An Edge Centric Middleware for City Surveillance Platform (CityPro)

Mariam Hakim
Faculty of Sciences - I
Lebanese University
Beirut, Lebanon
mariamhakim181@gmail.com

Ibrahim Tawbe
Faculty of Sciences - I
Lebanese University
Beirut, Lebanon
ibrahim.tawbe@gmail.com

Mohamed Dbouk
Faculty of Sciences - I
Lebanese University
Beirut, Lebanon
mdbouk@ul.edu.lb

***Abstract*—Smarter Cities provide better management for city services, reporting problems, and predicting future issues to enhance city protection. The need for a robust and unified middleware platform in Smart Cities is not yet a solved problem. Traditional middleware that evolved in enterprise environments can't handle smart cities complexities. Therefore, a platform -CityPro- that integrates multiple existing systems into a collaborative environment to protect the city is proposed. It highlights the concept of "connectors" between different city systems and a central "core" system. In this article we provide a new middleware (architecture and framework) - called Edge Centric Middleware- that adapts CityPro's architecture to connect existing city systems to the core system. The suggested solution tries to solve issues such as merging heterogeneous systems, event processing, real-time data, scalability, availability, and interoperability, while exploring edge computing capabilities. We also provided an implementation for this middleware to serve as an open-source generic framework that can be extended and customized.**

Keywords—Smart Cities; Business Processing; Edge Computing; Business Intelligence; Big Data; Smart Data

I. Introduction and Problem Position

In a typical modern city, multiple computerized standalone systems exist, e.g. banks, hotels, and hospitals. All of these systems work separately which limits their powers to a specific business domain (e.g. bank), specific area, etc. The amount of challenges a city faces is increasing daily. Smart City paradigms approach these challenges by using the city's resources efficiently to optimize services and managements such as traffic control, electricity, public safety, etc. It is worth to note that tackling these issues using technological advancements is not for "modern" cities only. Cities in developed countries can and should take measures following Smart City paradigm to solve challenges. Smart City is not a one-shot solution; developed countries can gradually build up their Smart City eco-system.

Public safety is not considered a luxurious service; it's an essential challenge for any city. Cities are facing variety of risks such as natural disasters, terrorists' attacks, crimes, vehicle accidents, etc. What triggers these risks is traditionally monitored by different governmental agencies; the weather monitoring agency differs from local police forces. On the other hand, these risks put citizens in danger and coordinating the emergency procedures is critical to lower the losses. Furthermore, detecting and dealing with an emergency is not enough, there is a need to predict and act before things happen. That's why models are built and run to simulate real-world scenarios using machine learning and artificial intelligence. In addition to the models, early alarms sometime come from the correlation between different data sources. The data produced by these sources may become gigantic over time; such data can be considered as a valuable mine of information. Therefore, a technique must be adapted to access this data wisely in order to benefit the whole city.

These necessities nourished the idea of an integrated platform of a collaborative surveillance system, called CityPro [1].This system is intended to protect and monitor people and public infrastructures. It is expected to:

- Operate within live-mode by using the city digital infrastructures
- Combine and inter-operate heterogeneous pre-existing operational systems

A. *CityPro; an overview*

Ref [1] presents CityPro; a collaborative platform for city protection tries to standardize the relation between different systems with a centralized, supervised control and data repository architecture shown in Fig. 1. It defines data providers as domain-specific independent (stand-alone) systems that coexist within the considered territory such as police departments, fire stations, banks, etc. In the context of CityPro these systems are data producers. They are accessed through "collaboration-links" or "connectors". CityPro defines a connector as "Dedicated links that are materializing the collaborative inter-relationships between CityPro components. They mainly consist of ETL like dedicated data exchange automated protocols based on 'adapter pattern'." CityPro delivered the general architecture of the system, while we still need to investigate deeper into providing a standard and uniform access to these heterogeneous distributed systems with minimum effort at the data provider side.

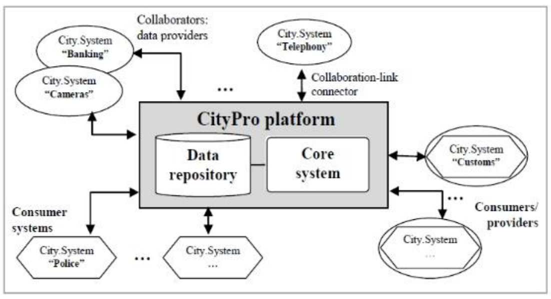


Fig. 1. CityPro's General Architecture

To address the need for city surveillance, CityPro defines two data flows: periodically where data continually arrives from data providers and on-demand where the central system asks providers for instant detailed data. We need to define the standard protocols for this data exchange.

Highly distributed systems of this kind may produce gigantic amounts of data. A pure central system might struggle in handling all this data flow and delivering value, especially in the case of near real-time alerts. That's why it's tempting to use the distributed computation environment along the process of detecting anomalies and preparing the data for analysis.

Privacy and security are always an issue in any collaborative and network-based solution. In today's world the approach to this issue is a mix between political or governmental policies and technical implementations. From the technical side we should take these issues into consideration from early stages, starting from the design of the system.

Applying our concept to CityPro, provides a middleware between data providers such as (banks, customs, hospitals…), the existing information systems, and the "Core" system of the city. Although we started by challenges raised by CityPro, we designed and implemented the middleware to be generic, opensource, and customizable so it can be applicable in many scenarios.

B. *Problem position and proposed approach*

CityPro intends to combine and inter-operate (in a supervised mode), heterogeneous pre-existing operational systems; e.g., banks, hospitals, cellular/landline phone management engines, police-stations, video surveillance networks, etc. The proposed architecture should provide insights to protect the city and some events should trigger real-time alerts. Yet we are faced with major challenges. The data providers are heterogeneous distributed systems, where each provider has implemented its own technological stack (Hardware-OS-Database-App layer). In addition most data providers hold big data volumes. In its architecture the core system which holds the federated data repository is responsible for analysis and decision making.

This article is dedicated to finding a feasible solution that uses edge computing concepts to optimize data transfer and delegate some core system operations and computations to near-source components especially in the case of real-time alerts, while maintaining the security when moving and accessing remote data from other parties. These operations should scale accordingly and should be done reliably where the system should guarantee high availability and fault tolerance.

In the later sections of this paper, we are going to introduce an architecture that adopts edge computing concepts and incorporate techniques to access distributed heterogeneous information systems. By creating a middle layer between the Core System and the data providers such as banks, police, airports, customs, etc. The proposed architecture is expected to enhance the interoperability within the system itself and the city systems. It is also expected to maintain a high availability and create a scalable cache for messages from various providers.

In this paper, Section II covers the previous works and technologies. The work and solution we are adapting for the smart repository in section III. Section IV presents a basic implementation and evaluation of the work. Finally, Section V concludes and states some future works.

II. Technologies and State of The Art

Traditional middlewares aren't fit to play the role of CityPro's connectors. Additional criteria and functionalities are required. Therefore, reviewing traditional middlewares in the edge computing era opens a door for more options and facilities.

A. *Edge and Fog Computing*

Ref [2] refers to edge computing as the enabling technology that performs computation at the edge of the network, on downstream data on behalf of cloud services and upstream data on behalf of IoT services. Another technical trend is Fog Computing a term created by Cisco. While many interchange it with edge computing, Cisco states that "Fog computing is a standard that defines how edge computing should work" [3].

B. *Middleware in the Context of Cities & Surveillance*

Before edge computing and smarty city era, "Embedded Middleware on Distributed Smart Cameras" [4] designed and implemented a middleware for distributed embedded image processing on a network of smart cameras. They embedded the middleware in the camera device. In our case CityPro, instead of the camera or device there is a complete information system – the data provider. Unlike the embedded middleware used in [4] where they interchange data between cameras, in CityPro data partners don't interact with each other. Furthermore, in CityPro an embedded middleware can't handle big-data, scalability, availability, and other processing tasks. That is why we evolved the concept of embedding to an edge device.

1) Civitas

Ref [5] highlighted that traditional middleware technologies are designed for enterprise environments and can't address issues of heterogeneity and scalability in a Smart City. They connected different entities such as citizens, governmental institutions, and companies to the "Civitas" platform which is considered as the core of the IT infrastructure in the Smart City. The connection is through a device called "Civitas Plug", these devices can be smartphones, company servers, residential gateways…

Another key point in the Civitas platform is the "Core Nodes". They are servers that host different kind of services to the city entities. They consider that heterogeneity and inter-operability is solved by the distributed object-oriented middleware, where each entity is an object with a set of defined standard interfaces. It also supports event-based communication through publish-subscribe pattern.

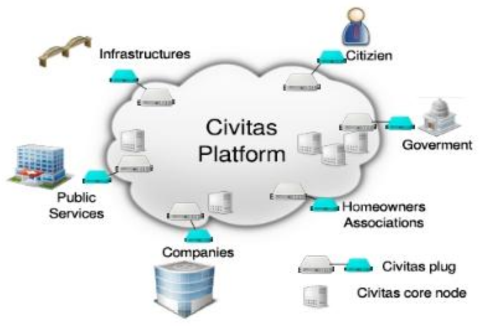


Fig. 2. Civitas Platform [5]

We share the idea of the plug device, but this device can't alone solve the availability and scalability issue, that's why we added a broker layer between our "edge device" and the "core system". Moreover, we needed a deeper study of the inner-design and inner-workings of the plug device.

2) InterSCity

Ref [6] noted that there is no agreed middleware platform for SmartCities' platforms. They listed three factors for this challenge: security and privacy policies, the lack of scientific and practical validation, and the "extensive use of development of non-opensource software" which is causing inter-operability issues and limits the collaboration among researchers. They applied the micro-service paradigm to provide a modular and scalable middleware. InterSCity microservices architecture is shown in Figure 4. The abstraction is through "city resource" a logical concept that resembles a physical entity such as cars, traffic lights, etc.

Each resource has attributes and functions to provide data and receive commands. For communication protocols, microservices use synchronous HTTP Rest API and asynchronous message bus using RabbitMQ. Their work adopts many open source projects such as PostgreSQL and Redis. In CityPro context the distributed existing information systems are more data producers that service providers. Furthermore, to some extent, middlewares and micro-services solve different problems. Micro-services architecture takes the whole application (Smart City) and de-couples it into independent services.

C. Accessing Distributed Heterogeneous Database Systems

Today's computing and analysis techniques are tempting to integrate different Information Systems to provide additional insights. On the other hand, many international enterprises are providing services that span different subjects in different locations. In CityPro project which tries to maximize the benefit of the distributed operating systems, there is a clear case for this challenge.

Ref [7] highlights the need to "combine and analyze the distributed data along with contextual factors". The authors list the current solutions and technologies in the cloud computing infrastructure stack (Azure, Amazon, private stacks) in three main domains: managing distributed clusters, distributed data processing models (such as MapReduce), and the data management service across datacenters which "integrated different cloud data storage services by providing a transparent interface" such as Simple Cloud API , PDC@KTH's proxy service, Open Grid Services Architecture Data Access and Integration OGSA-DAI).

1) MUSYOP

Ref [8] provided "a federated approach - a mediator server - that allows users to query access to multiple heterogeneous data sources" for relational databases, Triplestore, NoSQL databases, and XML with a management layer using SPARQL and mapping different databases to RDF.

2) Apache Spark

One more interesting solution for accessing different datasets is Apache Spark. Apache Spark is a "unified analytics engine for large-scale data processing" [9]. The unified part is baked into the Spark SQL module. Spark SQL Layer is built on top of two interfaces Data Frame API and Data Source API which supports schema understanding, reading data with filters, and writing custom aggregations.

There are two important sides of this topic: distribution and heterogeneity. For the heterogeneity part there are two trends: use ontology-based solutions or build custom interface layer. It's also important to note that whatever the integration solution. SQL is the preferred language to query these distributed datasets. The SQL layer is used for the unified access with the added benefit that it can easily integrate with upper layer tools and technologies such as BI tools. Often suggested solutions tackle the whole process from accessing the data to integrating it, but my focus is on providing the interface to different database systems and the integration part will be solved in later phases of CityPro.

D. Message Broker

A Message Oriented Middleware (MOM) is responsible for sending and receiving data encapsulated in messages between different distributed systems. A MOM can be with a broker or broker-less. TIBCO Inc. [10] defines a message broker as a discrete service that provides data marshaling, routing, persistence, and delivery to all appropriate consumers.

We will highlight different technologies and researches with respect to important features we are interested in persistent cache, high availability and fault tolerance, scalability, with added value features such as message formats optimizations for binary messages and compression. First we consider current production-grade technologies. The state of art in persistence is to use a journaling file system write-ahead commit log backed by operating system page cache this is implemented in Apache Kafka [11] and Apache ActiveMQ Artemis [12] or delegate this to a database that uses this implementation. For high availability and fault tolerance, a replication set of 3 brokers is recommended for production. Scalability is done horizontally by adding more nodes as brokers, but a consensus and management service is needed to keep track of nodes and data index in the cluster. One of the well-known and heavily used software that implements this functionality is Apache Zookeeper [13] which itself can be replicated. Ref [14] introduced "EQS: an Elastic and Scalable Message Queue for the Cloud". They discuss automatic scaling and load balancing in message queues to optimize the throughput along systems by layering additional components for monitoring, rules, and scaling management.

Custom drivers for different database engines will use these interfaces to support those functionalities. Currently drivers for most database engines (MySQL, MongoDB, HBase, Cassandra, HDFS ...) are already implemented and ready for production use.

E. Complex Event Processing (CEP)

With various systems and sensors generating and sending data, there is a need to detect interesting patterns along the data streams. CEP paradigm has an opposing concept to regular databases. Instead of executing a query on a dataset, the data is executed on a well-defined query. This technology has been deployed and heavily used in the financial sector especially for fraud detection.

Ref [15] introduced the concept back in 1998. It was a hot topic again in the research community in 2006-2009 where it was discussed in the context of big-data and adding machine learning for prediction of events. Many commercial and open-source CEP systems are available such as Apache Flink [16], Siddhi.io [17], and Esper [18].

III. Proposed Solution; An Edge Centric Middleware for Citypro

The "Edge Centric Middleware" consists of two parts: edge devices distributed along with the data providers' systems and a broker between edge devices and the "City Core System". We delegate some computation tasks to the distributed edge devices which provide a uniform and standard interface to the variety of existing systems. The edge devices follow the concept of black box to tackle privacy and security concerns. While the broker handles the scalability and availability of message queues from a distributed network of edge devices.

A. General Architecture

There are two main parts for the Edge Centric Middleware: the edge device and the broker as shown in Fig. 3.

The Edge Device - It's a software and hardware package deployed at the edge network of the data provider and connected to the existing system via computer network communications. It can scale up from a simple computer board such as Raspberry-Pi [19] to a rack of servers. The main roles of the edge device are:

- Provide an interface to access different databases at rest at the provider side according to a schema contract required by the government agency
- Consume live data from the provider side according to a schema contract

- Provide the required computation resources to host and execute data summary and ETL-like operations
- Send batches of data according to the configured time interval and schema contract
- Detect and propagate real-time alerts specified by a defined list of triggers
- Enabler for the confidentiality and integrity of the data and business rules in question

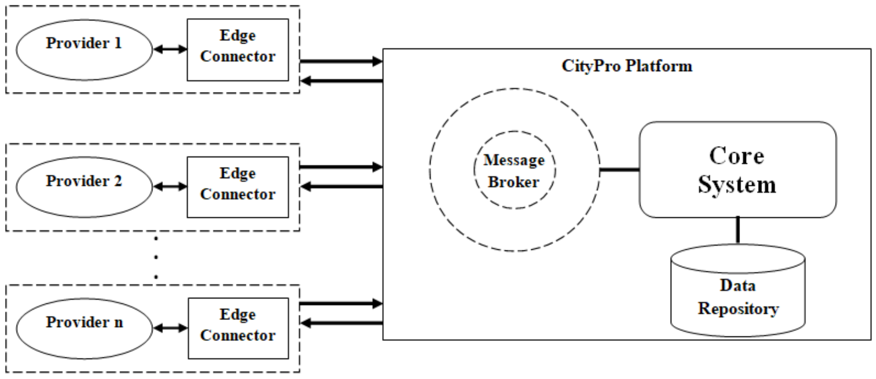


Fig. 3. Proposed Solution General Architecture

The Broker - It is between the network of distributed edge devices and the core system. Only our certified edge devices can connect to the broker, this enhances the privacy and quality of data flowing through the middleware. From the broker point of view edge devices are data producers and the "Core System" is a data consumer. The broker roles are:

- A message queue that supports high volume and speed of data
- Support publish-subscribe pattern
- Support high availability in case of systems failures and high traffic
- Support horizontal scalability to handle existing and new systems

While we invested more on the edge device part to propose a new framework, we opted to rely on existing technologies for the broker side. In addition to realizing the above criteria, where traditional message queues support producer and consumer, some brokers also support a logic- processing endpoint such as Apache Kafka's [11] Processing API which can be used for data integration at this stage before consuming the data.

B. Edge Device Software Framework

The edge device framework is a software & hardware package. In this section, we will describe the software stack of the edge device (Fig. 4). We decomposed the framework into subcomponents with decoupled functionalities. We used a file-based configuration for some settings and standard communication protocols for inter-component communications and for the outer interfaces whether with the data provider or the core system.

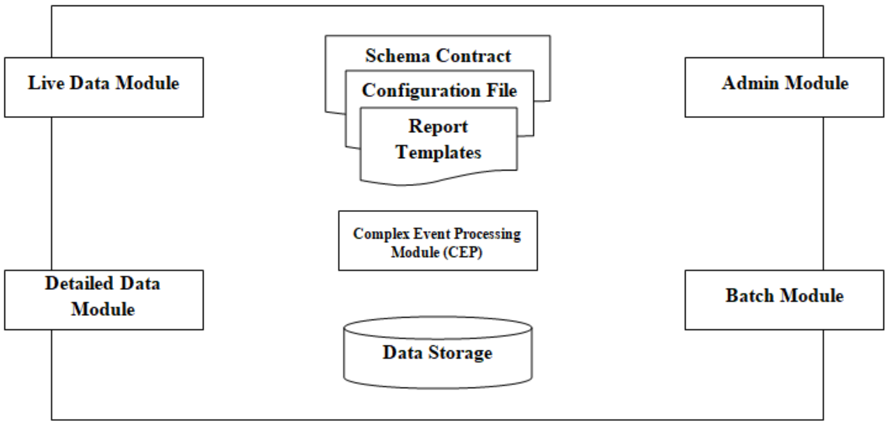


Fig. 4. Edge Device Inner Components

Schema Contract: States the required (selected) fields and fields' types from the data provider.

Live Data Module: Consumes live data from the provider and validates the raw data according to the schema contract.

Complex Event Processing Module: Works on a stream of data and filter events that match the required query. The query uses standard language SQL. Any matching result should be sent to the broker immediately.

Detailed Data Module: Provides an interface to query heterogeneous databases and files. It also generates dynamic reports of the results using the reports templates. This module provides a unified standard query interface language SQL.

Data Storage: Stores temp data between batch intervals. It's optimized for high-performance sequential operations.

Batch Module: Queries the cache according to the defined time interval and sends them to the broker.

Admin Module: Receives commands and direct queries from the core system and replies with the result.

C. *Data Flow in the Edge Centric Middleware*

Edge Centric Middleware supports three data flow modes:

- A defined event pattern can trigger sending data near real-time to the core system
- The core system requests detailed data on a specific subject. The request is fulfilled by the edge device which sends back a reply message.
- Data is collected from the provider, prepared, and then sent in batches to the core system

1) *Live Dataflow*

This is considered as the regular periodic scenario (Fig. 5) that is always tracking familiar patterns from within the data to register any possible anomalies.

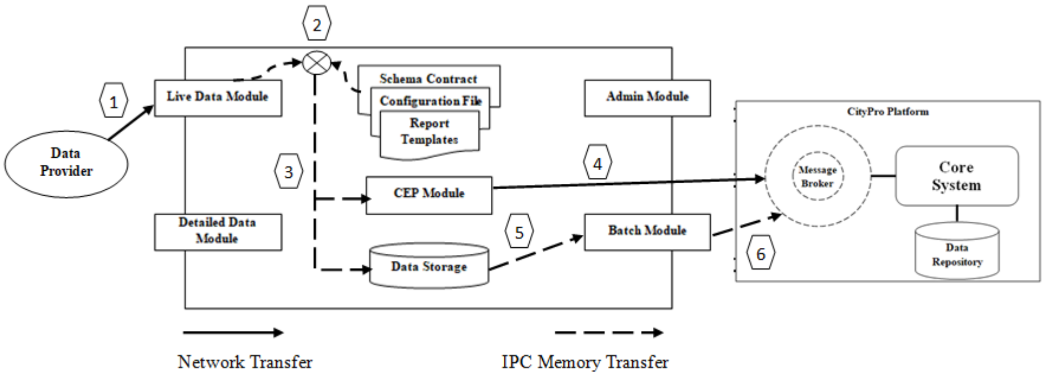


Fig. 5. Live Data Flow

1. Data is streamed from the data provider live over a network connection to the live data module which accepts data at a specific open TCP port.
2. Live Data Module uses the schema contract to validate and apply light computation on the incoming data whether it's as simple as attribute selection or ETL-like operations.
3. Live Data Module forwards processed data to the Complex Event Processing (CEP) module and to the cache storage in parallel at the same time via internal memory.
4. In the CEP module the stream of data is executed on the event pattern query. Upon any match, the event is forwarded at real-time to the broker via network connection. CEP publishes to a specific broker topic to avoid real-time alerts delays.
5. The batch module runs at custom time intervals, collects cached data, and sends them in a batch to the broker via a network connection

2) *On-Demand Data Flow*

This scenario (Fig. 6) is initiated whenever the platform requires immediate and detailed data, especially in the cases of alerts. In this case the platform directly connects to the provider's edge without the need of an intermediate broker.

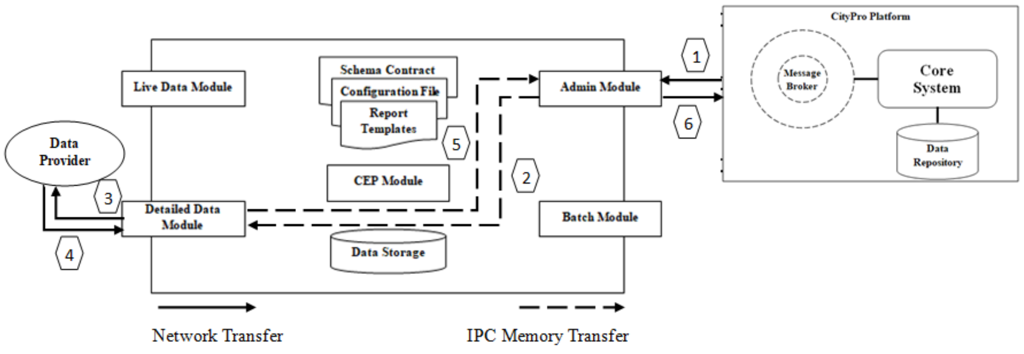


Fig. 6. On-Demand Data Flow

1. CityPro Core initiates this process by sending a request, which is a detailed query about a specific subject, to the admin module which awaits connections and commands.
2. The admin module initiates a new instance of the detailed data module with the proper report parameters such as the exact datastore query and the report template.
3. The detailed data module has the capability of querying different types of databases whether SQL or NoSQL. After querying the provider's database at rest and getting the result, the detailed data module will build the report and forward it to the admin module via inter-process communication.
4. The admin module will send back the report to the CityPro Core.

3) *Provider Live Data Source*

Two modes operate while getting live data from the provider. To achieve this, we studied two paradigms at the abstract level.

First Paradigm (Two-Tier Systems): We consider the database as the source of "live" data. So, we must detect and forward any data changes at the database level and forward the changes to a live data stream.

Second Paradigm (Three-Tier Systems): We can stream data live from the business logic layer or we can use the database layer in a similar way to the first case.

IV. Implementation and evaluation

To test the "Edge Centric Middleware" the inner workings and how the data flows we considered the case of Telecom Call Detail Records (CDR). We generated live phone calls and used Microsoft SQL Server as a database solution at the provider side. Our edge part of the middleware was deployed on a RaspberryPi [19] board and connected to the simulated database. The edge device detected alerting patterns and sent them to the broker, in addition to sending batches of data. We configured a Kafka broker with two topics and validated the data flow. Furthermore, we simulated a city core system panel to test the admin channel.

A. *Telecom Test Case*

Preparing a DataSet - Due to privacy concerns, there are no real data sets for telecom CDR. So, we generated random CDR records. The CDR schema includes ID, CALLING_NUM, CALLED_NUM, START_TIME, END_TIME, CALL_TYPE, CHARGE, and CALL_RESULT

We wrote a NodeJS [20] script to randomize the values while keeping the numbers in Lebanese format. This script keeps running emulating current phone calls that are taking place right now.

Database Engine and Notification Service - After generating call detail records (CDR) we consider Telecom as a data provider system. At the provider side we used Microsoft SQL Server [21] as the database solution. To establish live data from the telecom data provider system to the edge framework we implemented a .NET service that wraps an MS SQL Server feature called "query notifications". "Query Notifications" are best defined and documented as "query notifications that allow applications to be notified when data changes. This feature is particularly useful for applications that provide a cache of information from a database." [22] Using query notifications our .NET service streams any new data inserted in SQL Server to our edge framework. This stream is serialized using Apache Avro [23].

Edge Framework on RaspberryPi - The software stack for the edge is a cross-platform so we don't have a problem in selecting the hosting operating system. For the hardware, the edge software can scale from a small computer board to a rack of enterprise servers according to the load at each data provider. This makes it more efficient in any budget planning. For testing, we deployed it on a RaspberryPi [19] board (Fig. 7) with the following specifications:

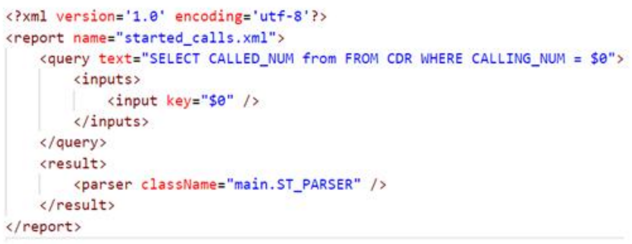
- Model 3 B+
- 1 GB RAM
- 32GB Storage
- Quad-core 64-bit processor clocked at 1.4GHz.
- 300Mbps Ethernet

Fig. 7. RaspberryPi 3 B+

Deploying and running the edge software framework on limited resources such as the RaspberryPi board proved the performance and the work that is done to optimize the computing footprint.

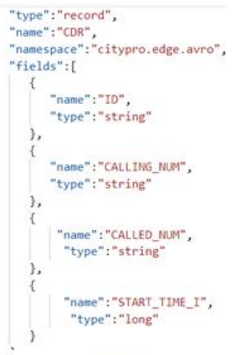
Kafka Broker - The broker of the "Edge Centric Middleware," was tested with Apache Kafka [11]. For the telecom (CDR) data case, we created two topics: one for normal data batches, and one for alerts. The batch component in the edge framework published messages to the normal topic while the CEP module published messages to the alert topic. This will guarantee fast delivery for alerts and then the infrastructure supporting each topic can be scaled and optimized accordingly.

Some of the implementations are shown in Fig. 8.



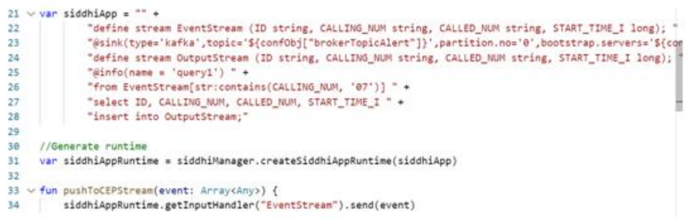
```
<?xml version='1.0' encoding='utf-8'?>
<report name="started_calls.xml">
    <query text="SELECT CALLED_NUM from FROM CDR WHERE CALLING_NUM = $0">
        <inputs>
            <input key="$0" />
        </inputs>
    </query>
    <result>
        <parser className="main.ST_PARSER" />
    </result>
</report>
```

Fig (a)



```
"type":"record",
"name":"CDR",
"namespace":"citypro.edge.avro",
"fields":[
    {
        "name":"ID",
        "type":"string"
    },
    {
        "name":"CALLING_NUM",
        "type":"string"
    },
    {
         "name":"CALLED_NUM",
          "type":"string"
    },
    {
          "name":"START_TIME_I",
           "type":"long"
    }
`
```

Fig (b)



```
var siddhiApp = "" +
        "define stream EventStream (ID string, CALLING_NUM string, CALLED_NUM string, START_TIME_I long); "
        "@sink(type='kafka',topic='${confObj["brokerTopicAlert"]}',partition.no='0',bootstrap.servers='${cor
        "define stream OutputStream (ID string, CALLING_NUM string, CALLED_NUM string, START_TIME_I long); "
        "@info(name = 'query1') " +
        "from EventStream[str:contains(CALLING_NUM, '07')] " +
        "select ID, CALLING_NUM, CALLED_NUM, START_TIME_I " +
        "insert into OutputStream;"

//Generate runtime
var siddhiAppRuntime = siddhiManager.createSiddhiAppRuntime(siddhiApp)

fun pushToCEPStream(event: Array<Any>) {
    siddhiAppRuntime.getInputHandler("EventStream").send(event)
```

Fig (c)

Fig. 8. Components Implementation: a) Report Template - b) Schema Contract - c) CEP Module

B. *Assessment and evaluation*

Although we didn't implement much security features, we consider our solution as a security and privacy enabler. For example, we deployed our framework on a separated and dedicated hardware where we can add physical tampering detection. In addition, only our certified edge devices can send messages to the broker. Furthermore, encryption on the network layer and on the device cache storage can be added.

V. CONCLUSION, OBSERVATION AND FUTURE WORK

Smart cities tackle development obstacles and improve the quality of life for citizens. CityPro system focuses on city protection by supporting the collaboration of existing operational systems. The need for a robust and standard middleware is critical for any smart city platform.

However, due to challenges such as continuous big data streams, heterogeneous systems, security, and privacy, in-addition to non-opensource software solutions there is a lack of such a middleware. This article proposes a new architecture for a smart city middleware using emerging edge computing trends and provides an open-source implementation for the proposed framework.

The "Edge Centric Middleware" consists of two parts: edge devices distributed along with the data providers' systems and a broker between edge devices and the "City Core System". We delegate some computation tasks to the distributed edge devices which provide a uniform and standard interface to the variety of existing systems. The edge devices follow the concept of black box to tackle privacy and security concerns. While the broker handles the scalability and availability of message queues from a distributed network of edge devices.

More metrics and experimental validations are needed. We had very limited time to develop a PoC (Proof of Concept) for this research. More experimentation should stress test big data flows.

The software implementation is based on standard and opensource technologies. Developing within an opensource community helps in boosting the pace of solving issues and adding features.

ACKNOWLEDGMENT

This work is supported and funded by the Department of research at the Lebanese University, Lebanon.

REFERENCES

[1] M. Dbouk, M. Mcheick and I. Sbeity, "CityPro: city-surveillance collaborative platform," Int. J. Big Data Intelligence, vol. 4, no. 3, 2017.

[2] W. Shi, J. Cao and Q. Zhang, "Edge Computing: Vision and Challenges," IEEE Internet of Things Journal, vol. 3, no. 5, 2016.

[3] Cisco, "Edge computing vs. fog computing: Definitions and enterprise uses," [Online]. Available: https://www.cisco.com/c/en/us/solutions/enterprise-networks/edge-computing.html.

[4] B. Rinner, M. Jovanovic and M. Quaritsch, "Embedded Middleware on Distributed Smart Cameras," in 2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07, 2007.

[5] F. J. Villanueva, M. J. Santofimia, D. Villa, J. Barba and J. C. López, "Civitas: The Smart City Middleware, from Sensors to Big Data," in 2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, 2013.

[6] A. d. M. D. Esposte, F. Kon, F. M. Costa and N. Lago, "InterSCity: A Scalable Microservice-based Open Source Platform for," in 6th International Conference on Smart Cities and Green ICT, 2017.

[7] L. Wang and R. Ranjan, "Processing Distributed Internet of Things Data in Clouds," IEEE Cloud Computing, vol. 2, no. 1, 2015.

[8] Z. Liu, F. Cretton, A. L. Calvé, N. Glassey, A. Cotting and F. Chapuis, "MUSYOP: Towards a Query Optimization for Heterogeneous Distributed Database," in International conference on Computing Technology and Information Management, Dubai, 2014.

[9] Apache Spark, [Online]. Available: https://spark.apache.org/.

[10] TIBCO Inc., "What is a Message Broker?," [Online]. Available: https://www.tibco.com/reference-center/what-is-a-message-broker.

[11] "Apache Kafka," [Online]. Available: https://kafka.apache.org/.

[12] "Apache ActiveMQ Artemis," [Online]. Available: https://activemq.apache.org/components/artemis/.

[13] A. Zookeeper. [Online]. Available: https://zookeeper.apache.org/.

[14] N.-L. Tran, S. Skhiri and E. Zim´nyi, "EQS: An Elastic and Scalable Message Queue for the Cloud," in 2011 IEEE Third International Conference on Cloud Computing Technology and Science, 2011.

[15] D. Luckham and B. Frasca, "Complex Event Processing in Distributed Systems," Stanford University, 1998.

[16] "Apache Flink," [Online]. Available: https://flink.apache.org/news/2016/04/06/cep-monitoring.html.

[17] "Siddhi," [Online]. Available: https://siddhi.io/.

[18] "Esper," [Online]. Available: http://www.espertech.com/esper/.

[19] "RaspberryPi," [Online]. Available: https://www.raspberrypi.org.

[20] "NodeJS," [Online]. Available: https://nodejs.org/en/.

[21] "Microsoft SQL Server," [Online]. Available: https://www.microsoft.com/en-us/sql-server/.

[22] Microsoft, "Query Notifications in SQL Server," [Online]. Available: https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/sql/query-notifications-in-sql-server.

[23] "Apache Avro," [Online]. Available: https://avro.apache.org/.