Event Notification System

Reynaldo Burgos Torres
Master of Engineering in Computer Engineering
Prof. Othoniel Rodríguez, Ph.D.
Electrical and Computer Engineering & Computer Science Department
Polytechnic University of Puerto Rico

Abstract — *Today's enterprises have a huge amount of automatic processes and systems performing a huge amount of tasks, which are very important to the business success and to support normal operations. Having an Event Notification System (ENS) where these events become available for other systems consumption, end-users as email notifications, and persisted for later analysis or system audit, could improve the business and make the business more productive. ENS is a middleware system that enables other systems to post notification messages for the user community or other interested services regarding vApp deployments, site outages, network connectivity issues, test content management, and any other later identified event of interest for the optimal function of the team or operation. Also improves business with providing notification faster so that long time tasks can notify other processes or people interested of whether the operation has been completed or an intervention needs to be done. This article covers the design and implementation of the ENS.*

Key Terms — *Message Broker, Micro Services, Publish-Subscribe.*

INTRODUCTION

Event Notification System (ENS) is as middleware system that enables other services to post events for the user community or other interested services regarding vApp deployments, site outages, network connectivity issues, test content management, and any other later identified event of interest for the optimal function of the team or operation that we need to notify. These messages are persisted for later analysis or system audit. The system also includes a message broker so we can add consumers to react to the event notifications. Some consumer's examples could be Graphical User Interface that want to show live events to users, and or disconnected services performing tasks based on these events.

The ENS is based on four main components Event Notification Publisher (ENP), Message Broker (MB), Event Notification Mailer (ENM), and the Event Notification Logger (ENL). The ENP component provides a REST API interface so that any service or software component can publish events. That way we provide a simple interface that can be consumed by almost any language (e.g. C++, Java, C#, Python) as all of these languages provide support for REST API consumption.

The ENP is responsible for receiving the event message, persisting it and deliver it to the MB. MB component is responsible for receiving ENP messages and provides interfaces for consumers to receive the messages. ENM is a consumer to the MB and is mainly responsible for sending an email for those consumed message to the subscribed users. ENL logs information about consumed and acknowledged events. This component persist this data so it can be later used for audit and event traceability.

REQUIREMENTS SPECIFICATION

Based on the business needs, and stakeholders discussion we have outline functional and nonfunctional requirements to make sure we meet all business needs.

Functional Requirements

- The service or software component shall be able to publish an event notification using the REST API.
- The event notification shall be persisted.
- The event notification shall be sent by email to subscribed users.

- The event notification shall provide a timestamp so we can know when the event occurred.
- The event notification shall provide a specific message of the event.
- The event notification shall provide information regarding who initiated the event.
- The event notification shall provide information regarding where the event occurred.
- The event notification shall be routed into its own topic or class so that message broker's consumers can listen to particular classes of events they are interested in.
- The event email notification message should be translated to four languages: English, Spanish, Chinese Simplified, and Japanese.

Nonfunctional Requirements

- The system must be available 24/7.
- The system REST API must be accessed using HTTPS.
- The MB must provide multiple protocols for consumer such as OpenWire and STOMP.
- The system must process a high volume of messages.
- The system must recover from failures.
- The system must retry to send email notification if failures occurs when sending email.

Use Cases

This section describes how the ENS should work and describes the most important system's features. It also describes interaction between system components: ENP, ENM, ENL, and the MB. Each use case focuses on a specific scenario, and describes the steps that are necessary to bring it to successful completion [1]. We have used a tabular approach to easily provide sufficient details about these interactions and also provide consistency between use cases. Here is an explanation of the template used:

- Use Case Id (required) - each use case have a use case id defined by UC-{number}
- Use Case Name (required) - each use case have a name to easily identify it. Basically is the operation's name.
- Description (optional) – column is used to describe the use case with more detailed explanation.
- Preconditions (optional) – required state or operations to be performed before use case.
- Actors (required) – entities involved in use case.
- Normal Sequence (required) – list of ordered steps to complete operation.
- Postconditions (optional) – required operations to be performed after use case. Could be some reset of state, etc.
- Exceptions (optional) – used for exceptions or errors when executing normal sequence.
- Comments (optional) – used for general or useful information.

Here are the most important use cases in the ENS:

Table 1
UC-01 Publish an Event Notification

Description	Service wants to publish an event.
Preconditions	Service has connection to server hosting REST API.
Actors	Service, ENP, MB, Data Store
Normal Sequence	1. Service prepare event with required information. 2. Publish the message using REST API. 3. Event notification is persisted. 4. Event notification is forwarded to MB. 5. Service receives HTTP OK response.
Postconditions	N/A
Exceptions	If Event Notification validation fails, or cannot be persisted, ENP response with a 4xx HTTP Status. Response of 5xx for any other server failure.
Comments	N/A

Table 2
UC-02 Receive Email Notification

Description	Subscribed end user to a particular event receives an email event notification.
Preconditions	Event has reach MB. Users previously subscribed to email notifications.
Actors	MB, ENM, ENL, End User, Corporate SMTP.
Normal Sequence	1. ENM consumes event notification from MB. 2. ENL logs consumed message. 3. ENM prepares email message. 4. ENM sends email to users interested on event using corporate SMTP. 5. ENM acknowledge message to MB. 6. ENL updates log entry as acknowledged. 7. End user receives the email message.
Postconditions	N/A
Exceptions	1. ENM fails to send email. ENM will retry to send email notification.
Comments	Email event notification message should be translated to: English, Spanish, Chinese and Japanese.

Table 3
UC-03 Consume Event Notification from MB

Description	Any Service can consume an event directly from MB.
Preconditions	Service is subscribed and listening to an event.
Actors	Service, MB, ENL
Normal Sequence	1. Service Receive Event Notification. 2. ENL logs consumed message. 3. Service Acknowledge Event Notification. 4. ENL updates log entry as acknowledged.
Postconditions	N/A
Exceptions	1. Service does not receive Event Notification due to a failure. Event can be recover if using a durable subscription.
Comments	MB routes Event Notification so consumers can subscribed to desired topic.

Table 4
UC-04 Retrieve Historical Event Notifications

Description	Persisted Event notification can be retrieved for later analysis.
Preconditions	REST API client has access to ENP.
Actors	REST API client, ENP.
Normal Sequence	1. REST API client execute HTTP GET method to Event API. 2. ENP responds with JSON list containing historical Event Notifications.
Postconditions	N/A
Exceptions	N/A
Comments	N/A

DESIGN SPECIFICATIONS

The ENS was implemented using the Micro Services architecture and developed using the Java Messaging Service (JMS). The JMS provides a standard java API for creating, sending, receiving and reading of messages [2]. In the modern cloud architecture, applications are developed as smaller independent blocks making it easier for developing, debugging and or maintenance. Each service is running on its own process and responsible for running its own task. Each service communicates to each other using the MB using Publish-Subscribe pattern. This architectural pattern is widely used today's applications. Applications are loosely coupled publisher or subscriber does not know existence of the other. Publisher and Subscriber only know how to either publish event messages to MB or to consume. This pattern also enables event-driven and asynchronous event notifications, while improving performance, reliability and scalability.

ENM component – Linux service responsible for consuming event messages and creating, and sending notifications as emails. Ports and protocols as follows:

- Ports – None
- Protocols – OpenWire, SMTP

ENP component – Linux service providing REST API for event publishing and responsible for routing and sending these event messages to the Active MQ broker. ENP uses Tomcat embedded to host REST API service. Port and protocols as follows:

- Ports – 5003 (REST API)
- Protocols – OpenWire, HTTPS

ENL component – Active MQ plugin to log consumed and acknowledged events. All events are logged in the Data store for later audit.

MB component – ActiveMQ server instance responsible for receiving messages from ENP and provide these to the message consumers. The MB instance uses ActiveMQ clustering feature so MB clients can auto-reconnect to another broker in case the MB goes down. The clients shall use the failover: // protocol in order to handle MB cluster. Ports and protocols as follows:

- Ports – 61616 (Listening Port), 8161 (Web Console)
- Protocols – OpenWire, HTTPS

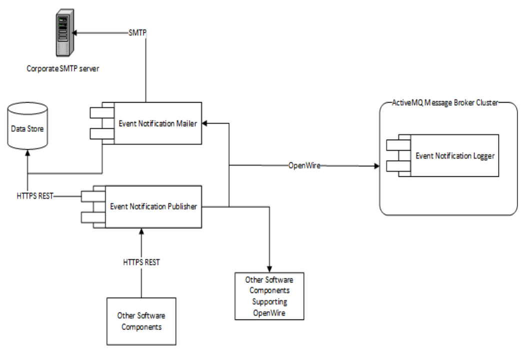
SMTP
Corporate SMTP server
Data Store
Event Notification Mailer
ActiveMQ Message Broker Cluster
OpenWire
Event Notification Logger
HTTPS REST
Event Notification Publisher
HTTPS REST
Other Software Components Supporting OpenWire
Other Software Components

Figure 1
Overall System Architecture

DETAILED DESIGN SPECIFICATIONS

This section includes detailed design specifications for the software components. We have use a short descriptive paragraph and activity diagrams to provide enough details for the software component development.

ENM Design Specifications

ENM listens to the event notifications. Once gets a new notification it prepares the email using Message Type, and Message Sub-Type translates the message to be sent to the user community. The event is only sent to the users subscribe to the event topic.

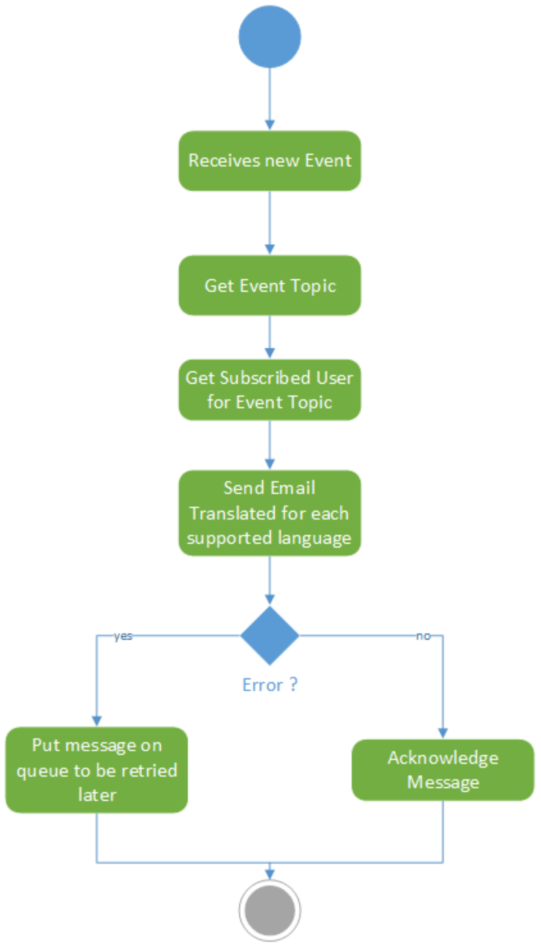
Receives new Event
Get Event Topic
Get Subscribed User for Event Topic
Send Email Translated for each supported language
yes
no
Error ?
Put message on queue to be retried later
Acknowledge Message

Figure 2
ENM Activity Diagram

ENP Design Specifications

ENP supports a REST API for other services to post events for the user community or other interested services regarding vApp deployment events, site outages, network connectivity issues, test content management. The events are stored in the Data store API. ENP uses a dual index model to store the events and then could be translated easily. The first index defines the Message Type and the second index defines the Message Sub-Type. Using a dual index model allows us to dynamically add new messages and co-locate them in the file w/o the waste of having reserved blocks of pre-defined IDs for specific purposes. The intent is to simplify message management and reduce the chance of accidental duplicate or have similar message creation.

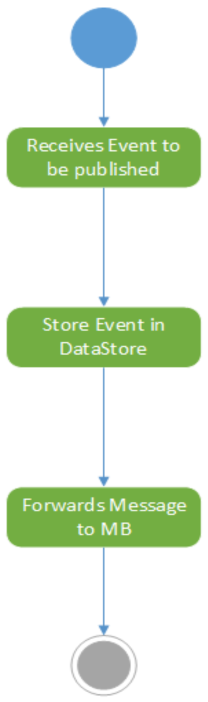

Figure 3
ENP Activity Diagram

ENL Design Specifications

The ENL intercepts all events being consumed and logs it to the Data Store. Once event is acknowledge it updates the event record in the Data Store as acknowledged. This will enable us to do audits and see what has been acknowledged and by who.

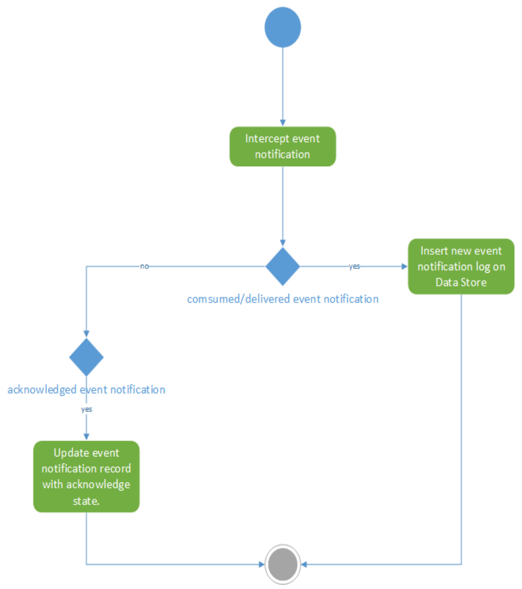
Intercept event notification
no
yes
Insert new event notification log on Data Store
comsumed/delivered event notification
acknowledged event notification
yes
Update event notification record with acknowledge state.

Figure 4
ENL Activity Diagram

Implementation Plan

This section outlines the plan for the software implementation. ENM and ENP services were developed using Spring Framework as the base framework. Spring framework is one of the most popular libraries today in the market for Web Development, built under the Java language. We also used other frameworks or libraries such as Project Reactor that helped us managing an internal queue for sending email notifications concurrently without blocking the main thread responsible for listening MB event messages. The ENL was developed as an ActiveMQ plugin leveraging the ActiveMQ plugin feature.

Also due to complexity and to improve software maintainability we have used some object oriented design patterns such as:

- **Producer Consumer Pattern:** used to manage internal queue for sending email messages. Message Listener acts as a producer. The message listener puts received message into the queue to be later processed. The mailer class acts as a consumer. It dequeuers a message and send it through email.
- **Singleton Pattern:** used for Email class and Spring configuration classes.
- **MVC Pattern:** used for developing the REST API.

Test Plan

For ensuring the ENS works as designed and meets customer’s requirements, we performed verification and validation tests. For validating system is working and built correctly we have develop unit tests to perform automatic tests and regression tests anytime we make changes to our code. Unit tests were developed using JUnit, Spring Boot Test and Mockito frameworks. For validating the ENS we performed manual tests to ensure we have built the right solution and validate we have met the customer’s needs.

RESULTS

The ENS was successfully implemented and deployed to the QA environment and it is planned to be released in the incoming Program Increment. All components have passed their Unit Tests and their respective Acceptance Tests, after executing these tests we have seen no issue within the tests performed.

FUTURE WORK

Since the scope of this project was to develop the base middleware infrastructure to be able to send Event Notifications and notify end users with emails and we also had a three month time constraint, there are some future work identified that will improve the system. Some of these features are:

- Develop a system to create notification and store them in the database. This will ensure that there are no duplicates or similar Event Notifications. Notifications should be first proposed and then revised by a Subject Matter Expert.
- Create a Dashboard User Interface to show health of sites using HEALTH Event Notifications.
- Create Reports engine to do extensive audit/analysis of the Event Notifications. Experts on statistics can look for trends on the data and predict or improve business.
- System or GUI to audit and trace what messages has been sent to consumers and which ones has been acknowledge.
- Add other vehicles of notification instead of just email notifications (e.g. WhatsApp, Messenger, etc.). This will require to research on business policies to make sure there is no violation.

REFERENCES

[1] C. Horstman, “The Object-Oriented Design Process” in *Object-Oriented Design & Patterns*, 2nd ed. NJ, USA: B.S, 2006, ch. 2, pp. 48-49.

[2] Sun Microsystems, Inc. JavaTM Message Service Specification, April 2002.