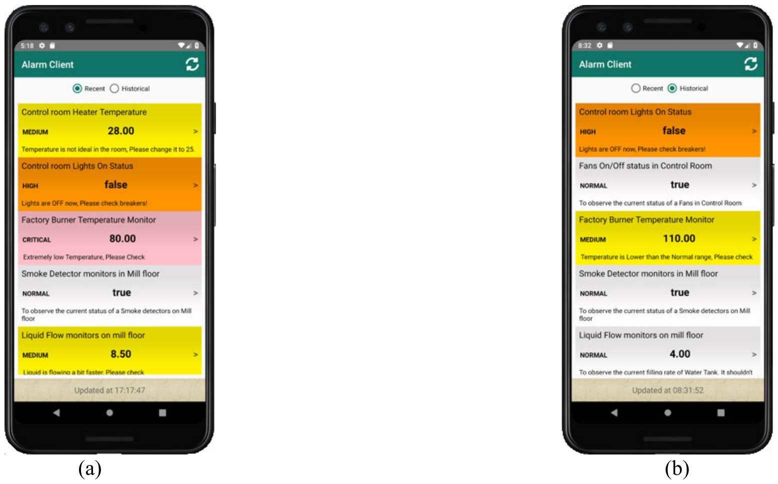
FIGURE 16. a. iOS client for flour mill (Realtime view). b. iOS client for flour mill (Historical view).

home automation alarms as shown in **Figure 19a and 19b**. Similarly, the generated iOS client is simulated to display and monitor the home automation alarms as shown in **Figure 20a** and **Figure 20b.** The source code of APTE and APICS models of Home Automation alarms can be found for further evaluation at [39].

VI. DISCUSSOIN AND LIMITATIONS

This article introduces an open source Model-driven Alarms Framework (MAF) where Alarm Profile for Industrial Control Systems (APICS) is proposed. APICS is based on standard UML profiling mechanism to enable the modeling of server, mobile clients and configuration requirements of alarms with simplicity. In this context, it can be argued that APICS doesn't support drag / drop functionality for the modeling of alarms requirements, as supported in existing industrial software like SIMATIC WinCC. As a result, the modeling in APICS is relatively complex as compared to existing industrial automation software. Actually, the UML is a well-known modeling standard and its profiling concepts like stereotypes etc. are commonly utilized. Therefore, modeling in APICS is fairly simple. Furthermore, the existing APICS concepts can easily be imported to Sirius framework in order to accomplish the modeling with drag / drop functionality [33].

In Section V, the applicability of MAF is demonstrated where real PLCs are utilized for data acquisition, however, actual sensors are not attached with PLCs. For this reason, it can be argued that the applicability of MAF in real industries can be questionable. Actually, MAF is purely developed


(a)

5:18

Alarm Client

(● Recent) (○ Historical)

Control room Heater Temperature
MEDIUM 28.00 >
Temperature is not ideal in the room, Please change it to 25.

Control room Lights On Status
HIGH false >
Lights are OFF now, Please check breakers!

Factory Burner Temperature Monitor
CRITICAL 80.00 >
Extremely low Temperature, Please Check

Smoke Detector monitors in Mill floor
NORMAL true >
To observe the current status of a Smoke detectors on Mill floor

Liquid Flow monitors on mill floor
MEDIUM 8.50 >
Liquid is flowing a bit faster. Please check

Updated at 17:17:47

(b)

8:32

Alarm Client

(○ Recent) (● Historical)

Control room Lights On Status
HIGH false >
Lights are OFF now, Please check breakers!

Fans On/Off status in Control Room
NORMAL true >
To observe the current status of a Fans in Control Room

Factory Burner Temperature Monitor
MEDIUM 110.00 >
Temperature is Lower than the Normal range, Please check

Smoke Detector monitors in Mill floor
NORMAL true >
To observe the current status of a Smoke detectors on Mill floor

Liquid Flow monitors on mill floor
NORMAL 4.00 >
To observe the current filling rate of Water Tank. It shouldn't

Updated at 08:31:52

FIGURE 17. a. Android client for flour mill (Realtime view). b. Android Client for Flour Mill (Historical View).

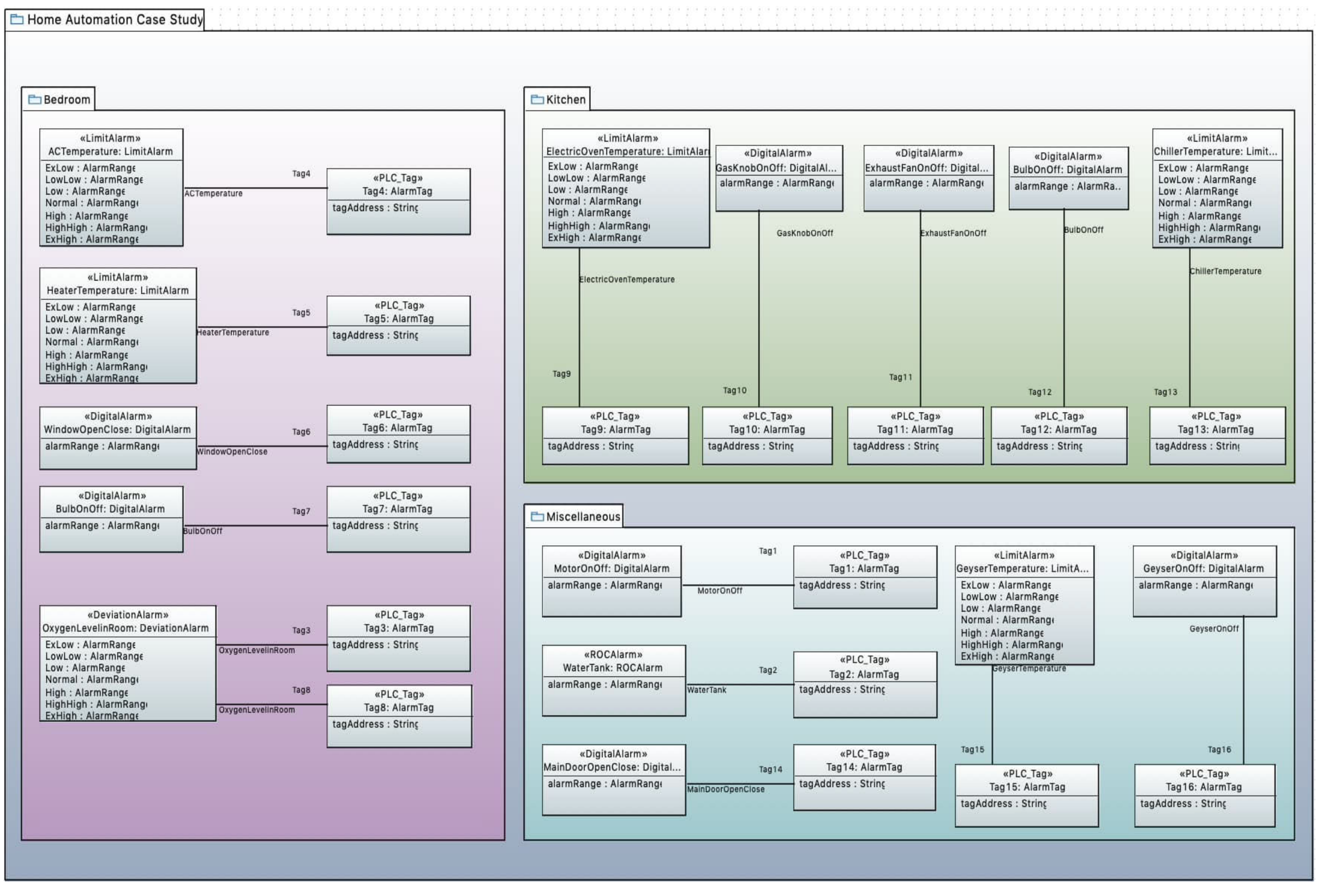
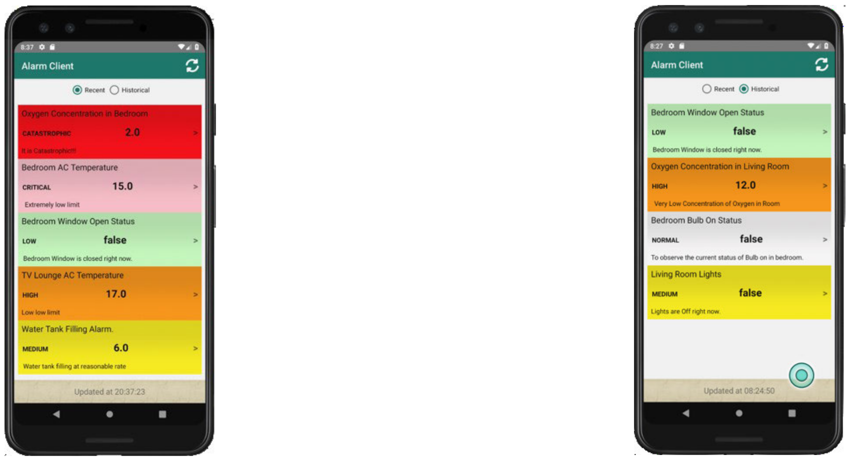

Home Automation Case Study
Bedroom
«LimitAlarm» ACTemperature: LimitAlarm
ExLow : AlarmRange
LowLow : AlarmRange
Low : AlarmRange
Normal : AlarmRange
High : AlarmRange
HighHigh : AlarmRange
ExHigh : AlarmRange
ACTemperature
Tag4
«PLC_Tag» Tag4: AlarmTag
tagAddress : String
«LimitAlarm» HeaterTemperature: LimitAlarm
HeaterTemperature
Tag5
«PLC_Tag» Tag5: AlarmTag
«DigitalAlarm» WindowOpenClose: DigitalAlarm
alarmRange : AlarmRange
WindowOpenClose
Tag6
«PLC_Tag» Tag6: AlarmTag
«DigitalAlarm» BulbOnOff: DigitalAlarm
BulbOnOff
Tag7
«PLC_Tag» Tag7: AlarmTag
«DeviationAlarm» OxygenLevelinRoom: DeviationAlarm
OxygenLevelinRoom
Tag3
«PLC_Tag» Tag3: AlarmTag
Tag8
«PLC_Tag» Tag8: AlarmTag
Kitchen
«LimitAlarm» ElectricOvenTemperature: LimitAlar
«DigitalAlarm» GasKnobOnOff: DigitalAl...
«DigitalAlarm» ExhaustFanOnOff: Digital...
«DigitalAlarm» BulbOnOff: DigitalAlarm
alarmRange : AlarmRa..
«LimitAlarm» ChillerTemperature: Limit...
ElectricOvenTemperature
GasKnobOnOff
ExhaustFanOnOff
BulbOnOff
ChillerTemperature
Tag9
Tag10
Tag11
Tag12
Tag13
«PLC_Tag» Tag9: AlarmTag
«PLC_Tag» Tag10: AlarmTag
«PLC_Tag» Tag11: AlarmTag
«PLC_Tag» Tag12: AlarmTag
«PLC_Tag» Tag13: AlarmTag
Miscellaneous
«DigitalAlarm» MotorOnOff: DigitalAlarm
MotorOnOff
Tag1
«PLC_Tag» Tag1: AlarmTag
«ROCAlarm» WaterTank: ROCAlarm
WaterTank
Tag2
«PLC_Tag» Tag2: AlarmTag
«DigitalAlarm» MainDoorOpenClose: Digital...
MainDoorOpenClose
Tag14
«PLC_Tag» Tag14: AlarmTag
«LimitAlarm» GeyserTemperature: LimitA...
GeyserTemperature
Tag15
«PLC_Tag» Tag15: AlarmTag
«DigitalAlarm» GeyserOnOff: DigitalAlarm
GeyserOnOff
Tag16
«PLC_Tag» Tag16: AlarmTag

FIGURE 18. Instance model containing alarms and devices information for home automation.

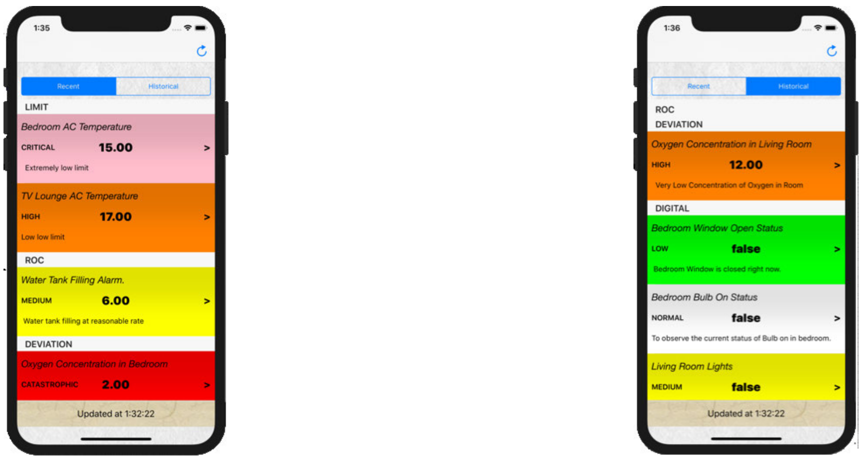
in academia without any funding. Therefore, we have very limited resources and it is not possible for us to attach all real sensors with PLCs. However, we utilize real PLCs registers to fetch the tags values through DA server for the processing of alarms. Moreover, we utilize Modbus protocol for data acquisition where all four types of registers (holding, input, discrete and coils) are utilized. Furthermore, we frequently alter the values of PLCs tags through DA server (write operation) to verify the correct response of alarm server and clients accordingly. Consequently, on the basis of aforementioned


Alarm Client
Recent Historical
Oxygen Concentration in Bedroom
CATASTROPHIC 2.0
It is Catastrophic!!!
Bedroom AC Temperature
CRITICAL 15.0
Extremely low limit
Bedroom Window Open Status
LOW false
Bedroom Window is closed right now.
TV Lounge AC Temperature
HIGH 17.0
Low low limit
Water Tank Filling Alarm.
MEDIUM 6.0
Water tank filling at reasonable rate
Updated at 20:37:23

Alarm Client
Recent Historical
Bedroom Window Open Status
LOW false
Bedroom Window is closed right now.
Oxygen Concentration in Living Room
HIGH 12.0
Very Low Concentration of Oxygen in Room
Bedroom Bulb On Status
NORMAL false
To observe the current status of Bulb on in bedroom.
Living Room Lights
MEDIUM false
Lights are Off right now.
Updated at 08:24:50

(a) (b)

FIGURE 19. a. Android client for home automation (Realtime view). b. Android client for home automation (Historical view).


Recent Historical
LIMIT
Bedroom AC Temperature
CRITICAL 15.00
Extremely low limit
TV Lounge AC Temperature
HIGH 17.00
Low low limit
ROC
Water Tank Filling Alarm.
MEDIUM 6.00
Water tank filling at reasonable rate
DEVIATION
Oxygen Concentration in Bedroom
CATASTROPHIC 2.00
Updated at 1:32:22

Recent Historical
ROC
DEVIATION
Oxygen Concentration in Living Room
HIGH 12.00
Very Low Concentration of Oxygen in Room
DIGITAL
Bedroom Window Open Status
LOW false
Bedroom Window is closed right now.
Bedroom Bulb On Status
NORMAL false
To observe the current status of Bulb on in bedroom.
Living Room Lights
MEDIUM false
Updated at 1:32:22

(a) (b)

FIGURE 20. a. iOS client for home automation (Realtime view). b. iOS client for home automation (Historical view).

facts, it can be established that MAF is widely applicable in real industries. In this regard, we are currently coordinating with different industries to validate the MAF in actual industrial processes.

In large industrial processes, hundreds of alarms are processed and visualized at a time. For such situations, we have analyzed the capabilities of MAF. Particularly, we have utilized 400 tags and processed 300 alarms through MAF. It is observed that alarms server successfully received the values of 400 tags from DA server and processed the configured alarms under 500 milliseconds time span. On the other hand, alarm clients (i.e. Android and iOS) successfully displayed / updated the generated alarms from alarm server under 500 milliseconds timespan without any lag. This further proves the applicability of MAF in large and complex industrial processes.

The information of historical alarms is really important while dealing with the large number of alarms in short time. MAF clients not only support the visualization of real time alarms but also provide the functionality of historical alarms. Particularly, real time alarms received from the alarm server are temporarily stored in alarm clients (i.e. Android and iOS) with important alarm attributes like timestamp, alarm severity, acknowledgment and comments (if any) etc. This historical information of received alarms is displayed by the alarm clients under "historical" tab. Once the storage of alarm clients exceeds 1 MB, the older alarms are replaced with newer alarms.

To summarize, MAF is a first open source framework that simultaneously support the modeling and transformation of alarm server and mobile clients (i.e. Android and iOS). Furthermore, it also supports all major types of alarms i.e. *Limit, Deviation, Rate-of-Change (RoC)* and *Digital Alarms*. Consequently, MAF is able to manage the modern and complex alarms requirements for wide-ranging industrial control systems. The framework like MAF is hard to find in the literature (Section II-A). On the other hand, MAF provides several advantages over existing industry automation software's (e.g. SIMATIC WinCC, Genesis64 etc.) as follows: 1) MAF is open source and freely available while industrial software

are proprietary i.e. pay per tag policy. Consequently, MAF is a really cost-effective solution with mobile client's support which is an essential requirement of modern industrial processes. 2) The underlying development techniques and tools of MAF are clearly given in this article. Therefore, industry and academia can customize MAF as per requirements. For example, polling rate can be altered in alarm server as per particular data acquisition requirements. Furthermore, MAF can be extended to support latest OPC UA concepts like subscription-based data acquisition etc. To summarize, MAF provides a solid platform for new researchers and practitioners for further enhancements. On the other hand, industrial software do not publicize their underlying development procedures, therefore, the customization / extension is not possible.

VII. CONCLUSION AND FUTURE WORK

In this article, a Model-driven Alarms Framework (**MAF**) with mobile clients support is introduced. Particularly, an Alarm Profile for Industrial Control Systems (**APICS**) has been developed by extending the standard Unified Modeling Language (UML) concepts. It enables the modeling of server, mobile clients and configuration requirements of alarms altogether with simplicity. As a part of MAF, a fully open source Alarms Profile Transformation Engine (**APTE**) is implemented. APTE automatically generates: 1) Kotlin code for alarm server, 2) native Android and iOS code for alarm clients in Kotlin and Swift respectively, from APICS-compliant models. Consequently, the generated alarm server and clients (i.e. Android and iOS) can be instantly deployed for the real time monitoring of industrial processes. The feasibility of MAF is demonstrated through *flour mill* and *home automation* case studies. The results prove that MAF provides a complete alarms solution where alarm server is able to generate several alarms simultaneously and mobile alarm clients provide real time visualization of multiple alarms. Therefore, it can be concluded that *MAF is a simple and cost-effective alarms solution for wide-ranging industries*.

Being an *open source solution*, MAF provides a solid platform for further extensions to meet the growing requirements of alarms in industry automation. For example, MAF can be extended to automatically take corrective measures against particular alarms by employing the *artificial intelligence / machine learning approaches*. Another possibility is to incorporate the *multimedia alarms features* in MAF where generated alarms can be automatically notified to concerned authorities through different mediums like email, phone call etc. Such extensions of MAF are open for researchers and practitioners of the domain.

REFERENCES

[1] J. Xiong, J. Li, J. Shi, and Y. Huang, "A decomposition-based development method for industrial control systems," *IEEE Access*, vol. 7, pp. 93161–93174, 2019, doi: 10.1109/ACCESS.2019.2927263.

[2] A. W. Al-Dabbagh and T. Chen, "Sounding off on industrial alarm systems," *IEEE Potentials*, vol. 37, no. 2, pp. 24–28, Mar. 2018, doi: 10.1109/MPOT.2016.2582219.

[3] D. H. Rothenberg, "Meet alarm management," in *Alarm Management for Process Control*, vol. 8. New York, NY, USA: Momentum Press, 2018, ch. 1, sec. 1, pp. 13–15.

[4] P. Goel, A. Datta, and M. S. Mannan, "Industrial alarm systems: Challenges and opportunities," *J. Loss Prevention Process Industries*, vol. 50, pp. 23–36, Nov. 2017, doi: 10.1016/j.jlp.2017.09.001.

[5] Z. Aslam, W. Khalid, T. Ahmed, and D. Marghoob, "Automated control system for indoor air quality management," in *Proc. Int. Conf. Energy Conservation Efficiency (ICECE)*, Lahore, Pakistan, Nov. 2017, pp. 85–88, doi: 10.1109/ECE.2017.8248834.

[6] S. Ardi, A. Ponco, and R. A. Latief, "Design of integrated SCADA systems in piston production manufacturing case study on the conveyor, the coolant, the hydraulic, and the alarm systems using PLC CJ1M and CJ1W-ETN21," in *Proc. 4th Int. Conf. Inf. Technol., Comput., Electr. Eng. (ICITACEE)*, Semarang, Indonesia, Oct. 2017, pp. 187–191, doi: 10.1109/ICITACEE.2017.8257700.

[7] D. Babunski, E. Zaev, A. Tuneski, and D. Bozovic, "Optimization methods for water supply SCADA system," in *Proc. 7th Medit. Conf. Embedded Comput. (MECO)*, Budva, Montenegro, Jun. 2018, pp. 1–4, doi: 10.1109/MECO.2018.8405970.

[8] J. Saha, A. K. Saha, A. Chatterjee, S. Agrawal, A. Saha, A. Kar, and H. N. Saha, "Advanced IOT based combined remote health monitoring, home automation and alarm system," in *Proc. IEEE 8th Annu. Comput. Commun. Workshop Conf. (CCWC)*, Las Vegas, NV, USA, Jan. 2018, pp. 602–606, doi: 10.1109/CCWC.2018.8301659.

[9] M. Cañizares, "Design and implementation of the alarms control system on a vital signs monitor," in *VI Latin American Congress on Biomedical Engineering CLAIB 2014, Paraná*, vol. 49. Cham, Switzerland: Springer, doi: 10.1007/978-3-319-13117-7_177.

[10] S. A. Paul, A. K. Carroll, and S. M. Treacy, "Designing the alarm management user experience for patient monitoring," in *Proc. CHI Conf. Extended Abstr. Hum. Factors Comput. Syst. (CHI EA)*, New York, NY, USA, 2016, pp. 710–717, doi: 10.1145/2851581.2851604.

[11] J. Ni, W. Zhu, J. Huang, L. Niu, and L. Wang, "Fall guard: Fall monitoring application for the elderly based on Android platform," in *Proc. 4th Int. Conf. Biomed. Signal Image Process. (ICBIP)*, New York, NY, USA, 2019, pp. 128–135, doi: 10.1145/3354031.3354055.

[12] I. Qasim, M. W. Anwar, F. Azam, H. Tufail, W. H. Butt, and M. N. Zafar, "A model-driven mobile HMI framework (MMHF) for industrial control systems," *IEEE Access*, vol. 8, pp. 10827–10846, 2020, doi: 10.1109/ACCESS.2020.2965259.

[13] M. He, L. Feng, and J. Qu, "Alarm information collection method in power environment supervision system of railway based on TCP/IP protocol research," in *Proc. 3rd Int. Conf. Commun. Inf. Process. (ICCIP)*, New York, NY, USA, vol. 2018, pp. 113–118, doi: 10.1145/3162957.316295.

[14] M. W. Anwar, M. Rashid, F. Azam, A. Naeem, M. Kashif, and W. H. Butt, "A unified model-based framework for the simplified execution of static and dynamic assertion-based verification," *IEEE Access*, vol. 8, pp. 104407–104431, 2020, doi: 10.1109/ACCESS.2020.2999544.

[15] P. Urban and L. Landryova, "Process knowledge building an optimized alarm system," in *Proc. 16th Int. Carpathian Control Conf. (ICCC)*, Szilvasvarad, Hungary, May 2015, pp. 563–566, doi: 10.1109/CarpathianCC.2015.7145143.

[16] P. Urban and L. Landryova, "Identification and evaluation of alarm logs from the alarm management system," in *Proc. 17th Int. Carpathian Control Conf. (ICCC)*, Tatranska Lomnica, Slovakia, May 2016, pp. 769–774, doi: 10.1109/CarpathianCC.2016.7501199.

[17] M. Bockelmann, P. Nickel, and F. Nachreiner, "Development of an online checklist for the assessment of alarm systems and alarm management in process control," in *Proc. Int. Conf. HCI Bus., Government, Organizations (HCIBGO)*, Vancouver, BC, Canada, vol. 10294. Cham, Switzerland: Springer, 2017, pp. 325–332, doi: 10.1007/978-3-319-58484-3_25.

[18] M. Bockelmann, P. Nickel, and F. Nachreiner, "The design of alarm systems and alarm management - an empirical investigation from an ergonomic perspective," in *Proc. Int. Conf. Appl. Hum. Factors Ergonom.*, Los Angeles, CA, USA, vol. 604. Cham, Switzerland: Springer, 2017, pp. 507–515, doi: 10.1007/978-3-319-60525-8_52.

[19] M. Bockelmann, P. Nickel, and F. Nachreiner, "Ergonomics analysis of alarm systems and alarm management in process industries," in *Proc. 20th Congr. Int. Ergonom. Assoc. (IEA)*, Florence, Italy, vol. 822. Cham, Switzerland: Springer, 2018, pp. 727–732, doi: 10.1007/978-3-319-96077-7_79.

[20] J. Zhu, C. Wang, C. Li, X. Gao, and J. Zhao, "Dynamic alarm prediction for critical alarms using a probabilistic model," *Chin. J. Chem. Eng.*, vol. 24, no. 7, pp. 881–885, Jul. 2016, doi: 10.1016/j.cjche.2016.04.017.

[21] V. B. Soares, J. C. Pinto, and M. B. de Souza, "Alarm management practices in natural gas processing plants," *Control Eng. Pract.*, vol. 55, pp. 185–196, Oct. 2016, doi: 10.1016/j.conengprac.2016.07.004.

[22] J. Wang, H. Li, J. Huang, and C. Su, "A data similarity based analysis to consequential alarms of industrial processes," *J. Loss Prevention Process Industries*, vol. 35, pp. 29–34, May 2015, doi: 10.1016/j.jlp.2015.03.005.

[23] Y. Cui, P. Bangalore, and L. B. Tjernberg, "An anomaly detection approach based on machine learning and SCADA data for condition monitoring of wind turbines," in *Proc. IEEE Int. Conf. Probabilistic Methods Appl. Power Syst. (PMAPS)*, Boise, ID, USA, Jun. 2018, pp. 1–6, doi: 10.1109/PMAPS.2018.8440525.

[24] A. Kanawaday and A. Sane, "Machine learning for predictive maintenance of industrial machines using IoT sensor data," in *Proc. 8th IEEE Int. Conf. Softw. Eng. Service Sci. (ICSESS)*, Beijing, China, Nov. 2017, pp. 87–90, doi: 10.1109/ICSESS.2017.8342870.

[25] A. E. Kashef and N. Barakat, "Intelligent alarm system to protect small, valuable items," in *Proc. Int. Conf. Comput. Appl. (ICCA)*, Beirut, Lebanon, Aug. 2018, pp. 326–330, doi: 10.1109/COMAPP.2018.8460470.

[26] G. A. Francia, "A machine learning test data set for continuous security monitoring of industrial control systems," in *Proc. IEEE 7th Annu. Int. Conf. CYBER Technol. Autom., Control, Intell. Syst. (CYBER)*, Honolulu, HI, USA, Jul. 2017, pp. 1043–1048, doi: 10.1109/CYBER.2017.8446474.

[27] J. Wang, Y. Zhao, and Z. Bi, "Criteria and algorithms for online and offline detections of industrial alarm floods," *IEEE Trans. Control Syst. Technol.*, vol. 26, no. 5, pp. 1722–1731, Sep. 2018, doi: 10.1109/TCST.2017.2723578.

[28] A. W. Al-Dabbagh, W. Hu, S. Lai, T. Chen, and S. L. Shah, "Toward the advancement of decision support tools for industrial facilities: Addressing operation metrics, visualization plots, and alarm floods," *IEEE Trans. Autom. Sci. Eng.*, vol. 15, no. 4, pp. 1883–1896, Oct. 2018, doi: 10.1109/TASE.2018.2827309.

[29] H. Huawei, W. Chunli, S. Ning, and J. Weiwei, "Design of intelligent alarm system for petrochemical process," in *Proc. Chin. Control Decis. Conf. (CCDC)*, Shenyang, China, Jun. 2018, pp. 5059–5064, doi: 10.1109/CCDC.2018.8408008.

[30] M. W. Anwar and F. Azam, "Proposing a novel architecture of script component to incorporate the scripting language support in SCADA systems," in *Proc. IFIP Int. Conf. Comput. Inf. Syst. Ind. Manage. (CISIM)*, vol. 8838, 2015, pp. 351–362, doi: 10.1007/978-3-662-45237-0_33.

[31] J. Sompura, A. Joshi, B. Srinivasan, and R. Srinivasan, "A practical approach to improve alarm system performance: Application to power plant," *Chin. J. Chem. Eng.*, vol. 27, no. 5, pp. 1094–1102, May 2019, doi: 10.1016/j.cjche.2018.09.020.

[32] *Xcode IDE and iOS Simulator*. Accessed: Mar. 2020. [Online]. Available: https://developer.apple.com/xcode/

[33] *Eclipse Sirius Platform*. Accessed: Mar. 2020. [Online]. Available: https://www.eclipse.org/sirius/

[34] J. Taheri-Kalani, G. Latif-Shabgahi, and M. Alyari Shooredeli, "On the use of penalty approach for design and analysis of univariate alarm systems," *J. Process Control*, vol. 69, pp. 103–113, Sep. 2018, doi: 10.1016/j.jprocont.2018.07.018.

[35] X. Xu, X. Weng, D. Xu, H. Xu, Y. Hu, and J. Li, "Evidence updating with static and dynamical performance analyses for industrial alarm system design," *ISA Trans.*, vol. 99, pp. 110–122, Apr. 2020, doi: 10.1016/j.isatra.2019.09.006.

[36] R. Kaced, A. Kouadri, and K. Baiche, "Designing alarm system using modified generalized delay-timer," *J. Loss Prevention Process Industries*, vol. 61, pp. 40–48, Sep. 2019, doi: 10.1016/j.jlp.2019.04.010.

[37] H. Tufail, M. Waseem Anwar, I. Qasim, and F. Azam, "Towards the selection of optimum alarms system in leading industry automation software," in *Proc. 8th Int. Conf. Ind. Technol. Manage. (ICITM)*, Cambridge, U.K., Mar. 2019, pp. 241–246, doi: 10.1109/ICITM.2019.8710731.

[38] *Fatek PLC*. Accessed: Jan. 2020. [Online]. Available: http://www.fatek.com/en/prod.php?catId=1

[39] *Model-based Alarms Framework (MAF) for Industry Control Systems*. Accessed: Aug. 2020. [Online]. Available: https://github.com/MDSE-Research-Group

[40] *Eclipse Oxygen*. Accessed: Jan. 2020. [Online]. Available: https://www.eclipse.org/oxygen/

[41] *Papyrus (UML & Profile) Modeling Tool*. Accessed: Jan. 2020. [Online]. Available: https://www.eclipse.org/papyrus/

[42] *Acceleo–Model-to-Text Transformation Tool*. Accessed: Jan. 2020. [Online]. Available: https://www.eclipse.org/acceleo/

[43] *IntelliJ IDEA Community Edition*. Accessed: Jan. 2020. [Online]. Available: https://www.jetbrains.com/idea/download/#section=mac

[44] *Android Studio*. Accessed: Feb. 2020. [Online]. Available: https://developer.android.com/studio

[45] *Android Emulator*. Accessed: Feb. 2020. [Online]. Available: https://developer.android.com/studio/run/managing-avds

[46] A. Amjad, F. Azam, M. W. Anwar, W. H. Butt, M. Rashid, and A. Naeem, "UMLPACE for modeling and verification of complex business requirements in event-driven process chain (EPC)," *IEEE Access*, vol. 6, pp. 76198–76216, 2018, doi: 10.1109/ACCESS.2018.2883610.

[47] M. W. Anwar, M. Rashid, F. Azam, M. Kashif, and W. H. Butt, "A model-driven framework for design and verification of embedded systems through SystemVerilog," *Design Autom. Embedded Syst.*, vol. 23, nos. 3–4, pp. 179–223, Dec. 2019, doi: 10.1007/s10617-019-09229-y.

[48] H. Tufail, F. Azam, M. W. Anwar, and I. Qasim, "Model-driven development of mobile applications: A systematic literature review," in *Proc. IEEE 9th Annu. Inf. Technol., Electron. Mobile Commun. Conf. (IEMCON)*, Vancouver, BC, Canada, Nov. 2018, pp. 1165–1171, doi: 10.1109/IEMCON.2018.8614821.

[49] Z. Wang, X. Bai, J. Wang, and Z. Yang, "Indexing and designing deadbands for industrial alarm signals," *IEEE Trans. Ind. Electron.*, vol. 66, no. 10, pp. 8093–8103, Oct. 2019, doi: 10.1109/TIE.2018.2885718.

HANNY TUFAIL received the master's degree in software engineering from the National University of Sciences and Technology (NUST), Pakistan. He has more than eight year's industrial experience. He is also an active member of the Model-Driven Software Engineering Research Group at CEME, NUST. His research interests are model-driven software engineering, mobile applications, and industrial control systems.

FAROOQUE AZAM is currently a Key Faculty Member of the Department of Computer and Software Engineering, EME College, NUST, Pakistan. He has been involved in postgraduate teaching and research since 2007. His areas of research interests are model-driven software engineering, model-driven testing, model-driven embedded applications, model-driven web engineering, as well as software design and architectures. Until April 2020, he has 138 international journal and conference publications on his credit. He is a regular member of the evaluation committees of the Pakistan Engineering Council (PEC) and the Higher Education Commission's Technological Development Funding (HEC-TDF).

MUHAMMAD WASEEM ANWAR is currently pursuing the Ph.D. degree with the Department of Computer and Software Engineering, CEME, National University of Sciences and Technology, Pakistan. He is also a Senior Researcher and an Industry Practitioner in the field of Model-Based System Engineering (MBSE) for embedded and control systems. His major research area is Model-Based System Engineering (MBSE) for complex and large systems.

MUHAMMAD NOUMAN ZAFAR received the M.S. degree in software engineering from the College of Electrical and Mechanical Engineering (CEME), National University of Sciences and Technology (NUST), Islamabad, in 2018. He is currently pursuing the Ph.D. degree with Malardalen University, Sweden. His research interests are model-driven architecture, distributed industrial control systems, the Internet of Things, big data, and robotics. He is also a member of the Model-Driven Software Engineering Research Group, CEME, NUST.

ABDUL WAHAB MUZAFFAR received the Ph.D. degree in software engineering from the National University of Sciences and Technology (NUST) Islamabad, Pakistan, in 2017. He is currently an Assistant Professor with the Saudi Electronic University, Saudi Arabia. His research interests include model-driven software engineering, data and text mining, bioinformatics, and machine learning.

WASI HAIDER BUTT is currently an Assistant Professor with the Department of Computer and Software Engineering, College of Electrical and Mechanical Engineering, National University of Sciences and Technology, Pakistan. His areas of interests are model-driven software engineering, Web development, and requirement engineering.