Component-Oriented Software Engineering Model for Heterogeneous Internet of Things Systems with Connectors using Machine Learning

Shahanawaj Ahamad

drshahwj@gmail.com
Department of Information and Computer Science
College of Computer Science and Engineering
University of Hail, Hail, Saudi Arabia

Summary
Component reuse has been proven both theoretically and empirically to increase software quality and productivity with an economically cost-effective option. This necessitates the use of a graphical editor for project modeling using component-based architecture and development. To aid in the creation of component-oriented software, a graphical editor was proposed for practice. Both machine learning and software engineering employ models based on components architecture. Aside from these smart characteristics, AI models may be able to help with prediction and decision-making. Communication between IoT system components must adhere to a set of guidelines and protocols for effective and predictive perspectives. Components must be able to communicate with one another in the deployed system. The heterogeneity issue in the Internet of Things arises when different IoT devices communicate using distinct sets of rules, features, and contexts. Components that can be reused are found in these or other systems or commercial off-the-shelf. Component-oriented systems rely on connectors to link up their reusable parts with other entities, components, or IoT devices through the use of related interfaces. COSE development tools provide application-level solutions for connectors and component-based development of systems. Linking and hookup ports on connectors are designed to work with the attached component and other interfaces. The communication protocols' packets are identified and organized by the connectors with their installed applications. A simulation feature can be added to the tools in order to show that the idea can be implemented in effective and efficient ways. Connectors allow moving data between different parts of computing systems. ML-based training and prediction have been shown in this work for performance analysis.

Keywords:
Component-Oriented Software Engineering, Heterogenous Internet of Things, Machine Learning, Software Reuse, COTS.

Manuscript received June 5, 2022
Manuscript revised June 20, 2022
https://doi.org/10.22937/IJCSNS.2022.22.6.86

1. Introduction

Historically, software was created using structured programming languages. The structured programming approach becomes ineffective as the software develops in complexity. In those days, designing, developing, and maintaining complex software features using structured programming was extremely difficult. Object-oriented design and programming were formed over a decade ago to address these challenges [1]. Even now, this has shown to be a viable strategy for developing a complicated software system. Component-based Software Engineering includes Component-based Software Development as a subfield. Figure 1 depicted the progress from structured to component-based development.

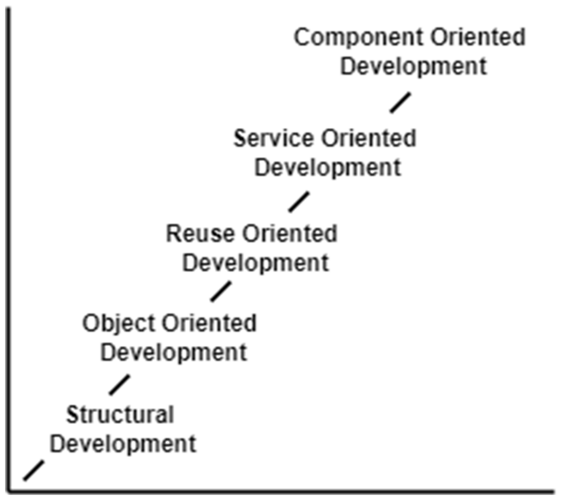
Component Oriented Development
Service Oriented Development
Reuse Oriented Development
Object Oriented Development
Structural Development

Fig. 1 Development chart based on software development.

Composing loosely linked individual components into systems using this method is a reuse-based strategy. The module, functionality, or web services can all be considered as components of the component. The ultimate system will be built by integrating all of the components. Different suppliers may sell these components because they are independent and evaluated independently components, much like the hardware, can be swapped out in the event of a failure [1,2]. The component-based software engineering must be applied correctly to avoid problems in the software architecture. Because the final system will be built using this design and all of its components will be integrated. The whole system's reliability will suffer if the architecture is flawed in any way. Fast software development and delivery to consumers can be achieved by using this method of working. Quality, on the other hand, is the most important consideration to be made before to the software's release [3]. In addition, a component-based program is easier to extend. Existing components do not need to be touched when new requirements are implemented because they can be provided in new components [4]. Component-based software development reduces long-term maintenance costs, which is a necessity for nearly every firm, which explains why component technologies have become so widely adopted.

In general, component-based software solutions have a shorter time to market since we can choose from a wide selection of available components. The ad hoc technique was once popular in the field of software engineering [23]. In the 1970s, the focus switched away from the previous method and toward a methodical approach. Subsequently, in the late 1980s, a software engineering object-oriented method was presented. With Component-based software engineering, a new approach to the software market was introduced recently. Figure 2 shows the general structure of component-based software engineering. The internal narrative expresses to us that all the systems are not new, but they are the unchanging parts of the current systems. This point is contributing to enhancing the performance and quality of the software development process. The software's complexity has increased as a result of the current circumstance. In order to meet these high criteria and handle the complexity, the component-based strategy might be implemented [5]. Rather than starting from scratch, it is more efficient to build a system by integrating components that already exist. Component-based development's effectiveness is one of its greatest assets. The use of predictable construction patterns and a standard software architecture is also encouraged by CBSE, leading to improved performance.

Software engineering includes systematic requirement studies and processes of software development concepts, principles, and further maintenance. The goal is to propose the processes and tools for software development that are more efficient and effective. In order to make software development (SD) more accessible and efficient, as well as increase the ability to produce sophisticated software and meet user requirements, research and study are always being conducted on this subject [6]. To help software engineers, advanced theories and concepts in the field of SD are being developed.

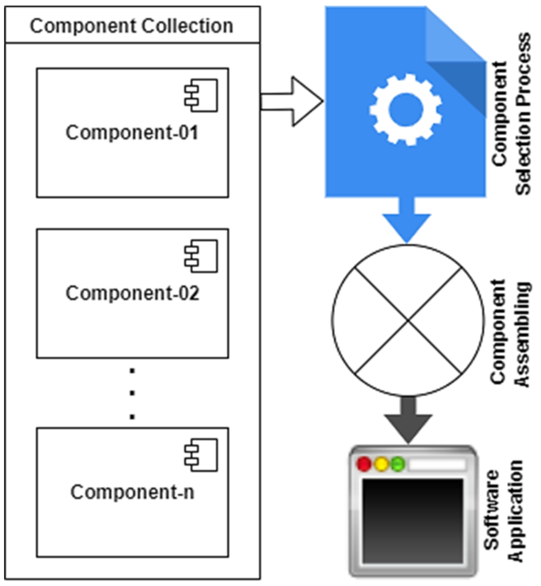

Fig. 2 General structure of component-based software engineering

The methodical and disciplined approach to software development and maintenance is known as "Software Engineering". The change from traditional software development to a systematic approach to software development was made possible by the concept of structured programming [7]. To bring software engineering to its current position, this structured programming idea has opened up specialized study areas during research, it was found that software projects must be managed well to avoid overruns in both time and money. In order to avoid delays and failures, a proactive activity is needed.

In order to identify the areas of the system that demand additional attention, an evaluation of the system's many components is required. High-level designs and the system's quality needs are both included in its architecture [8]. In this way, a thorough examination of the system's design can identify the most vulnerable parts of the system. The identification of vulnerable components can lead to a risk assessment and the development of risk mitigation strategies as a result of this identification. The quality of the system will improve as a result of the application of risk mitigation measures.

An applicable tool that uses a Component-Oriented methodology to break down system requirements in a hierarchical manner is the goal of the research. The

Component-Oriented Software Engineering Modeling Language (COSEML) previously developed by [9] can be used to model such system hierarchy. Component-based (CB) technology, in which only a limited number of development phases are dedicated to the consideration of components, is the current state of affairs. There has been a lack of a top-down approach that considers all components.

2. Literature Review

For instance, the Model-Driven Software Engineering (MDSE) standards use software models as a high-level abstraction by respecting model-to-model and model-to-code conversions. This allows the standards to be used to software development and simulation [10]. The foundation of DAML has been laid with its two pillars. To improve the degree of abstraction, libraries of higher-level APIs that also include associated frameworks are used. One other option for deep learning is Theano [8, 12], which is a framework that is compatible with both TensorFlow and Keras (https://keras.io) [24]. Examples of DAML workflow designers include visualization toolkits like TensorBoard [7] and KNIME [4]. The MDSE paradigm, in which model-to-code transformations may build software implementations from comprehensive models, is not followed by these methodologies. This is because the MDSE paradigm follows a holistic and logical approach. Workflows in KNIME [4], RapidMiner [14], and TensorBoard [17] that are referred to as Data-Flow Graphs (DFG) do not manage any aspect or issue that is not related to DAML. Some process designs, like KNIME [4,] create partial DAML code. There is a connection between MDSE and the various model-exchange formats, including PMML, PFA, and ONNX. The Data Mining Group is responsible for producing the XML-based standard [13], which has been used by around 30 firms throughout the globe. New to the DMG, PFA is a standard that is both more versatile and powerful than its predecessor, PMML. PFA offers a DSL that may be used for the implementation of any DAML method. Second, a process and DAML model might be modeled using PFA. The Internet of Things and machine learning-based architecture are both discussed in [14]. The ONNX platform makes it possible to develop ANN models with the help of TensorFlow libraries and frameworks such as Keras [1], PyTorch [13], Scitkit-Learn [20], MXNET [8], and Caffe2 [9]. The automation component of MDSE's second pillar has been included by DAML. PGMs have the potential to be applied as MDSE models, which may result in a comprehensive software implementation that has been considered. This was suggested by Infer.Net [6,23]. Only C# was supported for the production of source code by them. Despite being the most applicable approach to the MDSE paradigm, related ML models and PGMs are not adequate to describe real-world IoT/CPS software systems and produce entire source code from model instances. This is the case even though they are the most applicable method. The comparison of the many different kinds of models and methodologies that are employed may be seen in Table 1.

Table 1: Literature compared to the ML-Quadrat

Method used in paper	Work	Model Type
Machine learning framework Libraries	TransFlow, Keras, Scikit	DAML Models
DAML workflow Design	JNIME, RapidMiner	DAML Models
Model Interchange Format	PMML, PFA, ONNX	DAML Models
ML-based Model	Infer.Net	MLSE Models
MDE4 IoT	IoT ML	SE Model

3. Background

There are a variety of connected devices in the IoT world, including wired and wireless ones. Things like low-power sensor devices and high-performance gadgets are included in this category. In IoT systems, the number of devices results in a variety of network designs. Smart surroundings are the primary goal of IoT systems [11]. To have a smart environment, gadgets must be able to automatically communicate with one another and be connected. Decisions can be made based on the flow of information across organizations. According to the decisions made, an organization may move to a new state or communicate this information to another organization. A wide range of sensors and devices resulted in a wide range of IoT heterogeneity.

3.1 Software components

Independently executable software is one of the most prevalent kinds of software components. It refers to a piece of software that can be installed and executed on its own. It is not necessary to have any knowledge of the internal workings of a component in order to use it. A component's proper operation may be shown by a COTS service provider. A service component can be deployed into a program without the need for the developer to know the programming language in which the component is implemented or the location where the component is being executed [22]. This eliminates the need for the developer to worry about the location where the component is being executed. Components of the system should operate independently of one another and should only be connected to one another when absolutely essential. Because the components of a system are only loosely linked to one another, changing one component of the system will not have any effect on the other components of the system. What defines a component are the many interfaces that it provides. Components may either provide or need interfaces from other components in order to perform their intended functions. Because there are potential components that either supply or demand comparable services in order to function, interface specifications need to be clearly and exhaustively stated [15]. Users are thus able to make educated judgments about which components to employ based on the specific needs that are unique to them. An example of a component that may be found in figure 3 is shown below.

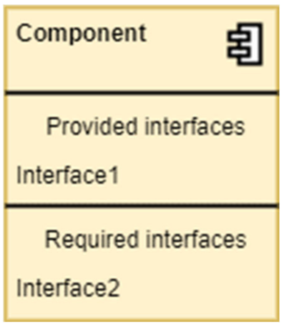

Fig. 3 Components structure.

3.2 Component-based software engineering

When building a software system using Component-Oriented Software Engineering, it is feasible to reuse pre-existing software components rather than having to start from zero and create them from scratch. The early year 2000 saw the introduction of a novel method known as COSE. The construction of components in the CBSE often makes use of object-oriented methodologies [16]. COSE places a greater emphasis on the reuse technique, while CBSE places a greater reliance on pre-built components.

The whole emphasis that CBSE and COSE place on the idea of components is a significant point of difference between the two organizations. As a consequence of the work done by COSE, systems may be broken down into two primary categories of primitives: connectors and components. A collection of connectors is used in order to accomplish the task of developing a finished software system. Conducting a domain study and erecting a domain model are the tasks that make up the first phase of the COSE development process. After doing an in-depth examination of a system's needs and specifications on an abstract level, it is feasible to disassemble the system into its component elements. After the process of decomposition, the definition of the abstract components comes next [17]. After that, it's time to begin searching for the various components that make up the system. The process of integration does not start until all of the components that were described have been identified. Utilizing the COSE software connections allows for the creation of an integrated system model. Figure 4 provides a visual representation of the COSE development process modeling.

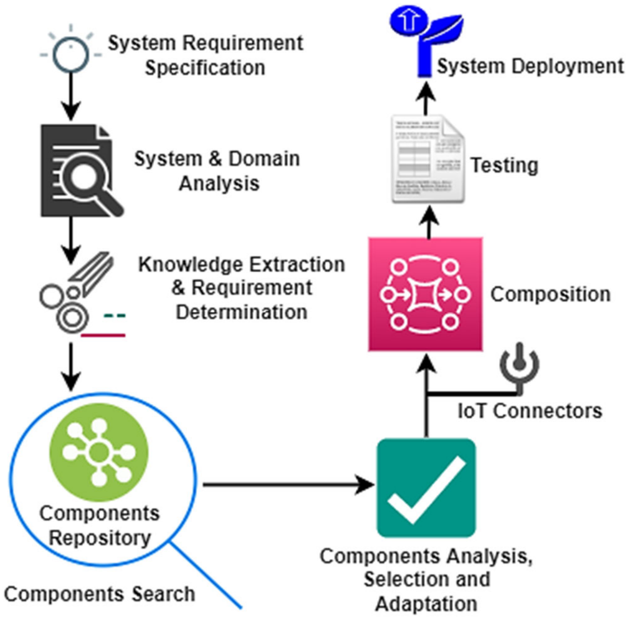
System Requirement Specification
System & Domain Analysis
Knowledge Extraction & Requirement Determination
Components Repository
Components Search
Components Analysis, Selection and Adaptation
IoT Connectors
Composition
Testing
System Deployment

Fig. 4 Component-Oriented Software Engineering Modeling (COSEM)

The architectural style is defined by a set of components, a topological arrangement, a set of semantic restrictions, and a set of connections. This style is an abstraction for a set of architectures that fit the definition of the style [25]. Garlan et al [26,27] have compiled a list of architectural styles based on their research into existing systems. Independent components, data flow-centered, data-centric and virtual machine architectures are all included in this short library.

Table 2: Architecture styles

Main Class	Characteristics
Independent components	The communication patterns are primarily controlled by processes, such as event systems and communicative processes.
Data Flow	In addition to the sequential and batch processing, transfer of data is accomplished by mechanisms such as pipelines.
Data Centered	For the purpose of modifying the main data store, independent calculations are used.
Virtual Machine	Systems that are constituted by the transformation of one sequence of instructions into another are examples of such a system. That are interpreters and rule-based systems.
Call and Return	Objects and call-based client-server architecture are examples of layered architecture, which typically employs a single thread of control.

4. Classification of Connectors

Interaction services provide for the possibility of a more generic classification of connectors; nevertheless, this does not explain the particular [19]. Connectors are separated into their own distinct types based on the types of interaction services they provide, such as an event, a linkage, a stream, a procedure call, an arbitrator, an adaptor, a distributor, and data access [28]. This is done so that new types of connectors can be developed, modeled, and analyzed.

In order to properly describe IoT systems inside COSE, IoT connections must first be defined and then implemented [20]. Connectors are used to link two different components using a variety of communication protocols in order to implement the solution outlined in [18]. Every component is a separate piece of the heterogeneous IoT system. It is safe to assume that any component that is coupled to a connection has a port on the connector that corresponds to it. Connector ports are responsible for sequentially handling the responsibilities of the physical layer, the internet layer, and the transport layer.

The connector software receives data from the ports, which it then processes in line with the information received from the ports. After additional components have been connected to the connection, it is then able to receive data packets from those other components [21]. Because they are aware of which other components are linked to them, connectors have the ability to do an analysis on core data. Because the connector has a second port that is linked to the second component, it is able to interact with the second component by exchanging data packets that are pertinent to the conversation. During the course of this task, data packets are generated and then sent on to the second component in line with the data packet structures of the components. Figure 5 shows an example of the Internet of Things connections and components.

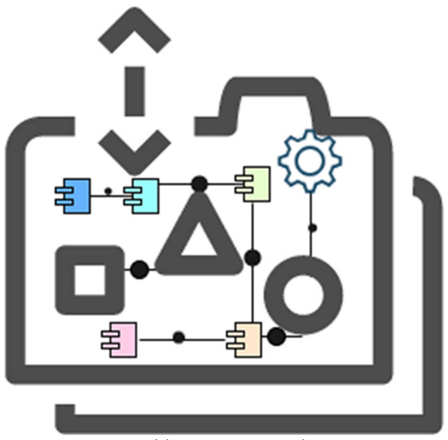

Fig. 5 IoT System with connectors and components.

5. Design Pattern

The term "software component technology" refers to the products and ideas that enable an approach that is based on the building of software components as from pieces that make up those components. Architecture and design are the patterns of high-level system structural design that are reflected by the types of components that make up systems, and the ways in which those components communicate and interrelate with one another [22], which is represented in the technology of software components [29]. The architectural pattern of high-level design is reflected by the types of components, this reflection is able to be seen when software components are used in both development tools and applications or systems that have been deployed. Component applicability analysis and deployment prediction are implemented by ML-based methods highlighted here below.

As a result, each component vector's values total to a predefined constant. For convenience, the total of each vector's axis is usually one after division by that constant. From here on, assume our design data collection consists of proportional or percentage vectors of:

$$p_i \geq 0, and\ \sum_{i=1}^{k} p_i = 1 \qquad (1)$$

The analysis' numerous ratios are impossible to compute. Zeros must be handled first. Presents a simple method for choosing stable imputed values. It's like multiplicative substitution.

$$r_j = \delta_j (if\ p_j = 0) \qquad (2)$$

$$r_j = p_j \left(1 - \sum_{l:p_l=0} \delta_l\right) (if\ p_j > 0) \qquad (3)$$

To change a proportional vector, divide each member by the geometric mean. The transformation can be applied after the zeros have been removed. The compositional vector r's CLR transformation is calculated as follows.

$$y = clr(r) = [\log \frac{r}{g(r)}] \qquad (4)$$

where

$$g(r) = [\prod_{i=1}^{k} r_i]^{1/k} \qquad (5)$$

6. Implementation and Results

The work presented in this paper does not analyze the usability and friendliness of a prospective user interface [23], since this was not its intended purpose for study. As a direct consequence of this, the system is easy to understand but not very user-friendly. The COSE model system's general architecture is shown below in Figure 6.

For instance, session basis design patterns may be realized by using session beans in their respective applications.

The session-based design pattern is responsible for handling activities relating to transactions and rollbacks. Establishing a connection to the database may be accomplished via the use of Data Access Object Design patterns or Entity Beans. The data access object design pattern maintains all of the JDBC needs in addition to a connection and resources drawn from a pool. Additionally, the data access object design pattern is responsible for managing failures and terminating connections.

The Data Access Object (DAO) pattern of the command determines what kinds of queries will be executed. So, what happens is the user supplies an XML file (or a class) that has instructions and queries in it. This framework is first designed with the COSE design pattern and tools, and it is then realized using programming language implementation. After some period of time, a component design pattern could become apparent as a result of using this framework. Figure 6 depicts the COSE design model for heterogeneous IoT systems.

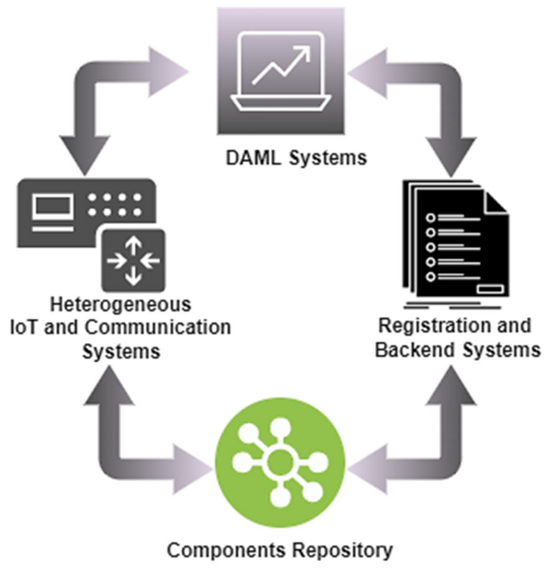
DAML Systems
Heterogeneous IoT and Communication Systems
Registration and Backend Systems
Components Repository

Fig. 6 System architectural framework.

The IoT makes use of a diverse selection of protocols [18], each of which is adapted to the unique characteristics of each IoT device. The following is a list of some of the Internet of Things protocols and communication systems [30,31] that are considered to be the most effective and widely used.

- Communication protocols such as 3G, 4G, LTE, and most advanced 5G.
- AMQP
- ZigBee 802.15.4
- CoAP
- Wi-Fi/802.11
- DDS
- TCP, UDP, IPv4, and future generation IPv6
- Bluetooth low energy (BLE)
- 6LoWPAN
- Z-Wave
- RFID
- SigFox
- PLC
- LPWAN
- MQTT

Communication between different types of Internet of Things devices, connectors, and components requires the usage of standardized protocols. The Internet Protocol, mostly abbreviated as IP, is a set of guidelines that determines how data is sent to and received from the internet. The protocols used by the Internet of Things (IoT) make certain that the information that is sent from one device or sensor to another device, a gateway, or a service is able to be read and comprehended by the respective gateways, devices, and services. There are several different protocols for the Internet of Things, each of which was developed and is functioning at its peak potential for a different initiative or goal. It is necessary to make use of the proper protocol in a suitable environment while working with the Internet of Things (IoT) since there is such a vast range of devices that are now available.

Wi-Fi was the protocol of choice for establishing a link between the internet network and the intended Internet of Things devices. The Wi-Fi standard is the one that's used for wireless networking. The practice of easing data flow between mobile devices and the internet network often makes use of it. Another ubiquitous standard that is used for networking devices and computers is known as Ethernet.

Ethernet is the name of the technology that is used for LANs (Local Area Networks) the majority of the time. It makes it possible to establish a hardwired connection between the devices and the internet so that a direct link may be created. It is a protocol that explains the process of how some interconnected devices can communicate their data well within the system or with other connected devices across the physical channel. This communication may take place between any two networked devices. It is part of the TCP/IP stack, which specifies the physical and link layers. Its responsibility is to describe how networked devices communicate their data. The cornerstone of this architecture is the IEEE 802.3 standard. Ethernet is a way that may be utilized inside of an IoT system in order to link stationary or fixed IoT devices. This is one of the purposes for which Ethernet was developed.

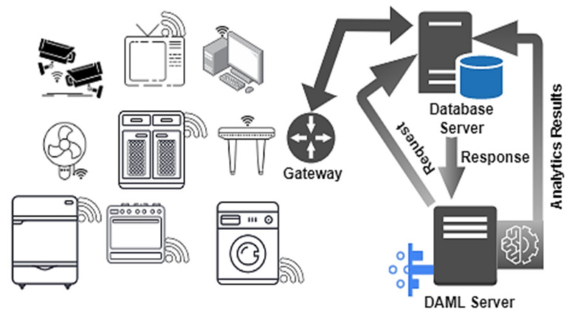

Fig. 7 Case study of the system architecture.

These appliances shown in figure 7 are assumed to have the same load or active power as the house as a whole for the time period. The goal is to forecast the washer and dryer's binary status (ON/OFF) at a given time. An 80 percent portion of the data was used to develop a supervised machine learning model for us. The ML model is tested on 20% of the samples, as a result. In machine learning, this is a frequent practice.

More research is needed to figure out if various splits might make things better or worse in the future. Furthermore, it is vital to note that it is not arbitrarily divided the data, as they are sequential (particularly time series) order and data in which they appear is critical. As a result, this supervised ML method in the Scikit-Learn package uses a multi-layer perceptron with one hidden layer of 100, Adam optimization and the Relu activation function, the and other default variables to classify a dataset.

Training the ML model referred to above, worked flawlessly on the test data. Metrics such as accuracy, precision, recall, and F1-Measure are frequently used in machine learning. Positive and negative classes in binary classification are defined as follows:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{6}$$

$$Precision = \frac{TP}{TP+FP} \tag{7}$$

$$Recall = \frac{TP}{TP+FN} \tag{8}$$

$$F1 - Score = \frac{2.Precision.Recall}{Precision+Recall} \tag{9}$$

True-Positive, True-Negative, False-Positive, and False-Negative situations are referred to as TP, TN, FP, and FN in the equations above. Other ML performance parameters, such as precision, recall, and F1-Measure, were all 99.9 percent, 100 percent, and 99.9 percent, respectively, in this trial. A highly capable MLP ANN classifier was expected to perform well given that the challenge was not difficult. There is no need to measure the performance of machine learning methods in this case study because we simply use the target libraries' APIs for this purpose. The examples are used to demonstrate the viability of the suggested method. Because the stated results here are purely for informational purposes, they do not contribute to the validation process.

Rather than guessing the labels of the ON/OFF classes, MLP ANN Regressor was implemented in Scikit-Learn. Both the Mean Absolute Error (MAE) and the Mean Squared Error (MSE), commonly referred to as the L2-Norm or the Euclidean Norm, are standard error metrics used to assess regression's effectiveness. This is how they're defined:

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |\widehat{y_i} - y_i| \tag{10}$$

$$MSE = \frac{1}{n}\sum_{i=1}^{n} (\widehat{y_i} - y_i)^2 \tag{11}$$

The plot of MAE and MSE in terms of percentage is shown in Figures 11 and 12 respectively, its data arranged in the table 4. We don't use the recommended approach to train an unsupervised ML model. ML is therefore created by hand. However, we use the same data set in our analysis. The black-box ML mode is used to link the pre-trained ML model to the software model. The remaining steps are identical to those in the unsupervised ML example presented before (including the performance).

These results are depicted in Figures 8 and 9, respectively. There is a clear correlation between the change in threshold and the change in recall rate. However, there is no discernible pattern in the rate of precision. Small thresholds lead to a significant number of components being found. As a result, the precision rate is low. This means that the precision rate rises when the threshold is increased since the number of components recovered decreases. To some extent, this decrease in the number of recovered parts is compensated for by an increase in the number of desirable components. It's not as evident as the shift in recall rate that precision has changed.

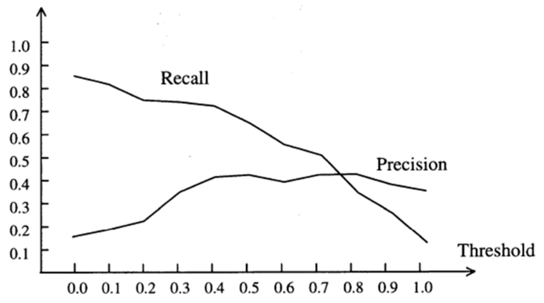

Fig. 8 Thesaurus-free precision and recall

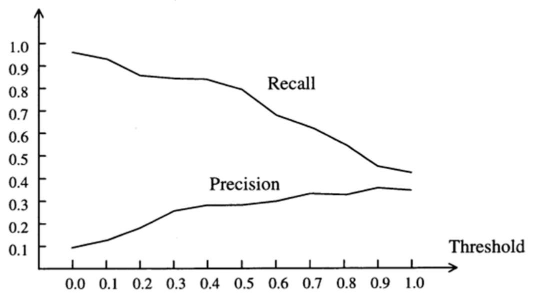

Fig. 9 Thesaurus with precision and recall

Table 3 shows the values of all the performance-related parameters for all the models. When compared to other models, the results of the hybrid strategy given in the tables show that it performs better. The performance graph is shown in Figure 10.

Table 3: Table for the performance of the model which is Accuracy, Recall, Precision and F1_Score

Model	Accuracy	Recall	Precision	F1_Score
SVM	0.8914	0.9145	0.8646	0.9615
ANN	0.9013	0.9465	0.8837	0.9632
ANFIS	0.9428	0.9579	0.9128	0.9756
Proposed	0.9832	0.9911	0.9598	0.9834

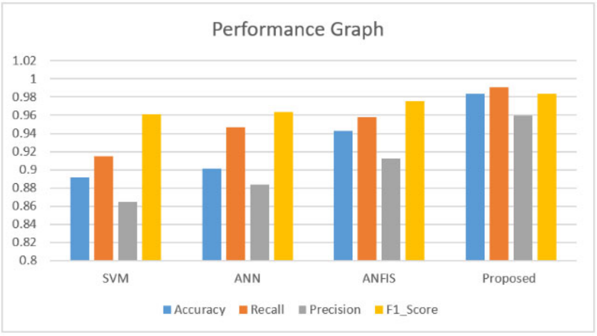

Fig. 10 Plot for overall performance.

Table 4: Table for the MAE and MSE

Model	MAE (%)	MSE (%)
SVM	93.4	6.6
ANN	94.2	5.8
ANFIS	96.5	3.5
Proposed	97.9	2.1

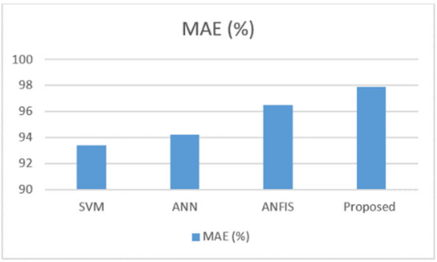

Fig. 11 Plot for MAE in terms of %

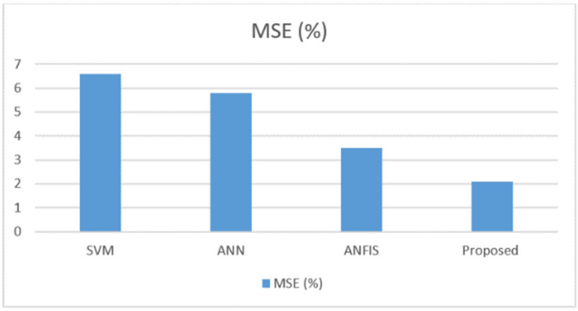

Fig. 12 Plot for MSE in terms of %

7. Conclusion

The idea of this research is for an architectural model for software development based on components rather than discrete pieces of code. Component-based software engineering, in contrast to traditional software engineering, makes it possible to deploy reusable software components, which traditional software engineering does not. The mapping of a software system to a collection of existing components is done concurrently via domain analysis and engineering as well as application engineering in the recommended paradigm for the software development process. The COSE tool has the capability of being upgraded to include newly developed Internet of Things components and connections. Before any more components that describe the protocol can be added, the precise packet structure of the IoT protocol must first be determined. After the user has uncovered specific knowledge about the packet's structure, the application level is where the payload data of the protocol has to be supplied. Examining the Internet of Things components that have been presented may be done with the help of the class diagrams that have been provided. The process of isolating and packing potential component candidates from legacy systems gave us the ability to disassemble and reassemble component parts.

Case studies have shown that design patterns may be used in the same manner as component-based design. On the other hand, this does not always mean that each and every design pattern can be immediately used as a components-based system implementation. It is not enough for a design pattern to be able to be utilized as a subcomponent inside a component in a component-oriented design; the pattern itself must also be employed. The use of design patterns has shown to be quite beneficial to the COSE model. In the case studies, it was determined that reuse was a more effective strategy. When the COSE system is being constructed, subproblems that have a higher effect are being addressed more rapidly. When it comes to design, you have the option of choosing either abstract or intricate patterns. However, there is no method to automatically produce code for the components of a program. It is quite evident that component and pattern catalogues are required. It's possible that the COSE tool will have the capability to search for and apply design patterns from the catalog, after which it will output code automatically. Further ML-based methods are used to analyze the performance of IoT systems with connectors implemented by the COSE model.

References

[1] S. U. Khan, A. W. Khan, F. Khan, M. A. Khan and T. K. Whangbo, "Critical Success Factors of Component-Based Software Outsourcing Development From Vendors' Perspective: A Systematic Literature Review," in IEEE Access, vol. 10, pp. 1650-1658, 2022, doi: 10.1109/ACCESS.2021.3138775.

[2] Sandeep SR, Ahamad S, Saxena D, Srivastava K, Jaiswal S, Bora A. To understand the relationship between Machine learning and Artificial intelligence in large and diversified business organisations. Materials Today: Proceedings. 2022 Jan 1;56:2082-6.

[3] D. Ameller et al., "Dealing with Non-Functional Requirements in Model-Driven Development: A Survey," in IEEE Transactions on Software Engineering, vol. 47, no. 4, pp. 818-835, 1 April 2021, doi: 10.1109/TSE.2019.2904476.

[4] Kaur H, Ahamad S, Verma GN. Elements of Legacy Program Complexity. International Journal of Research in Engineering and Technology. 2015;4(3):501-5.

[5] C. Yuan, Z. Liu, X. Wang and F. Yuan, "A Component Development Framework for Embedded Software," 2021 IEEE International Conference on Information Communication and Software Engineering (ICICSE), 2021, pp. 71-75, doi: 10.1109/ICICSE52190.2021.9404109.

[6] T. Lu, C. Liu, H. Duan and Q. Zeng, "Mining Component-Based Software Behavioral Models Using Dynamic Analysis," in IEEE Access, vol. 8, pp. 68883-68894, 2020, doi: 10.1109/ACCESS.2020.2987108.

[7] C. Paterson and R. Calinescu, "Observation-Enhanced QoS Analysis of Component-Based Systems," in IEEE Transactions on Software Engineering, vol. 46, no. 5, pp. 526-548, 1 May 2020, doi: 10.1109/TSE.2018.2864159.

[8] Moin, A., Rössler, S., Sayih, M., Günnemann, S.: From things' modeling language (thingml) to things' machine learning (thingml2). In: Guerra, E., Iovino, L. (eds) MODELS '20: ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems, Virtual Event, Canada, 18-23 October, 2020, Companion Proceedings, ACM, pp. 19:1–19:2, 2020

[9] Ali H. Dogru, "Component Oriented Software Engineering Modeling Language: COSEML", Computer Engineering Department, Middle East Technical University, Dec. 1999.

[10] A.A.K. Mohammad, M. A. Bari, S. Ahamad, M. Arshad, M. Ali Hussain, "Performance evaluation of reactive routing protocol using simulation knowledge", Materials Today: Proceedings, 2021, ISSN: 2214-7853, https://doi.org/10.1016/j.matpr.2021.01.752

[11] W. Dai et al., "Semantic Integration of Plug-and-Play Software Components for Industrial Edges Based on Microservices," in IEEE Access, vol. 7, pp. 125882-125892, 2019, doi: 10.1109/ACCESS.2019.2938565.

[12] S. Jha et al., "Deep Learning Approach for Software Maintainability Metrics Prediction," in IEEE Access, vol. 7, pp. 61840-61855, 2019, doi: 10.1109/ACCESS.2019.2913349.

[13] C. Diwaker et al., "A New Model for Predicting Component-Based Software Reliability Using Soft Computing," in IEEE Access, vol. 7, pp. 147191-147203, 2019, doi: 10.1109/ACCESS.2019.2946862.

[14] S. Ahamad, "System Architecture for Brain-Computer Interface based on Machine Learning and Internet of Things" International Journal of Advanced Computer Science and Applications (IJACSA), 13(3), 2022. http://dx.doi.org/10.14569/IJACSA.2022.0130357

[15] H. Sastypratiwi and Y. Yulianti, "Web Application Development using MVC-component-based approach," 2019 International Conference on Data and Software Engineering (ICoDSE), 2019, pp. 1-5, doi: 10.1109/ICoDSE48700.2019.9092609.

[16] Hartmann, T., Moawad, A., Fouquet, F., Le Traon, Y.: The next evolution of MDE: a seamless integration of machine learning into domain modeling. Softw. Syst. Model. (SoSyM) 18, 1285–1304 (2019)

[17] A. Triantafyllou, P. Sarigiannidis, and T. D.Lagkas, "Network protocols, schemes, and mechanisms for internet of things (IoT): Features, open challenges, and trends," Wireless Communications and Mobile Computing, vol. 2018, pp. 1– 24, 08 2018.

[18] R. Joshi, S. Mellor, and P. Didier, "The industrial internet of things volume G5: Connectivity framework," Industrial Internet Consortium (IIC), pp. 1–127, 2017.

[19] T. Joseph, R. Jenu, A. K. Assis, S. K. V. A, S. P. M, and D. A. G, "IoT middleware for smart city," IEEE International Symposium on Technologies for Smart Cities, 2017.

[20] S. Al-Sarawi, M. Anbar, K. Alieyan, and M. Alzubaidi, "Internet of things (IoT) communication protocols: Review," in ICIT 2017 Internet of Things (IEEE, ed.), pp. 685–690.

[21] M. C. Kaya, M. S. Nikoo, S. Suloglu, B. Tekinerdogan, and A. H. Dogru, Managing Heterogeneous Communication Challenges in the Internet of Things Using Connector Variability, pp. 127–149. Cham: Springer International Publishing, 2017.

[22] Xu Yingzhao. Research on the Development Trend of Integration of Embedded System and Internet of Things Wireless Internet Technology,2019,16(04):13-14.

[23] A. K. Sandhu and R. S. Batth, "A Hybrid approach to identify Software Reusable Components in Software Intelligence," 2021 2nd International Conference on Intelligent Engineering and Management (ICIEM), 2021, pp. 353-356, doi: 10.1109/ICIEM51511.2021.9445378.

[24] https://www.tutorialspoint.com/keras/index.htm

[25] Gorton, Ian. (2011). Understanding Software Architecture. 10.1007/978-3-642-19176-3_1.

[26] Kim, Jung & Garlan, David. (2010). Analyzing architectural styles. Journal of Systems and Software. 83. 1216-1235. 10.1016/j.jss.2010.01.049.

[27] Kang, Sungwon & Garlan, David. (2014). Architecture-Based Planning of Software Evolution. International Journal of Software Engineering and Knowledge Engineering. 24. 211-241. 10.1142/S0218194014500090.

[28] Mehta, Nikunj & Medvidovic, Nenad & Phadke, Sandeep. (2000). Towards a taxonomy of software connectors. Proceedings - International Conference on Software Engineering. 178-187. 10.1109/ICSE.2000.870409.

[29] Mark Richards, "Software Architecture Patterns", ISBN: 978-1-491-92424-2, O'Reilly Media, Inc, 2017.

[30] Madakam, Somayya & Ramaswamy, R & Tripathi, Siddharth. (2015). Internet of Things (IoT): A Literature Review. Journal of Computer and Communications. 3. 164-173. 10.4236/jcc.2015.35021.

[31] https://azure.microsoft.com/en-us/overview/internet-of-things-iot/iot-technology-protocols/

Dr. Shahanawaj Ahamad is an active academician and researcher in the field of Computer Science and Software Engineering with 17 years of experience. He completed 3 master's qualifications followed by a Ph.D. degree in Computer Science specializing in Software Engineering; contributed to publish 60 research articles and 3 books. He is designated as Asst. Professor and Program Coordinator of Software Engineering in the College of Computer Science and Engineering, University of Hail, Hail City, Saudi Arabia. He has been contributing significantly to various academic and administrative responsibilities, and a member of several scientific and research organizations including fellowship of British Computer Society UK. His research interest includes software engineering, software aging, program analysis, application of machine learning, IoT and cloud computing.