Publish/Subscribe Cloud Middleware for Real-Time Disease Surveillance

Silvino Neto, Márcia Valéria, Plínio Manoel, Felipe Ferraz
Recife Center for Advanced Studies and Systems (CESAR)
Recife – PE, Brazil
e-mail: {silvino.neto, marciavr.souza, ti.plinio}@gmail.com, fsf@cesar.org.br

Abstract - **This paper presents the design and implementation of a cloud-based middleware built on top of the Google Cloud Platform (PaaS), in order to exchange real-time information about outbreak notifications of global diseases in a system-level by using an extension of the HL7 Fast Healthcare Interoperability Resources (FHIR) specification to support statistical data based on the ICD-10 medical classification list. The proposed solution aims to allow healthcare organizations to register their systems to send and receive notifications, so the alerts are spread to all the subscribed systems using webhooks in a publish/subscribe fashion.**

Keywords - middleware; google cloud pub/sub; google cloud platform; FHIR

I. INTRODUCTION

There is an increasing demand for real-time monitoring of a broad variety of complex events. The processing of these information streams originated from multiple sources allows the early identification of threats and swift response. With the advent of modern communication technology, we are able to report incidences of disease outbreaks worldwide in a timely manner. Institutions such as the World Health Organization (WHO) and the Centers for Disease Control have been involved in the development of surveillance mechanisms that triggers alerts that support the decision-making process about how to respond to these incidents.

As results, there has been many successful experiences in using different forms of communication to exchange data related to surveillance and control of diseases, such as Short Message Service (SMS) [1][2], integration of device data capture [3] and system-level notifications [4].

Healthcare records are increasingly becoming digitized. In order to support system-level exchange of clinical data, a set of standards are required. The HL7 specification comprises a set of international standards to exchange clinical data between healthcare applications. In an attempt to improve its simplicity and extensibility, the HL7 introduced a new specification known as Fast Healthcare Interoperability Resources (FHIR). When compared with its predecessors, HL7 FHIR offers a whole new set of features, such as: support for multiple data formats: Extensible Markup Language (XML) and JavaScript Object Notation (JSON), extensible data model and a RESTful API.

This paper describes the Platform for Real-Time Verification of Epidemic Notification (PREVENT), a cloud-based message-oriented middleware in collaboration with the use of an extended instance of the FHIR specification to support statistical reports for disease surveillance in order to monitor and notify outbreak occurrences in real-time fashion. In our solution, we have developed our middleware application on top of the Google Cloud Platform, using the Google Cloud Pub/Sub, which is a many-to-many, asynchronous messaging service. Healthcare organizations may send and receive push notifications through the use of a registered webhook endpoint that can accept POST requests over HTTPS.

This paper is further structured as follows: In Section 2, we discuss the foundations for this paper. In Section 3, we present the architectural approaches proposed for the middleware and some of the design choices implemented. In Section 4, we explain our evaluation approach and present the results obtained. In Section 5, we discuss related work and finally, Section 6 presents our conclusions and possible future work.

II. FOUNDATIONS

In this Section, this paper presents key concepts that served as basis for the development of this work.

A. WHO

The WHO is a specialized worldwide health agency subordinated to the United Nations (UN) that, according to its constitution [5], one of its main objectives is the development and improvement of the health of people to the highest possible levels. Still, according to the WHO constitution, it is responsible for coordinating efforts to control and prevent outbreaks and diseases. The WHO supervises the implementation of the International Health Regulations and publishes a series of medical classifications, including the International Statistical Classification of Diseases and Related Health Problems (ICD) [6]. The ICD is designed to promote international comparability in the collection, processing, classification, and presentation of mortality and morbidity statistics.

According to the International Health Regulations (IHR), an international legal instrument that is compulsory in 196 countries and in all the WHO member states, its goal is to assist the international community in the prevention and response to potential cross-border public health risks. The IHR requires that countries report disease outbreaks and public health events to WHO [7].

The present work discusses a system platform that allows national health organizations, members of the United Nations, hospitals or healthcare agencies, regardless the location, to subscribe their applications to send and receive real-time notifications for disease surveillance. Thus, they contribute with the propagation of the notified information, so it can achieve the widest possible reach through the use of a cloud-based platform. Therefore, countries and healthcare organizations may act promptly under the emergency and disaster risk management protocol to prevent, prepare, respond and recover from incidents due to any danger that might represent a threat to human health security.

B. HL7 and FHIR

To reach its goal, this work analyzes a set of international standards that provide a framework for the integration and share of clinical and administrative data between systems and electronic devices dedicated to health care. The HL7 [8] was created in 1987 and has been maintained by Health Level Seven International, a nonprofit international organization that supports and promotes the development of international interoperability standards in healthcare systems.

The second version of HL7, an *ad hoc* approach to integrate various fields in health care, hospitals, clinics and administrative applications, has become a widely used standard, adopted and supported by most healthcare application vendors in North America [9]. Despite HL7 v2 wide acceptance, the limitations of the *ad hoc* approach have not allowed significant high scale use in larger multiplatform environments. Another downside observed on HL7 v2 is the lack of a formal data model that can unify concepts and interfaces for message transmission. HL7 v3 emerged as a response to all the problems recognized on the previous version. However, it was heavily criticized by the industry for being inconsistent, overly complex and infeasible to implement in real life systems. For a while, it appeared as if interoperability initiatives for health care had lost momentum.

Hence, FHIR was created with the objective of improving HL7 messaging standards and addressing some of the issues identified on the previous specifications. There have been discussions towards a new approach for data exchange in health care. This approach provides a Representational State Transfer (REST) interface, which is a very simple and lightweight interoperable alternative for system integration. REST-based architectures are known for its scalability, user-perceived performance and ease of implementation approach that provides a fast data transmission pattern mostly using the HTTP protocol [10]. Resource interoperability allows information to be readily distributed and provides an alternative to document-centric approaches by directly exposing data elements as services.

FHIR uses syntax based on XML or JSON, simplifying the system-level communication. It also offers support for an extensible data model, allowing applications to enhance its data structures using FHIR extensibility mechanism. The features mentioned on this paper were decisive factors for the adoption of FHIR on the development of PREVENT.

C. Cloud Computing and Scalability

According to A. T. Velte et al. “In essence, cloud computing is a construct that allows you to access applications that actually reside on a location other than your computer or other Internet-connected device; most often, this will be a distant data center” [11]. This is a constant reality for the majority of the Internet users on a daily basis. Among the benefits of cloud computing cited by [11], there are simplicity, knowledgeable vendors, more internal resources, security, and scalability.

Scalability is seen as a fundamental feature of cloud computing. It appears as if computational resources are infinite and end users easily notice the increase of performance of used resources in a cloud-based platform. Scalability is not restricted to expanding resource capacity, being scalable is to increase the capacity of operations in an efficient and adequate manner, maintaining the quality of service [12]. In the literature, it is possible to identify two dimensions of scalability: vertical and horizontal. Vertical scalability refers to the improvement of hardware capabilities by incrementing existing nodes individually. Horizontal scalability refers to the addition of extra hardware nodes to the current solution in a way that it can be possible to distribute application requests between these machines [13]. In the context of this work, PREVENT is designed and implemented with focus on horizontal scalability. In order to sustain a large volume of time-constrained notifications and to leverage the platform overall scalability, PREVENT is deployed in a cloud-based platform.

D. PREVENT and Complex Event Processing (CEP)

CEP is a new technology to extract information from distributed message-based systems. This technology allows users of a system to specify the information that is of interest to them. It can be low-level network processing data or high-level enterprise management intelligence, depending on the role and point of view of individual users. It operates not only on sets of events but also on relationships between events [14].

In order to respond in a suitable manner, it is fundamental to use technology that supports the construction and management of event-oriented information systems, and is also able to perform real-time data analysis. CEP consists in processing various events in order to identify their significance within a cloud of information [15]. CEP involves rules to aggregate, filter and match low-level events, coupled with actions to generate new, higher-level events from those events [16].

PREVENT has its own complex event processing unit, namely, PREVENT CEP Engine (PCEPE). PREVENT randomly receives data messages derived from healthcare applications subscribed as data providers or data sources, once data messages have been received, they are delegated to PREVENT internal complex data processing unit (PCEPE) that identifies the source and semantics of the data received, extracting relevant information.

After the information extraction phase previously described, the data collected goes through a second-phase analysis that intends to identify if the events notified at that time indicate a warning situation. As an example, a significant volume of reports of a certain disease from a specific geographically delimited area, points to a relevant situation of alert.

Finally, PREVENT only delivers relevant notifications to each subscribed message receiver. Figure 1 presents the processing flow for data received from health care organizations in the PREVENT platform.

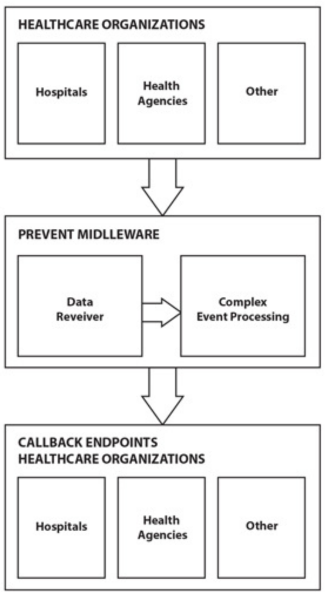


Figure 1. CEP diagram in PREVENT.

An event processing approach is ideal for applications concerned with the constant delivery of responses [15]. For sensitive information that requires a high level of consistency, it is extremely important that PREVENT responses are cohesive. This way, having an internal unit for the processing of the received events is required for the proper functioning of the PREVENT platform.

III. MIDDLEWARE

In this Section, this paper explores the proposed architecture and the workflow of events implemented in our middleware platform. Every step below is described as part of the process designed to perform the management of the subscribed applications, the process of data analysis and the delivery of notifications in the subscription topic:

- Healthcare applications may subscribe to our middleware platform in order to send and receive notifications to/from other systems;
- PREVENT will register the healthcare application in a subscription topic, and reply with an assigned application ID;
- Healthcare applications may now send notifications to PREVENT subscription topic;
- PREVENT will perform a real-time analysis of the data received, and publish notifications that match the specified criteria;
- Healthcare applications may now receive asynchronous notifications sent by other applications, delivered by PREVENT in a push request.

A. *System Architecture*

The system architecture designed for PREVENT is illustrated in Figure 2. In this diagram, PREVENT is organized into the Google App Engine [17] which is a hosting environment for web-based cloud applications. It is part of the Google Cloud Platform, as well as the Google Cloud Datastore, which is a schemaless NoSQL scalable datastore, and the Google Cloud Pub/Sub, an asynchronous messaging framework. Both framework platforms are used for data persistence and messaging (publish/subscribe pattern) services. It is important to mention that this middleware was developed in the Java programming language, using the Java Servlet API, a standard to implement applications hosted on Web servers under the Java platform. Despite being implemented on top of the Java platform, PREVENT is a completely agnostic technology, it uses interoperable standards such as HL7 FHIR, REST and JSON, and it can be integrated to any healthcare application, regardless the implementation language.

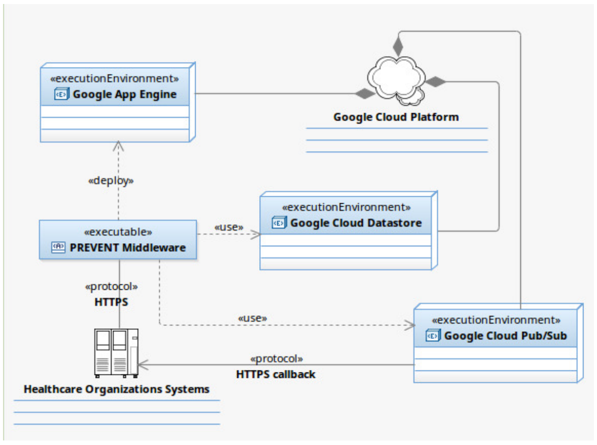


Figure 2. PREVENT Deployment Diagram.

Persistence is used on PREVENT to store data related to the subscribed healthcare applications and the statistical reports extracted from notifications received. The information stored by PREVENT at the Google Cloud Datastore is relevant not only for the delivery of real-time notifications, but it may also be useful for audit and access control policy and procedures. The data stored is replicated across multiple datacenters using a highly available platform based on the Paxos algorithm, which is a family of protocols for solving consensus in distributed environments [21].

PREVENT is designed to be a Message-oriented middleware (MOM) platform. It is implemented by using the publish/subscribe pattern, since it requires many-to-many communication. The Google Cloud Pub/Sub [18] platform supports two delivery strategies: push and pull delivery. In the push delivery, the server sends a request to the subscriber application at a previously informed endpoint URL for every message notification. In the pull delivery, the subscribed

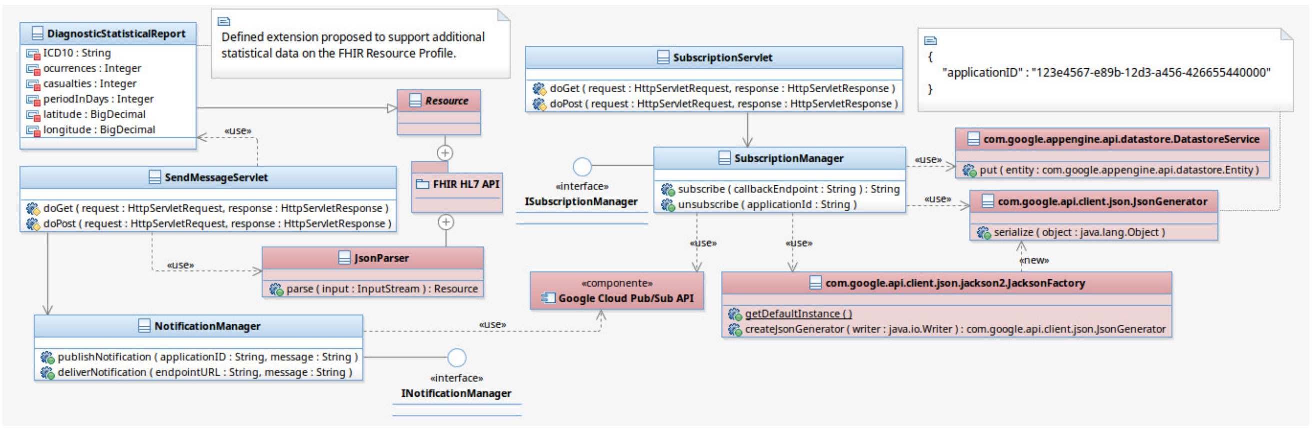

DiagnosticStatisticalReport
ICD10 : String
ocurrences : Integer
casualties : Integer
periodInDays : Integer
latitude : BigDecimal
longitude : BigDecimal
Defined extension proposed to support additional statistical data on the FHIR Resource Profile.
Resource
«use»
SendMessageServlet
doGet (request : HttpServletRequest, response : HttpServletResponse)
doPost (request : HttpServletRequest, response : HttpServletResponse)
FHIR HL7 API
«use»
JsonParser
parse (input : InputStream) : Resource
NotificationManager
publishNotification (applicationID : String, message : String)
deliverNotification (endpointURL : String, message : String)
«use»
«interface»
INotificationManager
SubscriptionServlet
doGet (request : HttpServletRequest, response : HttpServletResponse)
doPost (request : HttpServletRequest, response : HttpServletResponse)
{
"applicationID" : "123e4567-e89b-12d3-a456-426655440000"
}
«interface»
ISubscriptionManager
SubscriptionManager
subscribe (callbackEndpoint : String) : String
unsubscribe (applicationId : String)
«use»
com.google.appengine.api.datastore.DatastoreService
put (entity : com.google.appengine.api.datastore.Entity)
«use»
com.google.api.client.json.JsonGenerator
serialize (object : java.lang.Object)
«use»
«use»
«new»
«componente»
Google Cloud Pub/Sub API
com.google.api.client.json.jackson2.JacksonFactory
getDefaultInstance ( )
createJsonGenerator (writer : java.io.Writer) : com.google.api.client.json.JsonGenerator

Figure 3. Subscription & Notification Scenario Class Diagram.

application has to explicitly invoke the API pull method, which requests the delivery of any existing message available in the subscription topic to the invoker. In our middleware implementation, we have chosen the push delivery strategy, based on the following criteria:

- Reduced network traffic;
- Reduced latency;
- Restrict/eliminate impacts of adaptation on healthcare applications (No need for message handling and flow control).

However, it's important to note that Google Cloud Pub/Sub platform is still a Beta release, so few limitations may be applied. Currently, the only supported endpoint URL for push delivery is an HTTPS server that can accept Webhook delivery. For this reason, we designed an internal HTTPS endpoint message listener that can be used as a proxy by healthcare applications in order to receive notifications that are subsequently forwarded to a regular HTTP endpoint URL.

B. *Subscription Request*

Healthcare organizations that want their application to send and receive notifications from our middleware should send subscription requests to PREVENT informing a single parameter named callback endpoint. The value of this parameter should correspond to a regular HTTP or HTTPS URL that will be invoked for notification delivery. As a response, PREVENT will reply with a unique application ID assigned for the request in JSON format as illustrated in Figure 3. As exhibited in the class diagram (See Figure 3), our middleware platform receives subscription requests through a Java servlet. The requests are subsequently assigned to the SubscriptionManager class, responsible for interacting with the Google Cloud Pub/Sub and Google Datastore APIs in order to both create a new subscription and store application data.

Once successfully registered, healthcare applications are allowed to send notifications to our middleware by informing its unique application ID with an extended HL7 FHIR message instance in JSON format.

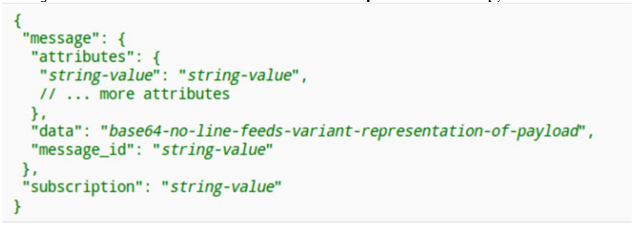
C. *Publishing Notifications*

Previously registered systems should be able to publish notifications to all the subscribed applications. In order to do so, subscribed applications should always inform their application ID with an extended version of the HL7 FHIR message, as shown on Figure 3. As demonstrated in the class diagram (See Figure 3), there is another servlet class to receive requests for notification dispatch. This servlet class is expecting an HL7 FHIR message in JSON format. After the message is successfully parsed into its Java object representation, it will be dispatched to the *NotificationManager* class, responsible for the delivery of the message by invoking a method on the Google Cloud Pub/Sub API to add the new message to the subscription topic, making it available for delivery.

As mentioned earlier, HL7 FHIR provides a flexible mechanism for the inclusion of additional information into the FHIR data model. The class *DiagnosticStatiscalReport* shown in the diagram above (See Figure 3) is an example of an extension implemented on top of the FHIR specification. According to the FHIR specification, in order to use an extension, we must follow a three-step process, as defined in [19].

D. *Delivery of Notifications*

Notifications added to the subscription topic will be asynchronously sent to the registered endpoints for every subscribed application, in a multithread context. Messages are dispatched in the body of an HTTP push request. The body is a JSON data structure as depicted on Figure 4.



```
{
 "message": {
  "attributes": {
   "string-value": "string-value",
   // ... more attributes
  },
  "data": "base64-no-line-feeds-variant-representation-of-payload",
  "message_id": "string-value"
 },
 "subscription": "string-value"
}
```

Figure 4. Google Cloud Pub/Sub JSON message data structure.

The data attribute contained into the message data type structure holds the HL7 FHIR message encoded in Base64 format. Once the FHIR message is retrieved, in order to restore it back to its original form, healthcare applications must decode it. Furthermore, to indicate the successful delivery of the message received and avoid duplicate deliveries, healthcare applications must return one of the following HTTP status code: 200, 201, 203, 204 or 102. Otherwise, the Google Cloud Pub/Sub retries sending the message indefinitely to assure its delivery, using an exponential backoff algorithm in order to avoid network congestion [23]. The use of an exponential backoff algorithm is a feature offered by the messaging platform, in order to guarantee message delivery in case of message destination is unreachable or unavailable. Therefore, no message expiration or timeout is applicable.

It is expected to configure push endpoints with SSL certificates, so data integrity is guaranteed since all messages sent to them are encrypted over HTTPS. However, healthcare applications that do not provide an HTTPS webhook enabled endpoint, may still receive notifications using a regular HTTP endpoint URL. As already mentioned, PREVENT offers an internal message listener component that acts as a proxy for notification of delivery. During the subscription request processing, PREVENT will automatically assign an internal push endpoint for healthcare applications that informed regular HTTP URLs. Therefore, for every notification to be delivered, PREVENT hands it over to its internal message listener component that subsequently dispatches the notifications to their corresponding destinations. In order to manage HTTP connections efficiently, PREVENT uses the Apache HttpComponents, which is a toolset of low-level Java components APIs focused on HTTP [20]. Apache HttpComponents is designed to be a flexible framework, supporting blocking, non-blocking and event driven I/O models. In our middleware implementation, we selected the asynchronous non-blocking I/O model, in order to be able to handle thousands of simultaneous HTTP connections in an efficient manner.

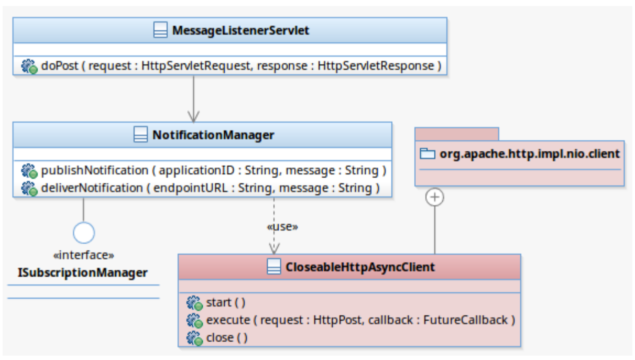

MessageListenerServlet
doPost (request : HttpServletRequest, response : HttpServletResponse)
NotificationManager
publishNotification (applicationID : String, message : String)
deliverNotification (endpointURL : String, message : String)
org.apache.http.impl.nio.client
«use»
«interface»
ISubscriptionManager
CloseableHttpAsyncClient
start ( )
execute (request : HttpPost, callback : FutureCallback)
close ( )

Figure 5. Message Listener Component Class Diagram

The class diagram shown on Figure 5 presents a short representation of the elements discussed on the previous paragraph.

IV. Evaluation

We have developed a proof-of-concept implementation in order to evaluate the middleware. Our goal is to show how effective and responsive a cloud-based middleware platform for real-time surveillance can be. In accordance with the criteria established, we use a few metrics to measure the efficiency and performance of the middleware platform. In our experiments, we evaluate the middleware by using a set of simulation tools. The test environment set up for the evaluation is composed by a cloud-based instance of the middleware distributed into the Google Cloud Platform, a set of 50 callback endpoints, and an Intel i5 2.60GHz 6GB RAM Linux workstation. Each callback endpoint simulates a subscribed healthcare application, previously registered on the middleware. In order to act as an enlisted application, we have implemented a simple Java servlet class and a PHP file that basically returns an HTTP status code of 200 (OK) to acknowledge the successful reception of notifications delivered. The callback endpoints are deployed into two separate cloud platforms: Digital Ocean [22] and Google Cloud Platform. The tests are divided in two different scenarios. The first test scenario is executed at the execution environment of callback endpoints. On the second test scenario, we use a local computer workstation to perform a stress test. Both test scenarios will be conducted using a set of preconfigured simulation tools. In this evaluation, the following metrics are gathered and further analyzed:

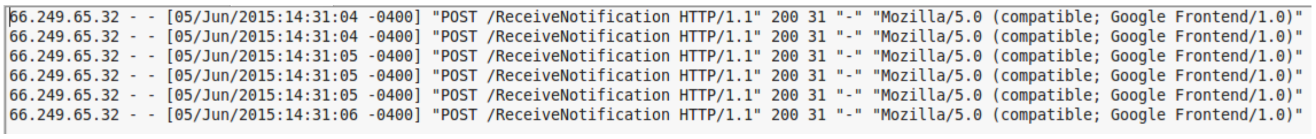
- Throughput: measured by the number of delivered notifications per second;
- Error Ratio: measured by the number of requests failed or rejected by the middleware;
- Message Loss Ratio: measured by the proportion between the number of lost messages and the total number of messages delivered;
- Message Delivery Time: measured by the time taken to a notification request to be sent to the middleware and received by the subscribed healthcare applications.

A. *Simulation Description*

We perform the dispatch of messages using JMeter, a Java based testing tool, to send multiple HTTP requests.

On the first test scenario, our test suite is configured to send one message request per second, limited to 60 messages to be delivered to 50 subscribed callback endpoints. Messages used in the test are defined on HL7 FHIR JSON format. Each message size is approximately 1 KB. This scenario seeks to evaluate the performance of the middleware about the delivery ratio and message delivery time. In order to measure the total message delivery time, we must consider that the actual delivery of each message sent to the middleware occurs in an asynchronous manner. Therefore, time tracking has to be split into two separate variables:

- T^1 = Amount of time it takes for message requests sent to the middleware to be responded;



```
66.249.65.32 - - [05/Jun/2015:14:31:04 -0400] "POST /ReceiveNotification HTTP/1.1" 200 31 "-" "Mozilla/5.0 (compatible; Google Frontend/1.0)"
66.249.65.32 - - [05/Jun/2015:14:31:04 -0400] "POST /ReceiveNotification HTTP/1.1" 200 31 "-" "Mozilla/5.0 (compatible; Google Frontend/1.0)"
66.249.65.32 - - [05/Jun/2015:14:31:05 -0400] "POST /ReceiveNotification HTTP/1.1" 200 31 "-" "Mozilla/5.0 (compatible; Google Frontend/1.0)"
66.249.65.32 - - [05/Jun/2015:14:31:05 -0400] "POST /ReceiveNotification HTTP/1.1" 200 31 "-" "Mozilla/5.0 (compatible; Google Frontend/1.0)"
66.249.65.32 - - [05/Jun/2015:14:31:05 -0400] "POST /ReceiveNotification HTTP/1.1" 200 31 "-" "Mozilla/5.0 (compatible; Google Frontend/1.0)"
66.249.65.32 - - [05/Jun/2015:14:31:06 -0400] "POST /ReceiveNotification HTTP/1.1" 200 31 "-" "Mozilla/5.0 (compatible; Google Frontend/1.0)"
```

Figure 6. HTTP messages delivered

- T^2 = The average of the amount of time it takes for the middleware to send notifications received to all of the subscribed callback endpoints and get their message acknowledged;
- This way, Total Message Delivery Time = $T^1 + T^2$.

In order to obtain the values required to calculate both variables T^1 and T^2, we use both JMeter Test Results Report and Apache log4j, which is a Java-based logging utility.

The second test scenario is divided into 4 different test cases. Each test case has different settings, related to the number of messages that are expected to be processed. The number of concurrent threads (publishers) configured for each test case are limited to 10, 20, 60 and 100, respectively. Every thread is expected to simultaneously send one message per second. Every message received needs to be processed and delivered to 50 subscribed endpoints. Therefore, we expect a total of 14500 messages delivered after all test cases are completed. To evaluate message delivery to all of the subscribed callback endpoints, we created a shell script using AWK, which is a data-driven scripting language used in Unix-like operating systems, to extract and parse NGINX and Apache HTTP Server access logs in order to quantify the number of successfully received requests based on a pre-determined pattern as shown on Figure 6.

The results obtained will be summarized and compared to the total amount of messages sent. As a result, we expect to evaluate the middleware performance under heavy load, collecting and analyzing metrics like the throughput and message loss ratio.

B. Results

The evaluation shows that a cloud-based middleware for real-time surveillance works reliably and efficiently to report critical events in a timely manner. As illustrated on Figure 7, from a total of 14500 messages sent to the middleware platform, we reported 356 failed or rejected requests and 135 messages that have not been acknowledged as delivered. It corresponds respectively to 2.45% and 0.93% of the total amount of messages processed. The results obtained could be even better, since we reported a large concentration of failed requests at the end of the last test case, due to quota limits exceeded.

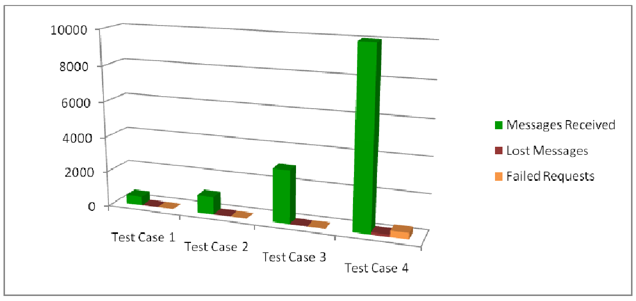


Figure 7. Message Delivery 3D Grouped Column Chart

In the same stress test scenario, we collected a set of performance related metrics, as shown in Table 1. The middleware comfortably supported the intense load of requests, maintaining good performance levels. Based on the results obtained, a few observations can be made about this platform. Throughput and performance are positively impacted under heavier load of concurrent requests. We believe that it happens due to reconfiguration algorithms implemented by the cloud platform, scaling to uphold the increasing volume of requests. However, we have also observed that the reliability of message delivery is negatively impacted under these circumstances.

TABLE I. PERFORMANCE RELATED METRICS

Number of Samples	Median (ms)	Throughput	kB/sec	Lost Messages	Failed Requests
500	691	48,4/sec	474,75	0	0
1000	394	55,2/sec	949,5	0	0
3000	293	103,4/sec	2848,5	3	0
10000	204	177,7/sec	9495,0	132	356

In the test scenario executed at a registered callback endpoint environment, we collected and measured the total amount of time for every notification to be delivered to all of the subscribed endpoints, as described on the previous subsection. Figure 8 presents the total message delivery time measured at specific intervals during the test execution. The x-Axis represents time (HH:MM:SS) intervals at which messages were delivered, while the y-Axis represents the total message delivery time in milliseconds for each notification sent during the test.

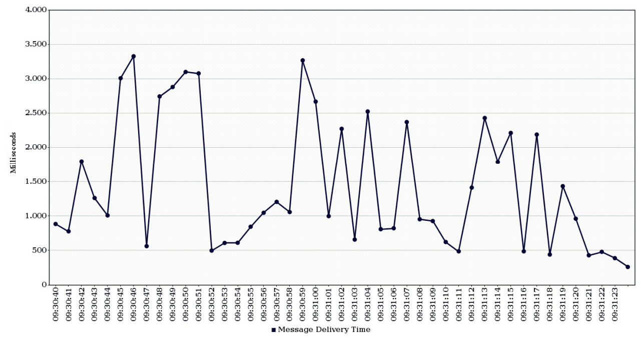


Figure 8. Total Message Delivery Time

Based on a string of experiments, we conclude that the differences observed in the results, exhibited on Figure 8, occur as a consequence of the following events: Network jittering, resource availability, and concurrency. The simulation results show that the performance provided by the platform, in terms of delivery ratio, throughput and timeliness is suitable to the middleware purpose. This is a consequence of the use of an asynchronous messaging approach and a cloud-based platform, capable of scaling and performing according to the restrictions imposed. Therefore, it is possible to offer a high degree of scalability using a publish/subscribe middleware for real-time disease surveillance. Even for larger applications scenarios, where the number of subscribed healthcare applications is considerably higher, or where the messages exchanged are in a higher number, we believe that it is possible to scale in order to support the increase of load. Further experiments have shown that when the number of concurrent messages is increased, the middleware's throughput is higher, while the performance is stable. However, with a higher amount of messages exchanged, we have observed an increase in the number of lost or unacknowledged messages.

V. RELATED WORK

Much of the related work has been covered in the previous Sections of this paper. In this Section, we revisit some of the topics discussed and present them in a summarized view.

Healthcare Interoperability. As mentioned on Section 2, subsection B, multiple approaches have been attempted in order to promote interoperability initiatives for healthcare systems, as demonstrated on both [3] and [24]. Recent work has been developed in order to offer a powerful and extensible standard specification for healthcare system-level integration, namely FHIR [9]. In our work, we have slightly extended the FHIR data model in order to include statistical information to be further processed by a CEP unit.

Disease Surveillance. This is an emerging field of research that has been achieving significant success in the early detection and report of disease outbreaks at regional scale. The DHIS 2 project as described in [1] uses Java enabled phones to send health related data using SMS. This is an example of how a low-cost disease surveillance mechanism can be helpful in the prevention or mitigation of occurrences, especially in developing countries [2]. Another successful example can be found in India and Sri Lanka, as described in the work of Waidyanatha *et al* [4]. The T-Cube project has been developed in order to detect emerging patterns of potentially epidemic events based on the analysis of digitized clinical health records. On our work, we use a similar approach, as previously described.

The ESS project developed in Sweden is an Event-based Surveillance System that uses records of telephone calls to the Swedish National Health Service, in order to monitor unusual patterns [25]. Statistical analysis is performed over collected data to calculate deviation limits. In an attempt to process larger datasets, Santos and Bernardino presented a system architecture for near real-time detection of epidemic outbreak at global scale using on-line analytical processing (OLAP) techniques [26].

VI. CONCLUSIONS

This paper has presented a middleware platform responsible for receiving and interpreting data informed by healthcare organizations, and based on the results obtained through data analysis, the middleware publishes real-time notifications to all healthcare applications subscribed to this platform.

There is an increasing need for timely delivery of messages and notifications, in very large user base platforms. In this context, it is extremely important to develop a platform capable of scaling to sustain the expected levels of performance and throughput under growing demand. PREVENT uses the FHIR specification in order to exchange system-level messages, presenting a market-friendly environment for real-time integration of applications. FHIR is currently published as a Draft Standard for Trial Use (DSTU), hence we hope that this work may serve as a contribution on the promotion of interoperability initiatives, and a step towards the development of an international disease surveillance platform.

We believe that several factors combined make PREVENT a scalable and efficient platform:

- An asynchronous event-based approach for message processing, reducing network contention and the number of threads needed to process the same workload;
- A network-efficient non-blocking I/O communication model for HTTP connections;
- And a cloud hosting infrastructure.

The results obtained from our experiments demonstrate that a cloud-based platform using the publish/subscribe pattern for real-time notifications represents an appropriate choice, in order to assure time-constrained delivery of mission-critical data.

The contribution of this paper is a first step to enabling the use of the FHIR specification for healthcare system integration in order to support a global system-level outbreak warning platform. Our ongoing research aims to perform experiments using heterogeneous environments and datasets, in order to present richer interoperable scenarios.

As future works, we plan to implement several extensions on our middleware platform in order to support functionalities like: security using OAuth; big data analytics for prediction using information extracted from notifications received; support for legacy and multi-format messages (HL7 v2, short message services, etc.) using a message adapter layer for data transformation, and multiprotocol integration using TCP, HTTP and HTTPS.

References

[1] L. Pascoe, J. Lungo, J. Kaasbøll, and I. Koleleni, “Collecting Integrated Disease Surveillance and Response Data through Mobile Phones,” IST-Africa Conference Proceedings, 2012, pp. 1-6.

[2] C. Déglise, L.S. Suggs, and P. Odermatt, “SMS for disease control in developing countries: a systematic review of mobile health applications,” Journal of Telemedicine and Telecare, 2012, v. 18, n.5, pp. 273-281.

[3] T. Tran, H-S. Kim, and H. Cho, “A Development of HL7 Middleware for Medical Device Communication,” Proc. 5th ACIS Int. Conf. Softw. Eng. Res., Manage. Appl., 2007, pp. 485-492.

[4] N. Waidyanatha et al., "T-Cube Web Interface as a tool for detecting disease outbreaks in real-time: A pilot in India and Sri Lanka," IEEE RIVF International Conference on Computing and Communication Technologies, Research, Innovation, and Vision for the Future (RIVF), 2010, pp. 1-4, doi: 10.1109/RIVF.2010.5633019.

[5] “Constitution of the World Health Organization”, 2005. [Online]. Available: http://goo.gl/2vvJBS. [Accessed: 30-May-2015].

[6] “International Statistical Classification of Diseases and Related Problems”, 2010. [Online]. Available: http://goo.gl/FCp5Js. [Accessed: 30-May-2015].

[7] “International Health Regulations”, 2005 Second Edition. [Online]. Available: http://goo.gl/m8Rajk. [Accessed: 30-May-2015].

[8] “Health Level Seven International – Homepage”, 2015. [Online]. Available: http://www.hl7.org/. [Accessed: 31-May-2015].

[9] D. Bender and K. Sartipi, "HL7 FHIR: An Agile and RESTful approach to healthcare information Exchange,” Computer-Based Medical Systems (CBMS), IEEE 26th International Symposium, 2013, pp. 326-331.

[10] R. T. Fielding, “Architectural Styles and the Design of Network-based Software Architectures,” Doctoral dissertation, University of California, Irvine, 2000.

[11] T. Velte, A. Velte, and R. Elsenpeter, “Cloud computing, a practical approach.” McGraw-Hill, Inc., 2009.

[12] P. Jogakekar and M. Woodside, “Evaluating the scalability of distributed systems,” IEEE Transactions on Parallel and Distributed Systems, 2000, v. 11, n. 6, pp. 589-603.

[13] D.F. Garcia, R. Garcia, J. Entrialgo, J. Garcia, and M. Garcia, “Experimental evaluation of horizontal and vertical scalability of cluster-based application servers for transactional workloads,” 8th International Conference on Applied Informatics and Communications, 2008, pp. 29-34.

[14] D. C. Luckham and B. Frasca, “Complex event processing in distributed systems,” Computer Systems Laboratory Technical Report CSL-TR-98-754, Stanford University, Stanford, v. 28, 1998.

[15] V. Vaidehi, R. Bhargavi, K. Ganapathy, and C.S. Hemalatha, "Multi-sensor based in-home health monitoring using Complex Event Processing," International Conference on Recent Trends In Information Technology (ICRTIT), 2012, pp. 570-575.

[16] D. Robins, "Complex event processing," Second International Workshop on Education Technology and Computer Science, Wuhan, 2010. [Online]. Available: http://goo.gl/jLROpc . [Accessed: 30-May-2015].

[17] “Google App Engine”. [Online]. Available: https://cloud.google.com/appengine/docs/. [Accessed: 13-June-2015].

[18] “Google Cloud Pub/Sub”. [Online]. Available: https://cloud.google.com/pubsub/docs/. [Accessed: 13-June-2015].

[19] “FHIR Specification Home Page- FHIR v. 0.0.82”, 2015. [Online]. Available: http://goo.gl/RDwuPm. [Accessed: 31-May-2015].

[20] “Apache HttpComponents.” [Online]. Available: https://hc.apache.org/. [Accessed: 13-June-2015].

[21] L. Lamport, “The part-time parliament,” ACM Transactions on Computer Systems (TOCS) Journal, 1998, v. 16, n. 2, pp. 133-169.

[22] “Digital Ocean” [Online]. Available: https://www.digitalocean.com/. [Accessed: 13-September-2015].

[23] Byung-Jae Kwak, Nah-Oak Song, and Miller, M.E., “Analysis of the stability and performance of exponential backoff,” IEEE Wireless Communications and Networking, 2003, v.3, pp. 1754-1759, doi: 10.1109/WCNC.2003.1200652.

[24] Fabio Vitali, Alessandro Amoroso, Marco Rocetti, and Gustavo Marfia, “RESTful Services for an Innovative E-Health Infrastructure: A Real Case Study,” IEEE 16th International Conference on e-Health Networking, Applications and Services, 2014, pp. 188-193.

[25] Deleer Barazanji and Pär Bjelkmar, “System for Surveillance and Investigation of Disease Outbreaks," 23rd International Conference on World Wide Web Pages, 2014, pp. 667-668.

[26] Ricardo Jorge Santos and Jorge Bernardino, “Global Epidemiological Outbreak Surveillance System Architecture,” 10th International Database Engineering and Applications Symposium, 2006, pp. 281-284.