International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 6, No.4 (July-2017)

http://dx.doi.org/10.12785/ijcds/060404

Automated Query Generation for Complex Event Processing: A Shapelets, Parallel Coordinates, and Clustering Based Approach

R.N. Navagamuwa[1], K.J.P.G. Perera[2], M.R.M.J. Sally[3], L.A.V.N. Prashan[4] and H.M.N. Dilum Bandara[5]

[1,2,3,4,5]*Department of Computer Science and Engineering, University Of Moratuwa, Katubedda, Sri Lanka*

Received 19 Apr. 2017, Revised 28 May 2017, Accepted 15Jjun. 2017, Published 1 July 2017

Abstract: Automating query generation for Complex Event Processing (CEP) enables users to obtain useful insights from data, going beyond what it already knew. Existing automation techniques are both computationally expensive and require extensive domain-specific human interaction. We propose a technique that combines parallel coordinates and shapelets to automate the CEP query generation. Moreover, if the provided dataset is unannotated, we run it through a clustering algorithm to cluster the time instances into different event groups. Then each instance would be represented as a line on a set of parallel coordinates. Then the shapelet-learner algorithm is applied to those lines to extract the relevant shapelets and will be ranked based on their information gain. Next, the shapelets with similar information gain are divided into groups by a shapelet-merger algorithm. The best group for each event is then identified based on the event distribution of the data set and is used to automatically generate queries to detect the complex events. This technique can be applied to both multivariate and multivariate time-series data, and it is computationally and memory efficient. It enables users to focus only on the shapelets with relevant information gains. We demonstrate the utility of this technique using a set of real-world datasets.

Keywords: Clustering, Complex Event Processing, Multivariate Time Series, Parallel Coordinates, Shapelets

1. Introduction

With the increased bandwidth availability and lower costs, more sensors are being deployed to build smart and connected systems. However, gaining useful, real-time insights from such large streams of data is increasingly becoming difficult. Processing data on the fly with Complex Event Processing (CEP) and stream processing techniques [1] are gaining popularity to overcome such limitations in Internet of Things (IoT) applications. For example, CEP combines data from multiple, streaming sources to identify meaningful events or patterns in real time. While the detection of relevant events and patterns may give insight about opportunities and threats related to the data being monitored (e.g., set of sensor readings in IoT applications and credit card transactions), significant domain knowledge is required to write effective CEP queries. Manual analysis of large data streams is not only tedious and error prone, but also important events are likely to be missed due to the limited domain knowledge of the query writer. A promising alternative is to automate the CEP query generation by automatically extracting/mining interesting patterns from the past data [2], [3], [4].

Time-series pattern mining and classification techniques are extensively studied in the literature. Dynamic Time Warping (DTW) [5] is one such technique used to measure the similarity between two, time series based on a distance measure. However, the computational complexity of DTW grows exponentially with large and multiple time series limiting its usages. Moreover, the accuracy of the results depends on the chosen sliding window, which is nontrivial to estimate [2]. A shapelet [6], [7] is a time series subsequence that is identified as being representative of class membership; hence, useful in time-series classification. AutoCEP [2] proposed a shapelet-based technique to automate the CEP query generation for univariate time series. This itself is a major limitation, as most real-world time-series used in CEP tends to be multivariate. Moreover, AutoCEP generates queries for each and every instance of the detected event, requiring the CEP engine to concurrently process multiple

E-mail address: randika.12@cse.mrt.ac.lk, pravinda.12@cse.mrt.ac.lk, jaward.12@cse.mrt.ac.lk, prashan.12@cse.mrt.ac.lk, dilumb@cse.mrt.ac.lk

queries. This unnecessarily increases the computational and memory requirements of the CEP engine and consequently degrades its performance. One trivial optimization is to use the assistance of a domain-expert to aggregate the queries and attempt to write one or few queries. Ultra-fast shapelets [8] are proposed for multivariate time-series classification. Ultra-fast shapelets calculate a vectorized representation of respective attributes of the dataset. Then a random forest is trained to identify the shapelets with respect to the total dataset. The leaves of the random forest are considered to be the symbols. The number of occurrences of a symbol in the raw data is counted and these symbol histograms are used for the final classification using random forests. While this technique is effective in classification, it cannot be used to generate CEP queries, as the generated random forest does not support backtracking and obtaining any relevant information as to what data lead to the classification of the event [8]. Rare itemset pattern mining (AprioriRare) [9] is another technique. This technique cannot be used to detect events that occur within a short time span. Moreover, most related work focus only on domain-specific datasets limiting the usability of the proposed techniques across diverse datasets and applications [10], [11].

We propose a technique that represents the given multivariate data set as a set of parallel coordinates, and then extract shapelets out of those coordinates to automatically generate CEP queries. Even a multivariate time series can be mapped to a set of parallel coordinates, by representing each time instance as a separate line. Extracted shapelets are sorted according to the information gains and then divided into several groups. Out of the all groups, best group for each event is identified. Then the most important shapelets in the identified groups are used to generate one CEP query per group. This enables one to generate CEP queries for commonalities, anomalies, as well as time-series breakpoints in a given multivariate time-series dataset without having any domain knowledge. Users can focus on groups with high or low information gain depending on the application. Moreover, compared to related work, shapelets identify most relevant attributes in a dataset for a particular event, enabling us to write more efficient CEP queries and only one query per event (unless the same event is triggered by unrelated attribute combinations). While our solution up to this point assumes that the input dataset is annotated, many real-world datasets are not annotated. It is not trivial to annotate a large multivariate dataset without expert judgment and extensive manual work. Therefore, to realize the true benefits of automated CEP query generation, it is essential to be able to handle unannotated datasets. To address this problem, we also propose a clustering technique to cluster the time instances into different event groups. Using a set of real-world datasets, we demonstrate that the proposed technique can be applied effectively to auto generate CEP queries for common and abnormal events while identifying the relevant features and event occurrence timeframe. Moreover, the proposed technique has a relatively low computational and memory requirements compared to prior work.

Rest of the paper is organized as follows. Section II introduces shapelets, parallel coordinates, and problem formulation. Section III presents the proposed technique. Section IV explains implementation details and the proposed clustering technique to handle unannotated datasets. Performance analysis is presented in Section V. Concluding remarks and future work are discussed in Section VI. This is an extended version of the paper in [12], and the major extensions include the clustering technique proposed to cluster unannotated time instances into different event groups and expanded performance analysis with both annotated and unannotated datasets.

2. Preliminaries

We first define relevant terms and then define shapelets and parallel coordinates as applicable to the domain of CEP query generation. The research problem is then formulated.

A. Definitions

Time-Series — A time-series $T = t_1, \ldots, t_m$ is an ordered set of m real-valued variables.

Multivariate Time-Series — A multivariate time-series $T = t_1, \ldots, t_m$ is a sequence of m vectors, *where* $t_i = (t_{i,1}, \ldots, t_{i,s}) \in \mathbb{R}^s$ with s attributes/variables.

Sub-sequence (S^t_p) — Given a time-series T, a subsequence S^t_p of T is a sampling of length $l \leq m$ of contiguous positions from T starting at time p, i.e., $S^t_p = t_p, t_{p+1} \ldots, t_{p+l-1}$, for $1 \leq p \leq m - l + 1$.

Set of All Sub-sequences (ST_l) — Set of all possible sub-sequences S^t_p that can be extracted by sliding a window of length l across T is $ST_l = \{\text{all } S^t_p \text{ of } T, \text{ for } 1 \leq p \leq m - l + 1\}$.

Sub-sequence Distance — Given T and S^t_p *SubsequenceDist*(T, S^t_p) is the minimum distance between p contiguous positions obtained by sliding S^t_p across T. We use Euclidean distance as the distance function.

Entropy — Consider a time series data set **D** consisting of two classes, A and B. Let proportions of objects belonging to class A and B be $p(A)$ and $p(B)$, respectively. Then the entropy of **D** is:

$$I(\mathbf{D}) = -p(A)\log(p(A)) - p(B)\log(p(B)) \quad (1)$$

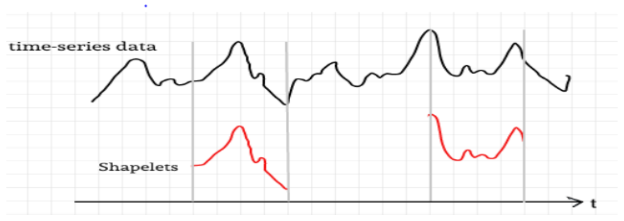

Figure 1. Time-series shapelets

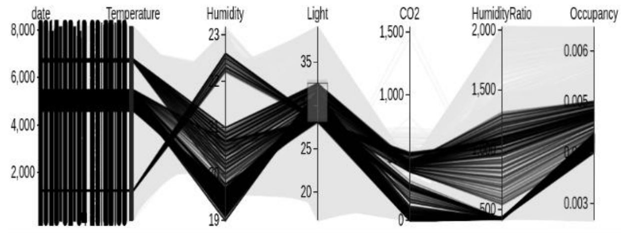

Figure 2. Parallel coordinates representation of Occupancy Detection dataset from [14]

Information Gain (Gain) — Given a certain split strategy *sp* which divides **D** into two subsets $\mathbf{D_1}$ and $\mathbf{D_2}$, let the entropy before and after splitting be I(**D**) and Î(**D**), respectively. Then the information gain for split *sp* is:

$$\text{Gain}(sp) = \text{I(D)} - \hat{\text{I}}\text{(D)}$$

$$\text{Gain(sp)} = \text{I}(\mathbf{D}) - (\text{p}(\mathbf{D}_1)(\text{I}\mathbf{D_1}\}) + \text{p}(\mathbf{D_2})\text{I}(\mathbf{D_2})) \qquad (2)$$

Optimal Split Point (OSP) — Consider a time-series data set **D** with two classes *A* and *B*. For a given S^t_p, we choose some distance threshold d_{th} and split **D** into $\mathbf{D}_1$ and $\mathbf{D_2}$, s.t. for every time series object $T_{1,i}$ in $\mathbf{D_1}$, $SubsequenceDist(T_{1,i}, S^t_p) \leq d_{th}$ and for every $T_{2,i}$ in $\mathbf{D_2}$, $SubsequenceDist(T_{2,i}, S^t_p) \geq d_{th}$. An Optimal Split Point (OSP) is a distance threshold that $Gain(S^t_p, d_{OSP(D,Stp)}) \geq Gain(S^t_p,d_{th})$ for any other distance threshold d_{th}.

B. *Shapelets*

As seen in Figure. 1 *Shapelets* can be defined as time-series sub-sequences. Shapelets can be of varying lengths, and many sub-sequences can be extracted by sliding a window of given length l. In shapelet-based classification, the objective is to identify a shapelet that is in some sense maximally representative of a class.

C. *Parallel Coordinates*

Parallel coordinates are widely used to visualize multivariate data [13]. Figure. 2 illustrates the parallel coordinates representation of the room occupancy dataset obtained from the UCI Machine Learning repository [14], which consists of six attributes. A dataset with *n* dimensions (i.e., attributes) is mapped to a set of points on *n* parallel lines, where each line represents an instance of data. These points are then connected using a line. A separate line is drawn for each instance of data (i.e., each row). For example, in Figure. 2 part of the dataset

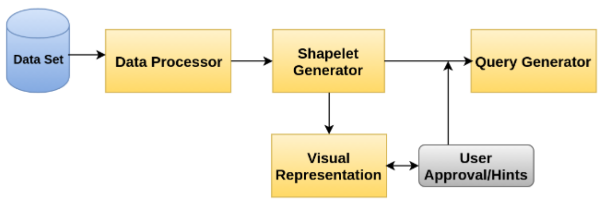

Figure 3. High-level architecture of the proposed solution

selected based on the *Light* attribute is shown in black, and rest of the data set is visualized in gray. When scaling these coordinate systems, it is recommended to use normalized data to prevent bias to certain dimensions.

D. *Problem Statement*

In contrast to the relational database systems that issue dynamic queries on stored and indexed data, CEP filters incoming streams of data through pre-written queries to detect events of interest. Hence, relevant queries need to be provided to the CEP engine a priori. We address the problem of needing domain knowledge to write a meaningful CEP queries through automation. Though a couple of related work attempt to automate CEP query generation, they support only univariate time series data [2].

We propose a solution which can be used to generate CEP queries for multivariate time series without requiring expert domain knowledge. In proposing the solution we assume that each instance in the obtained dataset is annotated according to the respective event (we relax this constraint in Section IV). In this work, we specifically focus on filter queries in CEP, as they are the most frequently used queries in CEP. Typically filter query has the following template:

$$\begin{array}{r} \textbf{SELECT } \{*\} \\ \textbf{WHERE } \{\text{attr}_1 \geq \text{a and attr}_2 < \text{b}\} \\ \textbf{WITHIN } \{\text{t}_1 \leq \text{time} \leq \text{t}_2\} \end{array} \qquad (3)$$

Therefore, the problem that this research attempts to address can be formulated as follows:

How to automatically construct a filter query per event, which contains the most relevant attributes, their range of values, and the event detection time frame?

3. Proposed Technique

To auto generate queries for Complex Event Processors, we propose the modularized architecture illustrated in Figure. 3. The four main modules perform the following tasks:

Data Processor — Converts the input dataset (e.g., time series data in text, XML, or CSV format) into a generic format used by the rest of the modules. If the dataset is pre-annotated, then each instance in the given dataset corresponds to an occurrence of a specific event, i.e., each data instance is classified/labeled with the corresponding event. The module then counts the number of events of each type, and their proportions with respect to the total number of events in the entire dataset.

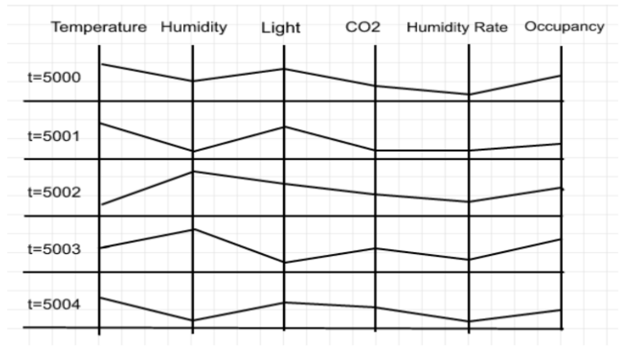

Figure 4. Multivariate time-series mapped as parallel coordinates

If the given dataset is not pre-annotated, we propose a clustering-based technique to annotate the dataset. The proposed clustering technique would cluster each data/time instance to a particular event. This is useful in calculating the information gain with respect to each shapelet and also to filter out the all generated shapelets to identify the most important shapelets.

Shapelet Generator — This is the core module of the system which uses pattern mining. This module identifies the most appropriate shapelets to represent each event. First, the multivariate (time series) dataset is mapped to a set of parallel coordinates. Figure. 4 is an exemplary representation of a multivariate time series with six attributes and five, time instances converted to parallel coordinates. Then all the shapelets are extracted from the parallel coordinates while varying the length l of the sliding window. Though the shape of the extracted shapelets depends on the order of the attributes, the final outcome of the solution is independent of the order. The length of an identified shapelet is bounded by the number of attributes m in the time series (i.e., $1 \leq l \leq m$). Therefore, our technique produces a much lower number of shapelets compared to prior work (e.g., [2]), where m can be as large as the length of the time series. Moreover, it is not required to apply heuristics or expert knowledge to determine the optimum minimum and maximum length of shapelets as in [2]. Therefore, our *Shapelet Learner Algorithm* is both computationally and memory efficient.

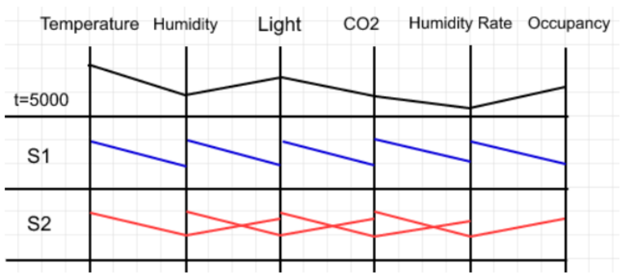

Figure 5. Shapelets slide across the time series

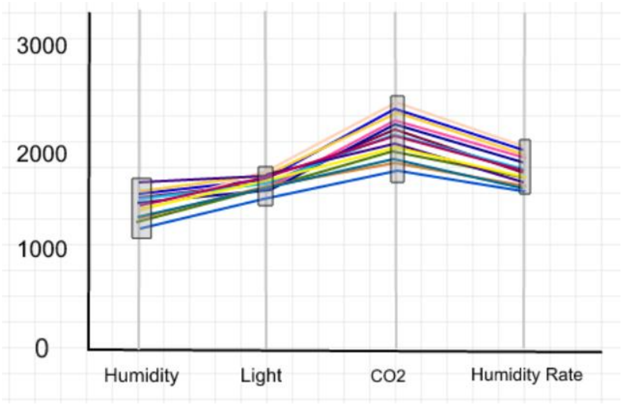

Figure 6. Shapelets that are representative of event

Once all shapelets are extracted, the next step is to identify a subset of the shapelets that are representative of patterns in the parallel coordinates. For this, we use information gain (Eq. 2) to quantify the extent to which a selected shapelet is similar to a given line on parallel coordinates. For example, Figure. 5 shows two shapelets, one with attributes 1 and 2 (shapelet S_1) and another with attributes 1, 2, and 3 (shapelet S_2). We slide both S_1 and S_2 across the line/row with $t = 5000$ and find the minimum distance between the shapelet and line. For example, S_1 has a relatively lower distance between the attributes 1-2 and 3-4, whereas S_2 has a relatively lower distance between attributes 1-3 and 4-6. This is estimated using the *SubsequenceDist()* function defined in Section 2.1. The same process is applied to all other time instances and shapelets. This results in a matrix of minimum distance values for each (shapelet, time instance) pair. Next using Eq. 1 and Eq. 2 provided in Preliminaries, we find the Optimal Splitting Point (OSP) [6] for each row of minimum distance values, to find the maximum information gain for each shapelet. The shapelets are then ranked based on the descending order of their information gain. We then use *Shapelet Merger Algorithm* to group shapelets within the ranked list with respect to their information gain. Because the shapelets with similar information gains produce similar insights, groups created using *Shapelet Merger Algorithm* allows us to cluster the similar informative shapelets together.

Finally, *Important Shapelet Extraction Algorithm* is used to identify the most suitable shapelets to represent each event type, which would result in an output similar to Figure. 6.

Visual Representation — This is an optional module that visualizes generated shapelets, enabling users to select what shapelets to choose for query generation. While the system can auto generate queries without any user suggestions, this module facilitates and accepts user approval allowing the user to select the shapelets that the user is interested in. As seen in Figure. 6 user may also select a subset of the attributes and their range of values that he/she expects to use in the generated queries. Such user intervention reduces false positives and improves the performance of the CEP engine, as not every identified event may be of practical importance.

Query Generator — Given the chosen shapelets, this module auto generates CEP queries based on the input provided by the hint generator module and incorporating any user-provided hints. Here we generate one query per each event with the relevant query parameters generated by the system, or set of attributes and ranges approved by the user. The module identifies the most relevant attributes and their value ranges to be used while constructing the query along with the optimal time periods within which each event occurs. Optimal time periods are identified by analyzing the event distribution of the actual dataset and choosing the longest event detection time period with respect to each occurrence of an event. Using these data, the module generates filter queries (similar to Eq. 3) for each and every event of the given dataset.

4. IMPLEMENTATION

In this paper, we introduce an enhanced way to define shapelets using parallel coordinates as an object with four attributes *s = (g, i, a, c)*. *g* is the information gain, which measures the similarity between shapelet and time series. *i* is the time-series identifier, which is the row number of the line on parallel coordinates (see Figure. 4). *a* is the starting column/attribute number. We store the normalized values of the attributes which belong to the particular shapelet in *c*. We also keep track of the original values *c*, as those are later required to generate CEP queries.

We first check on the dataset status and in case if it is not annotated we run the dataset through the proposed clustering technique and obtain an annotated version of the original dataset (see next Section for details). Next, we transform the multivariate time-series dataset into parallel coordinates as seen in Figure. 2. Our method builds upon two main phases which are illustrated in Figure. 7. Next, the implementation of each phase is discussed in detail.

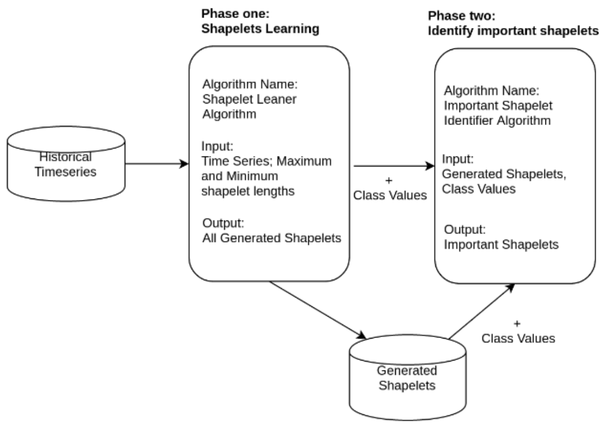
Phase one:
Shapelets Learning
Phase two:
Identify important shapelets
Historical Timeseries
Algorithm Name: Shapelet Leaner Algorithm
Input: Time Series; Maximum and Minimum shapelet lengths
Output: All Generated Shapelets
+ Class Values
Algorithm Name: Important Shapelet Identifier Algorithm
Input: Generated Shapelets, Class Values
Output: Important Shapelets
+ Class Values
Generated Shapelets

Figure 7. Architecture of the Shapelet Generator Module

Algorithm 1 Cluster Data

```
procedure ClusterData(Data,$\epsilon$,$m$,$NumberofEvents$)
    $normData \leftarrow Normalize(\text{Data})$
    $distList \leftarrow list()$
    $results \leftarrow array(\text{Data}.rows, NumberofEvents)$
    for $row$ in $normData$ do
        $distListRow \leftarrow list()$
        for $row1$ in $normData$ do
            $distance \leftarrow Euclidean(row, row1)$
            $distListRow.append(distance)$
        end for
        $distList.append(distListRow)$
        $opticsInstance \leftarrow$ OPTICS$(distListRow, \epsilon, m)$
        $opticsInstance.process()$
        $clusters \leftarrow opticsInstance.getClusters()$
        $noises \leftarrow opticsInstance.getNoise()$
        $numClust \leftarrow 1$
        for c in clusters do
            for value in c do
                if $NumberofEvents$ is 0 then
                    $temp \leftarrow numClust - 1$
                    $results[value][temp] \leftarrow results[value][temp] + 1$
                else
                    if $numClust \leq (NumberofEvents)$ then
                        $temp \leftarrow numClust - 1$
                        $results[value][temp] \leftarrow results[value][temp] + 1$
                    else
                        $temp \leftarrow NumberofEvents - 1$
                        $results[value][temp] \leftarrow results[value][temp] + 1$
                    end if
                end if
            end for$numClust++$
        end for
    end for$return \leftarrow results$
end procedure
```

A. Clustering Technique for Data Annotation

The proposed clustering technique becomes effective only if the user provides an unannotated dataset. Algorithm 1 initially calculates the Euclidean distances between each pair of data points or time instances. The resulting distance matrix is then clustered using OPTICS algorithm [15] resulting in an annotation for each time instance. In doing so, the dataset in common data format needs to be clustered in a manner in which each data/time instance is classified with respect to an identified event. The output of the clustering technique would modify the dataset by appending another column with the numerical values to denote the cluster number, which indicates the event type that each column belongs to. This information is then effectively used in the information gain calculation step of the query generator module.

The proposed clustering technique is domain independent and it is based on the numerical values within the dataset. We initially extract all numerical attributes of the dataset and then normalize those values. We then calculate the Euclidean distances between the data points of each time instance and produces a distance matrix. Then the resulting distance matrix is fed to the OPTICS algorithm, which clusters each row separately. The reason to use this technique is, in terms of detecting events, we look for different pattern instances within the obtained time instances throughout the dataset. Calculation of the distance of corresponding data points with respect to each time instances is one of the best methods to compare and identify the differences in patterns among the time instances. The key advantage of the proposed clustering implementation is its ability to work without any user input apart from providing the dataset, meaning any unannotated dataset could even be processed via our implementation.

The reason to have a distance matrix is that shapelets are distinguished according to the distances of each row. Thus, having a distance matrix to distinguish the dataset is more appropriate. Now each row is clustered with OPTICS algorithm, as it is an unsupervised, density-based clustering technique, which is more suitable for our approach as shapelets are extracted according to their similarity of distances and densities.

Then in the next iteration, the base time instance becomes the next time instance in the dataset and the above process will continue as explained. At the end of each iteration, the obtained Euclidean distances per each time instance with respect to selected base time instance is used to cluster the data points using the OPTICS algorithm. At the end, each time instance is put to a cluster that it belongs to. After scanning through the entire dataset we obtain the results array and scan through it and assign each time instance to the cluster which has the highest count in terms of its belongingness. This value will be appended to the dataset in which each time instance would have its corresponding event type.

Line 2 of the clustering algorithm (see Algorithm 1) normalizes the data and assign it to *normData* array. Then the for loop (line 5 onwards) starts to scan through each element in *normData* and for each of the element of this array, we calculate euclidean distance with all the other elements. The number of times line 8 is executed equals to the array size. This allows us to obtain a distance matrix *distListRow*. Afterward, each of the rows in the distance matrix is processed through the OPTICS algorithm to cluster, which contains one-dimensional clustering of the obtained distances. This is implemented in line 11 and 12. At the end, algorithm analyses the row-wise cluster distribution and assigns each row for the respective cluster which it happens to fall into most of the time. The rest of the code is implemented such that result array (line 4) is updated by giving the annotation.

To cluster the obtained Euclidean distances, we considered popular unsupervised, density-based clustering techniques, namely DBSCAN [16], OPTICS, and Single-Linkage Clustering [17]. One of the main drawbacks in DBSCAN is we have to decide parameters globally. Deciding parameters globally is important as without that the hierarchical nature of densities could not be measured. However, to do decide parameters globally, we need to have an idea of the data distribution within the dataset. Because from the beginning we intended to make the implementation domain and user independent, obtaining information on the data distribution within the dataset becomes infeasible. As DBSCAN cannot always be used to cluster data with different densities, we need to know the densities of data so that we can give a suitable threshold as a parameter. For instance, within the DBSCAN implementation, if a selected radius *r1* gives a cluster named *C* and another radius *r2* which is greater than *r1* gives a separate cluster named *B*, this would make *C* as a subset of *B*, which limits the precision of the derived clusters. This happens with inappropriate global parameter setting. This issue can be overcome in OPTICS algorithm by iteratively developing clusters starting from a small neighborhood radius.

Furthermore, hierarchical clustering techniques also do provide satisfactory results, but with the limitation of high time and memory complexity compared to density-based methods. Therefore, to go line with our objective of finding time instances of similar patterns, which are dense around another time instance, hence it is required to cluster the obtained Euclidean distance values considering the density and in doing so we did use OPTICS algorithm (which is an extension of DBSCAN which overcomes DBSCAN algorithm's limitations).

In terms of user interaction with our system, in which the user happens to be a domain expert, that user could provide the additional information such as the number of events within the dataset and proportionate event distribution to increase the accuracy of labelling the events. Conducting parameter tuning in the OPTICS algorithm also allows a user to increase the accuracy.

Algorithm 2 Shapelet Learner Algorithm

```
1: procedure SHAPELETLEARNER(D, l_min, l_max) ▷ Time series dataset D and
   shapelet length
2:     shapelets ← {}
3:     for each row r in D do
4:         wholeCandidate ← r
5:         l ← l_min
6:         while l ← l_max do
7:             l++
8:             start ← 0
9:             while start ← r.length do
10:                start++
11:                candidate ← {}
12:                m ← start
13:                while m<start + length do
14:                    m++
15:                    candidate[m − start] ← wholeCandidate[m]
16:                end while
17:                finalCandidate ← newShapelet
18:                finalCandidate.setContent(zNorm(candidate))
19:                finalCandidate.setRawContent(candidate)
20:                finalCandidate.setInfoGain(infoGain(candidate))
21:                shapelets.add(finalCandidate)
22:            end while
23:        end while
24:    end for
25:    return shapelets                 ▷ generated shapelets will be returned
26: end procedure
```

Algorithm 3 Shapelet Merger Algorithm

```
1: procedure SHAPELETMERGER(S, size)          ▷ Shapelet Array S sorted
   according to information gains and size of the group
2:     mergedShapelets ← {}
3:     count ← S.size()/size
4:     values ← {{}}
5:     currentRow ← {}
6:     while S.hasNext() do
7:         currentShapelet ← S.next()
8:         if count>0 then
9:             currentRow ← currentShapelet.getRawContent()
10:            rawSize ← currentRow.size() − 1
11:            classVal ← currentRow.get(rawSize)
12:            currentRow.remove(rawSize)
13:            currentRow.add(rawSize, currentShapelet.getSeriesId())
14:            currentRow.add(rawSize+1, currentShapelet.getStartPos())
15:            currentRow.add(rawSize + 2, classVal)
16:            index ← S.size()/size − count
17:            values.add(index, currentRow )
18:            count − −
19:        else
20:            count ← S.size()/size
21:            mergedShapelet.add(newShapelet(values))
22:            values ← {{}}
23:        end if
24:    end while
25:    return mergedShapelets           ▷ generated shapelets will be returned
26: end procedure
```

Algorithm 4 Important Shapelets

```
1: procedure GETIMPORTANTSHAPELETES(S, D, classValues)       ▷ Shapelet
   array S, Time Series Dataset D and array of class values
2:     shapeletArr ← list()
3:     classValProbs ← list()
4:     shapeletBucket ← map()
5:     clasNprob ← map()
6:     shapeDiff ← map()
7:     for i ← 0 to classValues.size() do
8:         var ← ShapeletBucket(classValues.get(i))
9:         classValProbs.add(findProb(D, classValues.get(i)))
10:        shapeletBucket.put(classValues.get(i), var)
11:    end for
12:    for all s ϵ S do
13:        for all val ϵ MaxProbClassVal(s).keys() do
14:            clasNprob.put(val,MaxProbClassVal(s).get(val));
15:            shapeletBucket.get(val).put(s);
16:        end for
17:    end for
18:    count ← 0
19:    for all c ϵ classValues do
20:        varMap ← map()
21:        aVal ← classValProbs.get(count)
22:        count ← count + 1;
23:        for all s ϵ shapeletBucket.get(c).getShapeletSet() do
24:            val ← clasNprob.get(c)
25:            varMap.put(s, val − aVal)
26:            shapeDiff.put(c,varMap)
27:            newMap ← shapeDiff.get(c)
28:            newShape ← GetMinDifShape(newMap)
29:            shapeletsArr.add(newShape);
30:        end for
31:    end for
32:    return shapeletsArr     ▷ Important shapelets array will be returned
33: end procedure
```

B. *Phase one: Shapelet Learner*

The *Shapelet Learner* extracts all the shapelets from the obtained parallel coordinates. We set the default minimum length (l_{min}) of a shapelet as two, while the maximum (l_{max}) is set to the number of attributes m (i.e., $2 \leq l \leq m$). However, a user may override these values. Afterward, Algorithm 2 of the Shapelet Generator module extracts all possible shapelets, while varying the shapelet length. First, the shapelet list is initialized to store the extracted shapelets (line 2). Then shapelets are extracted by going through each row *r* in the Dataset **D**. The outer loop increments the length of a shapelet *l* up to l_{max}, while the inner loop increments *start* to scans through each *r*. The inner-most loop (line 13-16) extracts all attributes between l_{min} to *l* for a given starting point. Then we convert each shapelet's content to standard normal using *zNorm()* function to prevent any biases to specific attributes. Moreover, in line 19 we also store the raw values of shapelets, as they are later required while generating queries.

We also calculate and save the information gain of each shapelets using *infoGain()* function (line 20). First, the *SubsequenceDist()* function is used to find the minimum distance between the row and the shapelet by sliding the shapelet across the row, one attribute at a time (see Figure. 5). By repeating the function, the minimum distance per each row is found and saved in an array. Then by sequentially splitting the array, we calculate the information gain using Eq. 2. Finally, for each shapelet, we get the maximum information gain and the corresponding split point, which would be the Optimum Splitting Point of the array. In line 21, each shapelet and its metadata are then added to the *shapelets* list, which is later returned by the algorithm.

C. *Phase Two: Shapelet Extraction*

All the extracted shapelets are first sorted according to their information gain. Then these shapelets are divided into a set of groups and then merged using Algorithm 3. The algorithm takes the set of shapelets *S* and number of shapelets per group (*group*$_{size}$) as the input. *group* $_{size}$ is proportional to the total number of shapelets. *group*$_{size}$ is selected based on the cluster pruning technique in which we set the number of groups to the square root of the total number of identified shapelets. If desired, the user may also define the $group_{size}$. Then in line 12 to 15, the grouped shapelets are updated by adding their series ID (i.e., raw number), starting positions (i.e., starting attribute index), and class value (i.e., event type). Finally, the algorithm returns all the merged shapelets (line 25).

Important Shapelet Finder algorithm (Algorithm 4) takes three parameters, namely merged shapelets, classified/labeled dataset, and class values (i.e., event types). Line 2 and 3 initialize two lists named *shapletArr* and *classValueProb*, which would respectively contain important shapelets and probabilities for class values within the total dataset. Then for each class value, a *set* data structure is created named *shapeletBucket*. *findProb()* function calculates the probability of the relevant class values within the dataset, and then put that to *shapeletBucket* (line 8 to 10). As the next step, (in line 12 to 17) each merged shapelet is included into a relevant *shapeletBucket*, based on the most probable class values for each group of shapelets. This is achieved using *maxProbClassVal*() function. Next, Algorithm 4 finds the absolute differences between the probabilities of actual events of the dataset and groups of shapelets. This is calculated using the *getMinDifShape*() function (line 28). Because the chosen group of shapelets per each event comprises of the minimum difference with respect to the actual event distribution in the dataset, it enables us to choose the most representative groups of shapelets per each event. Finally, the extracted group of shapelets are added to the *shapeletArr* (line 29).

Regardless of the CEP query language, two blocks are needed to generate a meaningful CEP query for an event (see Eq. 3). First, the time frame (or window) of the rule need to be identified from the extracted shapelets. This is specified using the *within* construct. Second, the conditions that need to be met on the captured sequence of events is defined using the *where* construct. Once the relevant parameters and constructs are known, we use the technique proposed in [2] to automatically generate the queries. The *conditions* are extracted using the selected attributes and their respective range of the important shapelets. If the user wants to get a CEP rule to identify multiple events, in addition to above two blocks, *filter* block should be added as follows:

within [*window*] {*relevent-events*}**where**[*conditions*] (4)

5. Performance Analysis

Here we provide an analysis of clustering technique as well as query generation for CEP using shapelets and parallel coordinates. We use two multivariate time series datasets obtained from UCI machine learning repository [14], [18] to conduct the performance analysis.

The accuracy of the proposed clustering technique is calculated with a direct comparison with the original datasets. The obtained datasets were annotated. For testing purposes, we removed the annotation and the rest of the attributes in the dataset were run through the proposed clustering technique, and the resulting annotation was compared with the original annotation. Moreover, we quantify the accuracy of the generated CEP queries using recall, precision, false positives, and false negatives. Furthermore, the computational complexity is theoretically analyzed.

A. *Occupancy Dataset*

Following results obtained upon the "Occupancy Detection" dataset provided in UCI Machine Learning Repository [14] which is a multivariate time series dataset which has seven attributes. The dataset itself consist of 8,143 instances out of which it has 6,414 instances which have a state of not occupied (*occupancy* = 0) and 1,729 instances which have a state of occupied (*occupancy* = 1) resulting approximately 78% of not occupied events and 21% of occupied events. Table I with summarized clustering results accounts for 92.55 % of accuracy.

TABLE I. Occupancy Dataset Clustering Results

Metric	Value
Maximum Radius (ε)	0.19
Minimum Number of Points (m)	2
Total number of time instances considered	1,100
Correctly clustered time instances	1,018
Incorrectly clustered time instances	82
Accuracy of the clustering technique	92.55% (1,018/1,100)

Our research has been conducted to solve the problem of generating queries to detect the occupied event, as well as not occupied event along with a timely representation. “Occupancy Detection” dataset [14] represents accurate occupancy detection of an office room from light, temperature, humidity, and CO2 measurements.

Figure. 8 illustrates the extracted shapelets visualization with respect to event 1 of detecting non-occupancy events as well as Figure. 9 displays the extracted shapelets visualization with respect to event 2 of detecting occupancy events. Most appropriate shapelets of Figure. 8 are within attributes 1 and 3 (i.e., humidity and CO2).

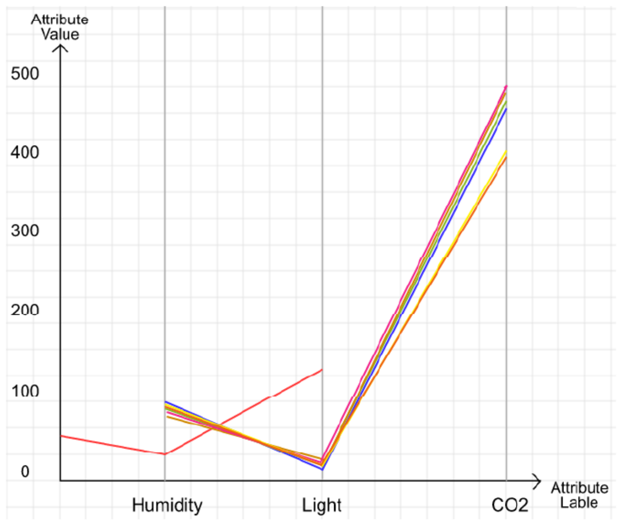

Figure 8. Shapelets corresponding to not occupied events

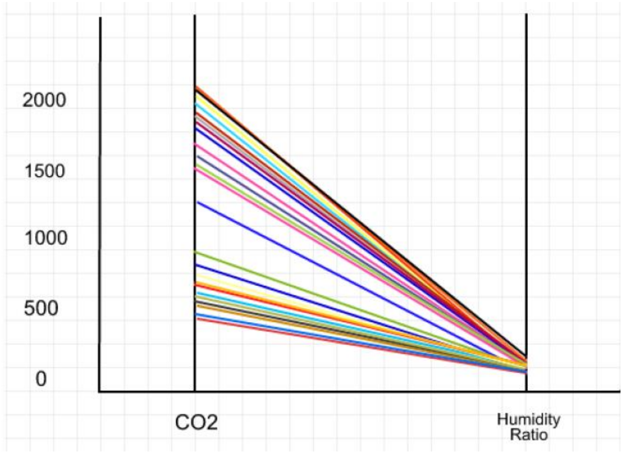

Figure 9. Shapelets corresponding to occupied events

TABLE II. OCCUPANCY DATASET EVENT DETECTION RESULTS

Event	Metric	Value
Not occupied	No of events in dataset No of events detected using CEP query Recall Precision False positives False negatives	291 286 98.28% 100.00% 0 5 (1.72%)
Occupied	No of events in dataset No of events detected using CEP query Recall Precision False positives False negatives	196 196 100.00% 84.48% 36(18.37%) 0

As seen in Figure. 9 for the occupied event CO2 and humidity ratio attributes are more relevant. The longest time window for not occupied events was between 17:32:00 - 22:23:00 on 2015/02/08. Whereas for the occupied events it was 14:49:00 - 18:40:00 on 2015/02/09. Based on these time gaps we set the event detection time frame. These attributes, their range of values, and the optimal event detection time frames are then used to generate queries.

Table II summarizes the accuracy of detected events based on the auto-generated queries. It can be seen that the generate CEP query is able to detect all the occupied events, it missed a few not occupied events (1.7% of total dataset). However, false positives for occupied events were relatively high (18.4%). Overall recall, precision, false positive, and false negative values are acceptable for both the occupied and not occupied events, indicating the usefulness of auto-generate CEP queries.

B. EEG Eye State Dataset

Following results obtained upon the "EEG-Eye State" dataset provided in UCI Machine Learning Repository [26] which is a multivariate time series dataset which has 15 attributes. Electroencephalogram (EEG) dataset related to opening and closing of eyes [18] which has been detected via a camera during the EEG measurement, and later used to annotate the EEG time series by analyzing the video frames. Eye-open state is indicated using binary 0 while the eye-closed state is indicated using 1. The dataset itself consist of 14,980 instances which comprise of time instances with respect to eye open and eye closed events.

TABLE III. EEG DATASET CLUSTERING RESULTS

Metric	Value
Maximum Radius (ε)	0.18
Minimum Number of Points (m)	2
Total number of time instances considered	1,000
Correctly clustered time instances	795
Incorrectly clustered time instances	205
Accuracy of the clustering technique	79.50% (795/1,000)

TABLE IV. EEG EYE SATE DATASET EVENT DETECTION RESULTS

Event	Metric	Value
Eye-open	No of events in dataset No of events detected using CEP query Recall Precision False positives False negatives	665 635 97.39% 100.00% 0 17 (2.67%)
Eye-closed	No of events in dataset No of events detected using CEP query Recall Precision False positives False negatives	69 68 98.55% 100.00% 0 1 (1.45%)

Table III with summarized clustering results accounts for 79.5 % of accuracy. Furthermore, Figure. 10 and 11 correspond to the most appropriate shapelets to detect event 0 and 1 respectively. Most appropriate shapelets for eye-open state are within attributes 8-13

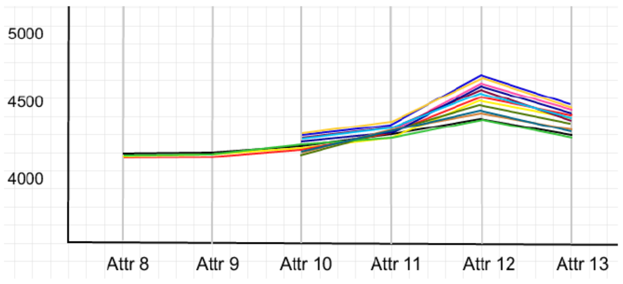

Figure 10. Shapelets corresponding to EEG eye-open state

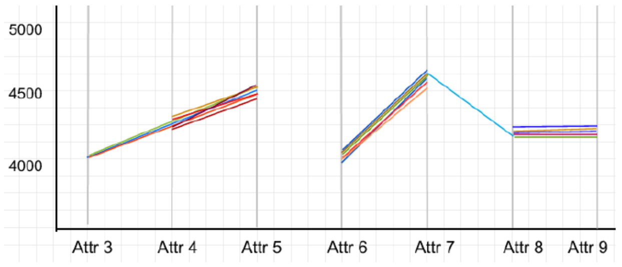

Figure 11. Shapelets corresponding to EEG eye-closed state

(Figure. 10). As seen in Figure. 10 attributes 4-5, 6-7, and 8-9 are more relevant for eye-closed event. The longest time window for eye-open state was between time stamps 128,349s to 204,516s, while for the eye-closed events it was 14,976s to 128,232s. We generate queries based on these three shapelets and event detection time frames. As there are three shapelets the **where** clause in Eq. 3 is of the form (*Attr 4's range* AND *Attr 5's range*) OR (*Attr 6's range* AND *Attr 7's range*) OR (*Attr 8's range* AND *Attr 9's range*).

Table IV summarizes the accuracy of detected events based on the auto-generated queries. For this dataset zero false positives are observed for both eye-open and eye-closed events. False negatively are also very low for both events (2.7% and 0.1%, respectively). Both the precision and recall are also close to 100%, indicating that results queries are able to detect relevant events with good accuracy.

Since the solution has been divided into four main algorithms, we have computed the time complexities of those four algorithms separately. Algorithm 1 has a time complexity of $\mathbf{O}(nm^2)$. Algorithm 2 has a time complexity of $\mathbf{O}(nm^3)$. Algorithm 3 has a time complexity of $\mathbf{O}(nm^2)$. Algorithm 4 has a time complexity of $\mathbf{O}(n^{3/2}m^3)$. Ultra-fast Shapelets [8] has a time complexity of $\mathbf{O}(pnm^2)$ and AutoCEP [2] has a time complexity of $\mathbf{O}(n^2)$ where n is the number of instances (time-series), m is the number of attributes and $p(< n)$ is a random number. Ultra-fast shapelets introduce a shapelet-based clustering technique in which they do not focus on query generation and AutoCEP only focuses on univariate domain makes it harder to compare those two techniques directly with ours as we cover full cycle from shapelet generation to the query generation.

6. Summary and Future Work

We proposed a technique to automatically generate CEP queries based on multivariate time-series data. The proposed technique initially clusters the multivariate time instances provided, the dataset is unannotated. Then it maps the annotated multivariate time-series to a set of parallel coordinates. Then key patterns that are representative of the events are identified using time-series shapelets. We also propose a technique to identify the most relevant shapelets per event, such that only a single CEP query will be generated per event. The proposed technique is both computationally and memory efficient compared to prior work, as the length of a shapelet is bounded by the number of attributes. Moreover, the performance of the CEP engine is also improved, as only one query will be generated per events. Furthermore, using two real datasets, we demonstrate that the resulting queries have good accuracy in detecting relevant events. In future, we plan to further improve the accuracy and extend the proposed technique to facilitate constructing CEP queries while capturing inter-dependencies among attributes in multivariate time series. Moreover, extending the techniques to other forms of CEP queries such as patterns and sequences would be of interest.

Acknowledgment

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of this paper.

References

[1] B. M. Michelson, “Event-driven architecture overview. event-driven soa is just part of the eda story,” Tech. Rep., 2006.

[2] R. Mousheimish, Y. Taher, and K. Zeitouni, "Complex event processing for the non-expert with autocep," in Proc. 10th ACM Intl. Conf. on Distributed and Event-based Systems, 2016, p. 340343.

[3] A. Margara, G. Cugola, and G. Tamburrelli, "Towards automated rule learning for complex event processing," Tech. Rep., 2013.

[4] G. Cugola, A. Margara, and G. Tamburrelli, "Learning from the past: automated rule generation for complex event processing," in Proc. 8th ACM Intl. Conf. on Distributed Event-Based Systems, 2014, p. 4758.

[5] A. Chotirat, E. Ratanamahatana, and Keogh, "Everything you know about dynamic time warping is wrong," in Proc. 3rd Workshop on Mining Temporal and Sequential Data, Seattle, WA, 2004.

[6] L. Ye and E. Keogh, "Time series shapelets," in Proc. 15th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining, 2009, pp. 947–956.

[7] Q. He, Zhidong, F. Zhuang, T. Shang, and Z. Shi, "Fast time series classification based on infrequent shapelets," in 2012 11th International Conference on Machine Learning and Applications, vol. 1, Dec 2012, pp. 215–219.

[8] M. Wistuba, J. Grabocka, and L. Schmidt-Thieme. (2015) Ultra-fast shapelets for time series classification. [Online]. Available: http://arxiv.org/abs/1503.05018

[9] L. Szathmary, A. Napoli, and P. Valtchev, "Towards rare itemset mining," in Proc. 19th IEEE Intl. Conf. on Tools with Artificial Intelligence(ICTAI 2007), 2016.

[10] H. Obweger, J. Schiefer, M. Stinger, P. Kepplinger, and S. Rozsnyai, "User-oriented rule management for event-based applications," in Proc. 5th ACM Intl. Conf. on Distributed event-based system, May 2011, pp. 39–48.

[11] A. Kavelar, H. Obweger, J. Schiefer, and M. Suntinger, "Web-based decision making for complex event processing systems," in Proc. 6th World Congress on Services, 2010, p. 453458.

[12] R. N. Navagamuwa, K. J. P. G. Perera, M. R. M. J. Sally, L. A. V. N.Prashan, and H. M. N. D. Bandara, "Shapelets and parallel coordinates based automated query generation for complex event processing," in Proc. IEEE Smart Data, 2016.

[13] J. Johansson and C. Forsell, "Evaluation of Parallel coordinates: Overview, categorization, and guidelines for future research," IEEE Trans. Vis. Comput. Graphics, vol. 22, pp. 579–588, 2016.

[14] R. Feldheim, UCI machine learning repository: Occupancy detection data set, 2016. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Occupancy+Detection+

[15] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "Optics: Ordering points to identify the clustering structure," in Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data, ser. SIGMOD '99, 1999, pp. 49–60. [Online]. Available: http://doi.acm.org/10.1145/304182.304187

[16] H. B¨acklund, A. Hedblom, and N. Neijman, "A density-based spatial clustering of application with noise," Data Mining TNM033, pp. 11–30, 2011.

[17] C. Jin, M. Mostofa, A. Patwary, A. Agrawal, W. Hendrix, W. keng Liao, and A. Choudhary, "Disc: A distributed single-linkage hierarchical clustering algorithm using mapreduce," in Proc. 4th International SC Workshop on Data Intensive Computing in the Clouds (DataCloud), 2013.

[18] UCI machine learning repository: EEG eye state data set, 2013. [Online]. Available: http://archive.ics.uci.edu/ml/datasets/EEG+Eye+State

R. N. Navagamuwa has graduated from the University of Moratuwa with a Bachelor's degree in Computer Science and Engineering. .He has successfully completed the Google Summer of Code Program in 2016 with Eclipse Foundation. His main interest areas are Algorithms, Distributed Programming, Enterprise Integration, Web Technologies and Image Processing. He currently works as a software engineer at AdroitLogic Lanka (Pvt) Ltd.

K. J. P. G. Perera has graduated from the University of Moratuwa with a Bachelor's degree in Computer Science and Engineering. His research interest is in the areas of Microservice Architectures, Big Data Analytics, Cloud Computing, Complex Event Processing and Web Technologies. He currently works as a software engineer at CAKE LABS (Pvt) Ltd

M. R. M. J. Sally has graduated from the University of Moratuwa with a Bachelor's degree in Computer Science and Engineering. His research interest is in the areas of Distributed Systems, Cloud Computing, Parallel Computing, and Scientific Computing. He currently works as a software engineer at ShipXpress (Pvt) Ltd.

L. A. V. N. Prashan has graduated from the University of Moratuwa with a Bachelor's degree in Computer Science and Engineering. His research interest is in the areas of Distributed Systems, Big Data Analysis, Cloud Computing, Microservice Architectures, and Complex Event Processing. He currently works as a software engineer at WSO2 Lanka (Pvt) Ltd.

H. M. N. Dilum Bandara received B.Sc. (First Class Honors) in Computer Science and Engineering from the University of Moratuwa, Sri Lanka in 2004 and M.S. and Ph.D. in Electrical and Computer Engineering from Colorado State University in 2008 and 2012, respectively. He is a Senior Lecturer at Dept. of Computer Science and Engineering, University of Moratuwa, Sri Lanka. His research interests are in the areas of IoT, Data Engineering, Distributed Systems (Cloud, P2P, and HPC), and Security. He is a member of the IEEE.