Surgical Marketing Broker -A Cloud Service for Enhancing Direct Digital Marketing

A dissertation submitted for the Degree of Master of Information Technology

A.A.E.S Abeysinghe

University of Colombo School of Computing

2020

Declaration

The dissertation is my original work and has not been submitted previously for a degree at this or any other university/institute. To the best of my knowledge it does not contain any material published or written by another person, except as acknowledged in the text.

Student Name: Eranjene Sandun Abeysinghe

Registration Number: 2017/MIT/002

Index Number: 17550021

17.11.2020

Signature: Date

This is to certify that this thesis is based on the work of Mr.**Eranjene Sandun Abeysinghe**

under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Certified by:

Supervisor Name:

______________________ ______________________

Signature: Date:

Abstract

Presented herein is a solution developed to add a new methodology to existing digital marketing methods in practice today. The existing practices of digital marketing methods are always merchant driven, meaning, the merchant decides, which offers get delivered and to whom and at what time, essentially, a push mechanism. Google AdsSense, Facebook Ads and mailing lists are good examples of such push mechanisms prevalent in the industry today. These push mechanisms have severe shortcomings when it comes to delivering the right marketing offers to the consumers, who are having complex set of selection criteria and who need them at the time they are needed, through the communication medium the consumer prefers. This problem requires a solution to be developed that is capable of handling many 'consumer aspects', worthy of calling a 'consumer-driven' digital marketing methodology, which is in fact a pull mechanism. Such a solution might find its use in domains where the consumers perform prolonged and complex searches such as lands and vehicles, where receiving the desired offer with the desired characteristics is a significant achievement for the consumer as well as a promised sale for the merchant. In this backdrop the solution may seem attractive to businesses such as ecommerce websites, tourism companies and vehicle merchants.

Acknowledgements

In the acknowledgement the primary thank goes to the Pearson Lanka employees who responded to the survey, which became the primary proof of the viability of the concept. Secondly, the project supervisor Dr. Dias for giving valuable feedback and motivation. I would also thank Mr. Dilan Anushka of Bank of Singapore, for validating and reviewing the technical stack used in the project.

Table of Contents

List of Figures

Figure **Page**

Chapter 1. Introduction

The problem is an identified gap in the field of direct marketing, which can be solved with a technical solution. Currently direct marketing takes place by means of email campaigns, SMS and social media messages. In this scenario the recipients of such direct marketing offers are referred as the consumers while the producers of the offers are known as merchants. This project is going to provide a new mechanism to allow the merchants to communicate their offers to the right consumers 'through the right medium at the right time. The 'right consumers' are a group of consumers who are aspiring to see the type of offer the seller is proposing. The 'right medium' signifies the ideal delivery method for the seller's offer to reach those consumers. The 'right time' is the time period which other consumers are looking for such offers.

Developing a technical solution to meet the stated requirements of three constraints; 'right consumers', 'right time', 'right medium', put simply, meeting those three Rs is the objective of this project. A practical example to see this problem persisting is the 'leisure and holiday' market. The consumers are searching for 'deals' on many platforms and finding a deal that matches the consumer's aspiration is never guaranteed. However, it does not mean such 'deals' do not exist, but simply implies he did not find the right deal in his domain of search. Consider, if a seller could reach the consumer in time, through a medium the consumer prefers, and offers the exact type of deal the consumer is looking for. This is a promised sale for the seller and a delighted consumer. As such, this problem is widely spread in the marketing domains which require long and complex searches, such as vehicles, flights, holiday deals, jewelry, lands and apartments. The reason is the high value of those commodities and the complexity of their nature, resulting in the consumer being picky in selecting. Therefore, the project is to explore the possibility of meeting those requirements to market an offer to a consumer in a scalable manner, meaning serving large number of requests.

Therefore, the scope of the project extends as far as implementing a fully-fledged system capable of accepting the consumer preferences and merchant's offers as inputs and generating the notifications as required to all the parties of interest. In addition, the project will take the additional task of enabling the merchants to verify if the intended consumers have opened the message.

The next sections of this thesis will entail a dissection of the project stages starting from the background of the project, the methodology used as the solution, the evaluation to find out the extent to which the stated objectives have been met and finally a conclusion to produce the verdict of entire project work. Under the background, the thesis will elaborate the need of the project by showcasing the requirements found, the use cases and the results obtained from surveys to lay foundation for the project. The methodology will encompass an in-depth view of the design of the solution constructed by means of UML and architecture diagrams. It will also provide an understanding of the various technologies utilized to meet the objectives. Evaluation is a fine critique of the work from the end users' perspective, namely the consumers and the merchants in the given context. The evaluation will be made essentially through a simulation of the real operation between the mentioned user groups. Conclusion section will essentially complement the project for the level of its achievements in terms of how best the stated problem of "To Right Consumer, At Right Time, Through Right Medium" has been solved.

Chapter 2. Background

2.2 Analysis

The project broke the ground with the survey to solidify the solution. The survey was distributed with the goal of identifying the key expectations of the consumers in terms of the value they receive with the solution and the type of mediums that they like to receive notifications through. The survey results are noted in the Appendix A. The results strongly speak in favor of the solution as 90.6% of the consumers are happy to be notified of the wishful item they have been searching for, which provides a solid footing for the project to be launched. The project has two names associated with it, one is its technical term synonymous with the type of operation it is carrying out and the other is the user-friendly term used for branding those are 'Surgical Marketing Cloud' and MessageBucket.lk respectively. Therefore, either of names used throughout the thesis is standing up for the same project.

The project was proposed to address a specific digital marketing gap, which would usher benefits to both the consumers and the sellers. It features a 'cloud computing' setting where the details are hidden from either parties and a large system acting as a mediator between the sellers and the consumers to satisfy their needs, meaning they interact only with the system but not between themselves. Figure 2.1 below shows the interaction with the system.

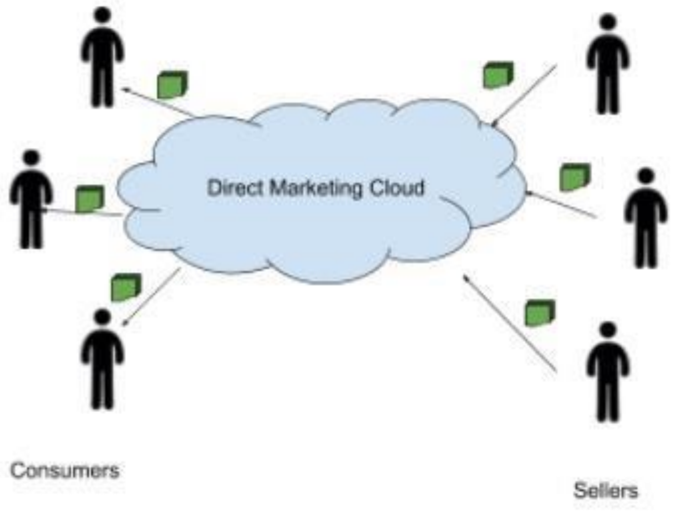


Figure 2.1- Context of the system

This initial separation of the end parties and placing a mediator system allows to control the interaction between those end parties. In this setting, the surgical marketing broker is developed, which enforces logics and rules to orchestrate and streamline the interactions between the end parties in a way that supports optimized digital marketing message delivery from the sellers to the consumers upholding the principle, ‘right offer to right consumer at the right time in the right medium’. Hence, the system analysis phase brought up few key user interactions to support that goal which are represented with the use case diagram in figure 2.2.

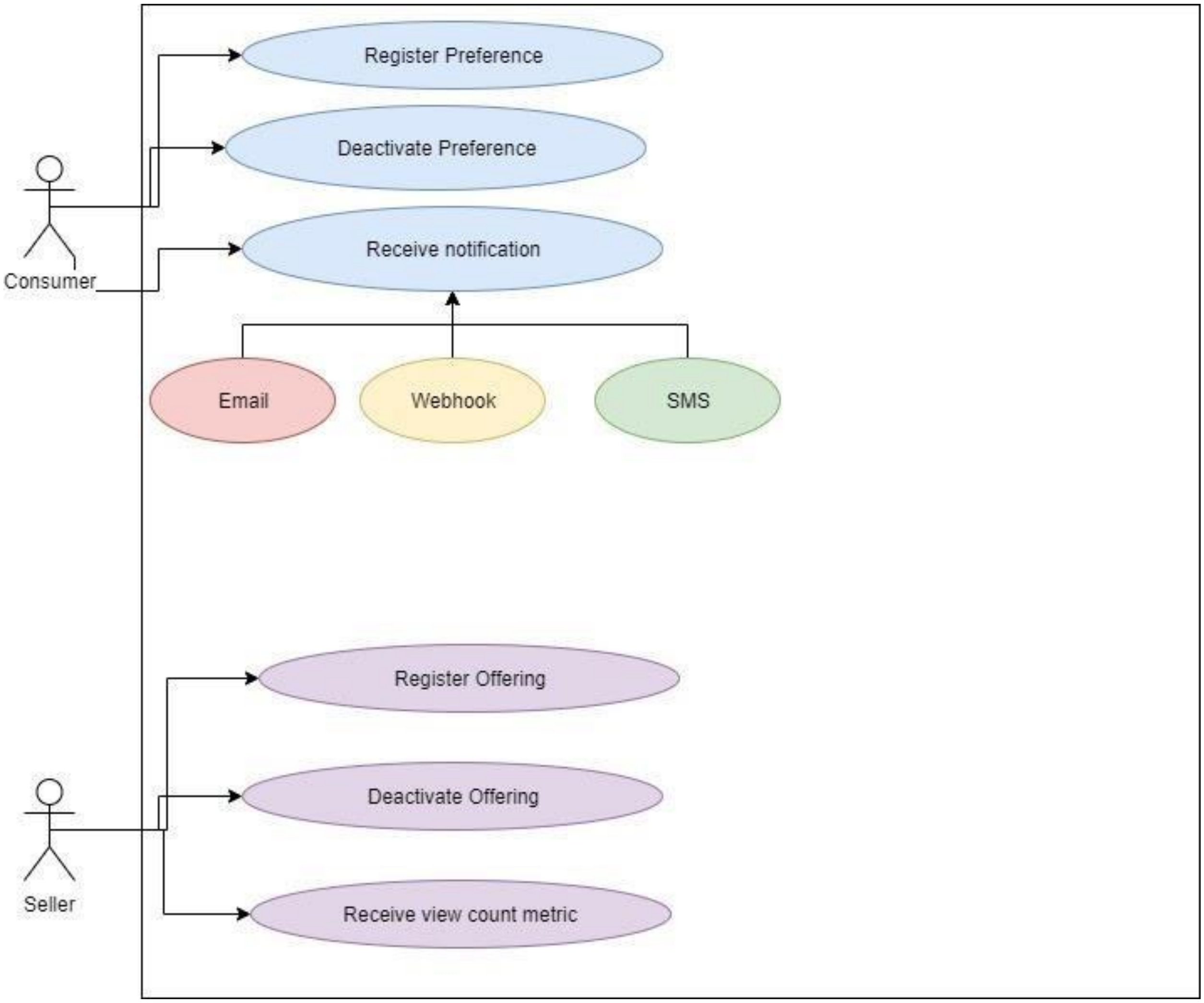


Figure 2.2- Use case diagram of the system

As seen in the figure 2.2, the philosophy is to get the consumer to express his preference of an offer for a complex search in a fine-grained manner while also letting the sellers to register their market offering in a fine grain manner. These are the key data inputs to the system which the system uses to satisfy each other’s expectations. In fact, there is no single way to capture these inputs, but a variety of ways ranging from a simple form to interactive mobile apps and personalized digital assistants. The user inputs originating from the consumer can be identified as the preferences and the notification method of his choice, while the input from merchant is the identified as offer.

The context diagram in figure 2.3 below illustrates the place of the system in relation to the external parties it interacts with. As explained earlier, both preferences and offers from external actors are fed into the system as inputs. As a result of processing those inputs, the system produces outputs which are in fact notifications sent to a variety of messaging platforms as designated by the consumer in his request. The white circle denotes that the system should be able to accommodate the future messaging platforms too if the need arises.

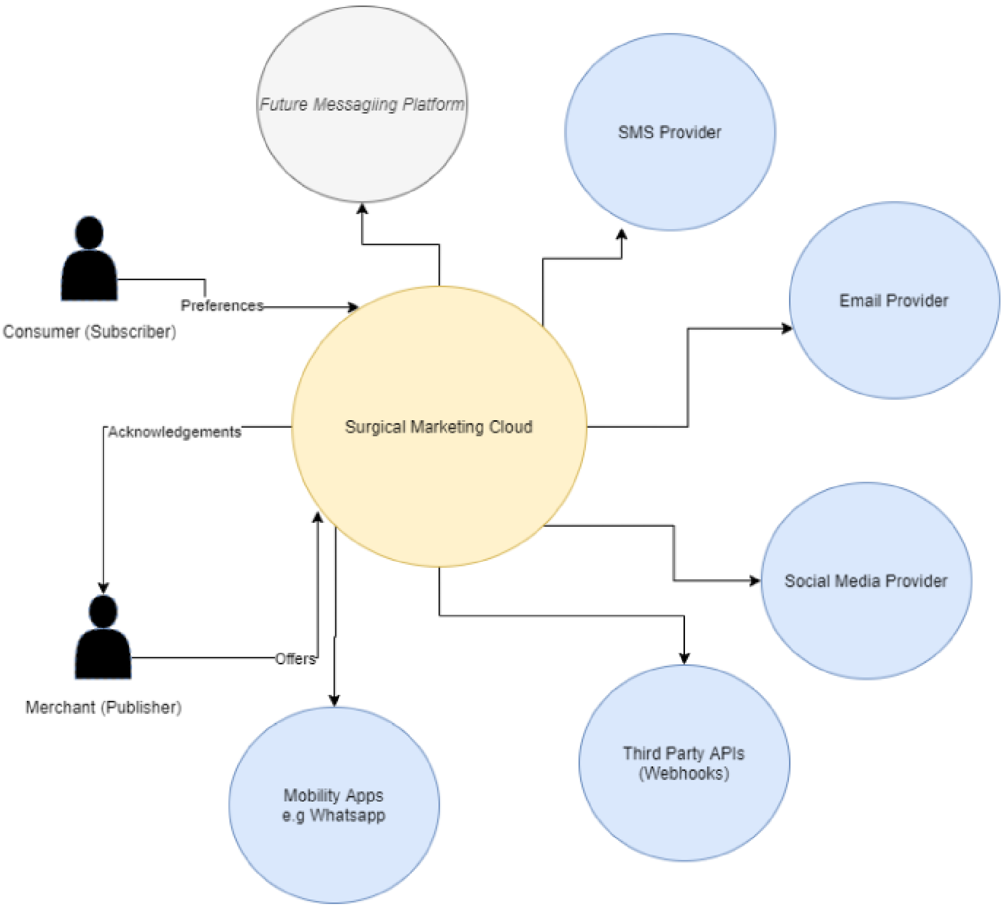


Figure 2.3- Context diagram of the system

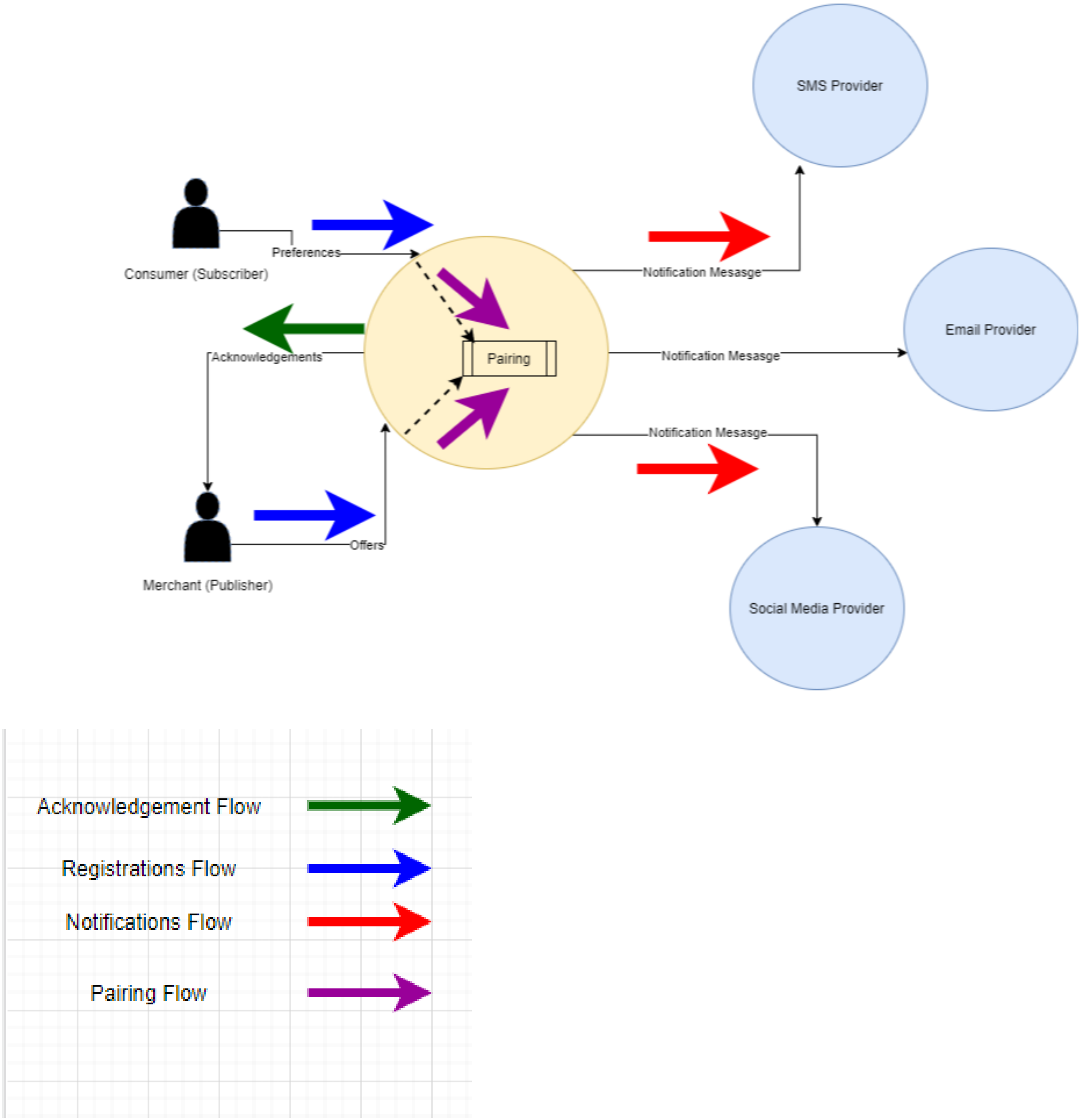


Figure 2.4 – Key workflows of the system

The figure 2.4 above represents a condensed context diagram with emphasis on categorized data flows. A brief analysis of flows will be helpful for the reader to understand the purpose of the project. Registrations flow, notifications flow and the acknowledgement flow are all boundary-crossing data flows interacting with external parties such as the merchants and consumers and the external notification platforms. The pairing flow is the only internal workflow and in fact the core workflow which produces the intended value of the system. In other words, it is the critical workflow which merges/pairs the right consumer preferences with the right offers and vice versa. It comes in the spotlight of the system as the largest process with heavy duty functionality consuming a large amount of computation power as well as the resources. Also, it is the process, whose latency can determine the efficiency, throughput of the system, meaning large delays can significantly lengthen the time it takes for a message to get to the consumers. It is also worthy of mentioning that the system's designated workflows are synonymous with a communication framework such as TCP, and in fact even with the general communication too. For example, on a broader note, it is easy to understand that the consumers are listeners who presents themselves through registrations while the merchants are speakers who sends out a message and once the right listener received the message, he will acknowledge for it. This in fact is the essence of the design of the solution.

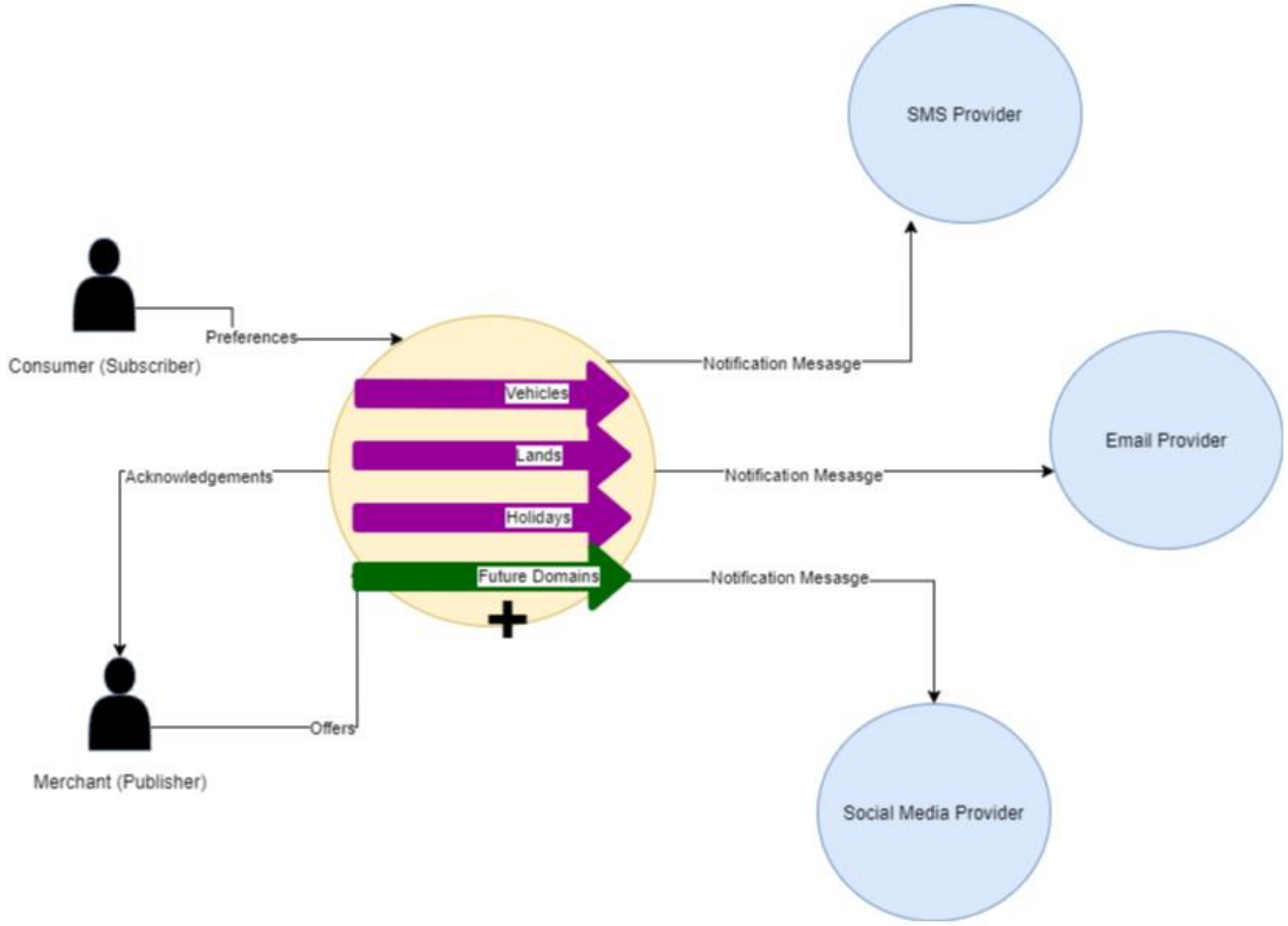


Figure 2.5- Multiple business domains through the system

In the problem statement, it has not specified any limitation of the system to adhere to a given business domain, such as vehicles or lands, but cater to any domain. This poses the consideration that the system should allow multiple independent streams of business domains, each having different configurations, user interactions, volume of data, velocity of data arrival as well as a verity of data. Most importantly, all the business domains should receive the same level of service from the system while allowing them to have their own independent configurations. For example, their own preference input forms should be allowed to be maintained and when pairing takes place, the system should pair offers to preferences only inside the given business domain. It also imperative to mention that the given business domains can theoretically grow infinitely or to a sufficiently large number, which the system must accept. Therefore, the system must take account of such additional constraints as well.

A further break down of the system from a context diagram to a level 1 data flow diagram as in figure 2.6, it is possible to observe the key processes within the system along with the associated data flows. As depicted, the system has a ‘U’ model of data flow, where the data originates from an external party and finally the outcomes arrive at the same parties as well. The color scheme used differentiates the processes based on their associated workflow i.e. registration, pairing and notifications etc. The system demonstrates multiple data flows from left to right while a single flow in the opposite direction, and this has been a key consideration in its architecture design. Because the closer process is to the left side, it is more feasible to work with data and user interactions, while being more on the right side means it is heavily process and system-oriented interacting with external parties and a long away from the user interactions. The green arrow is a major consideration for the architecture design and in fact a critical determinant for the future state of the system. The reason being that any and all of the preference and offers have to be paired in someway to generate a value of the system, which is carried out by the pairing process in the center of the system and fed into by registration processes representing both the consumer and the merchant. However, this process is costly as mentioned in a previous paragraph and therefore the architecture design aims to define a mechanism to bypass pairing through number of conceptual techniques and forward the message down to the notification process.

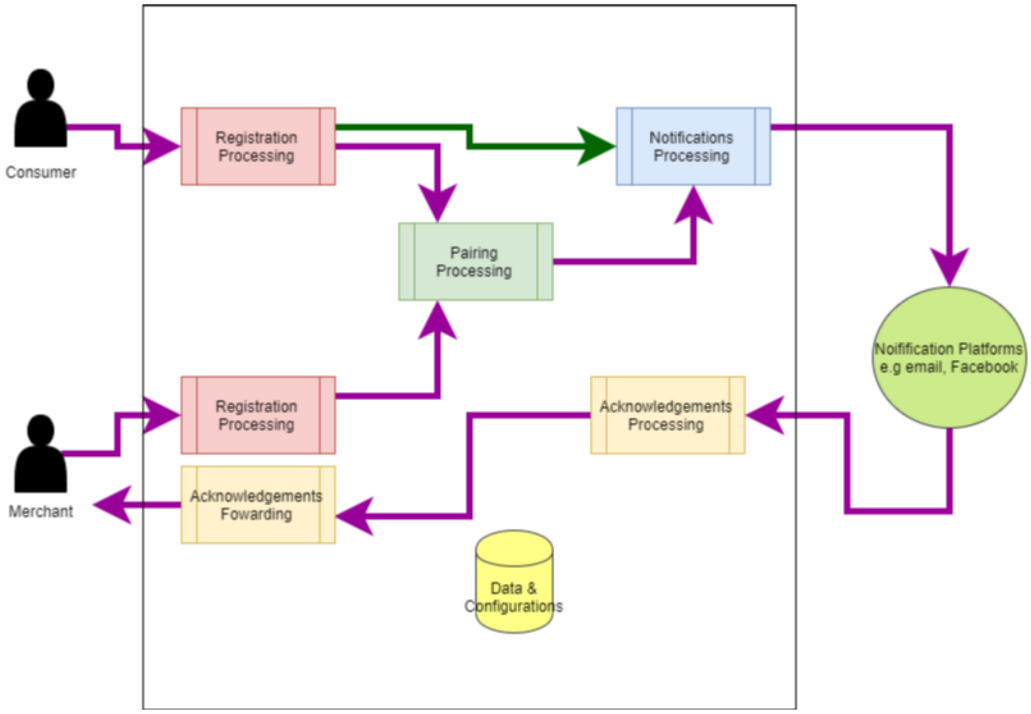


Figure 2.6- Level 1 data flow diagram of the system

In a behavioral analysis of the preferences against pairs a chart could be generated as shown in figure 2.7, which provides an insight into the number of potential pairings against every offer submitted to the system. It is an exponential growth, in which the 'pairing process' would be exhausted lowering the performance of the system and hampering any prospects of scalability of the solution. In order to provide an idea of the impact, an example would be taken such that, in a case where 1000 preferences are stored in a given business domain, a submission of a single offer would trigger 1000 comparisons and with the faster growth of preferences, it would certainly exceed 10 million comparisons per offer. This amply signifies that such behavior is not viable for the long run of the system at all.

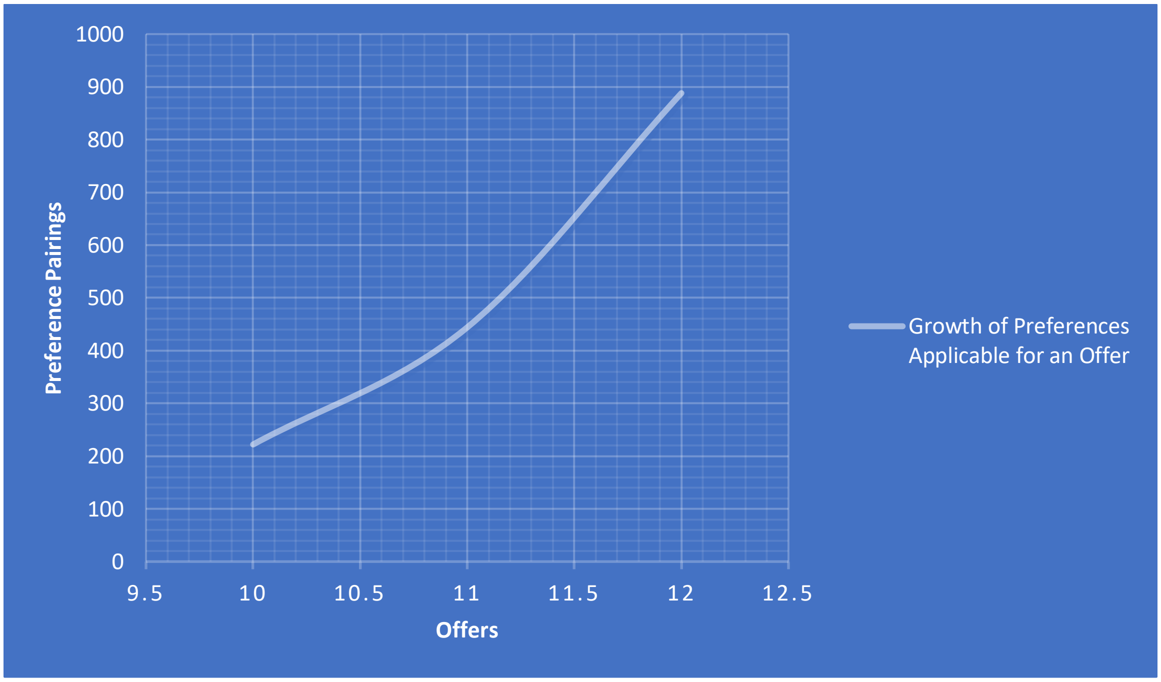


Figure 2.7- Growth of pairing

As a remedy, it was deemed feasible to employ grouping techniques where a multiple number of preferences are represented by a group. In such a setting an offer would only have to be compared against the criteria of each group rather than each preference, severely reducing the number of comparisons. A given set of groups to offer comparisons are triggered only in each independent domain and the same mechanism is implemented for all the domains. This means that a given offer of a domain will not be compared against groups of other domains, but only in its own one.

The illustration in figure 2.8 below, denotes the impact of grouping where instead of each offer having to be compared against 9 preferences, it would now be compared against 3 groups effectively reducing the number in many folds driving the efficiency up. This simple concept will set a major architectural milestone in the system design.

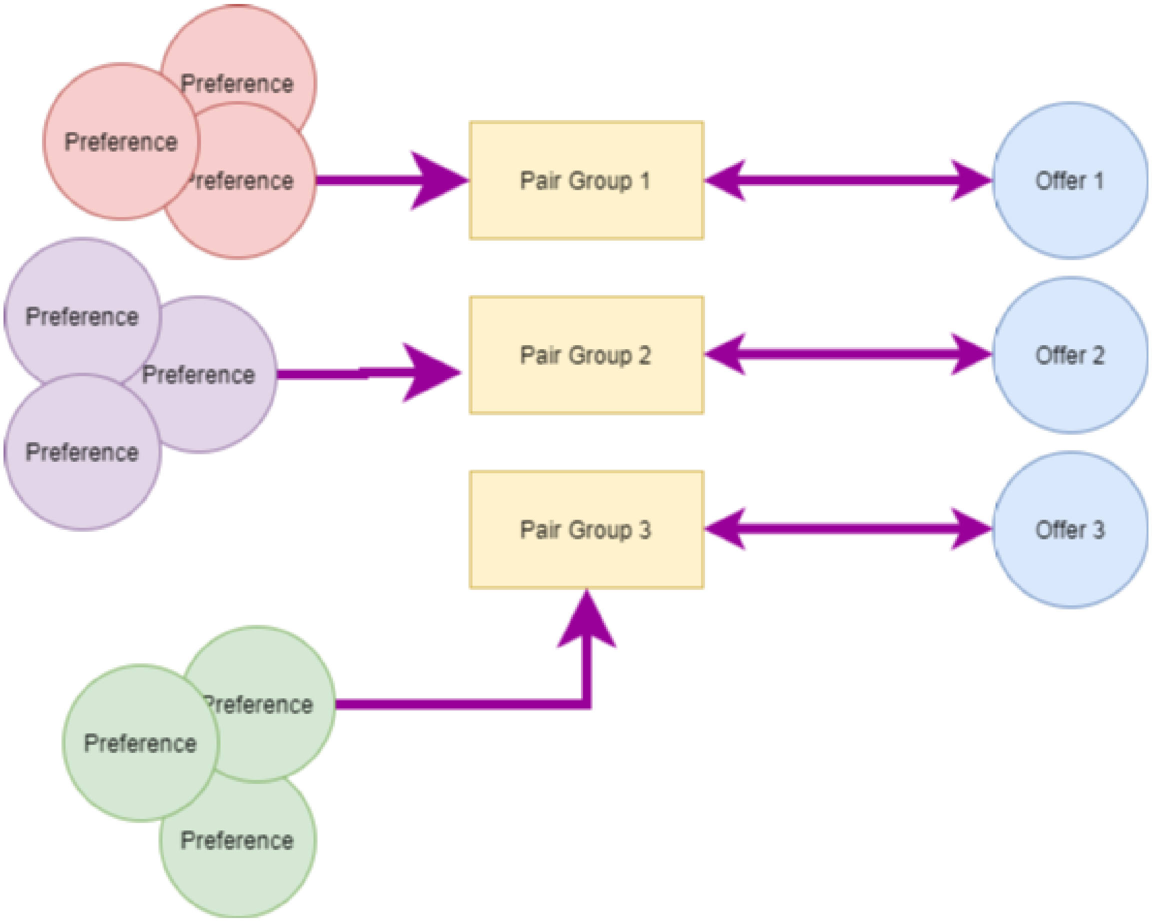


Figure 2.8 – Grouping to reduce the pairing explosion

The right technique of grouping in such a way would house large number of preferences in one group, keeping only fewer groups in existence making the system extremely efficient. This is a potential prospect in the future as the system accepts burgeoning number of preferences and therefore, it would stand as the ideal state the architecture of the system strives for.

The following diagram in figure 2.9 further solidify this stand with a projected fall of preference pairing as the grouping techniques are employed. This projection comes from the philosophy of the system design, where a at first a significant set of groups will be created to account for the variety of the preferences because one group can represent only a set of identical preferences. E.g. a preference of car with brand as Toyota and model as Prius will be assigned a group and any subsequent preferences with the same criteria will now be accepted to that group.

However, over time, with the system maturity, it is expected that many of the new coming preferences could fall into one of the existing groups, in which case the number of pairings will drop. E.g. all the preferences, however many, be it millions even, with criteria of a car having brand as Toyota and model as Prius will always fall into the group which is already paired.

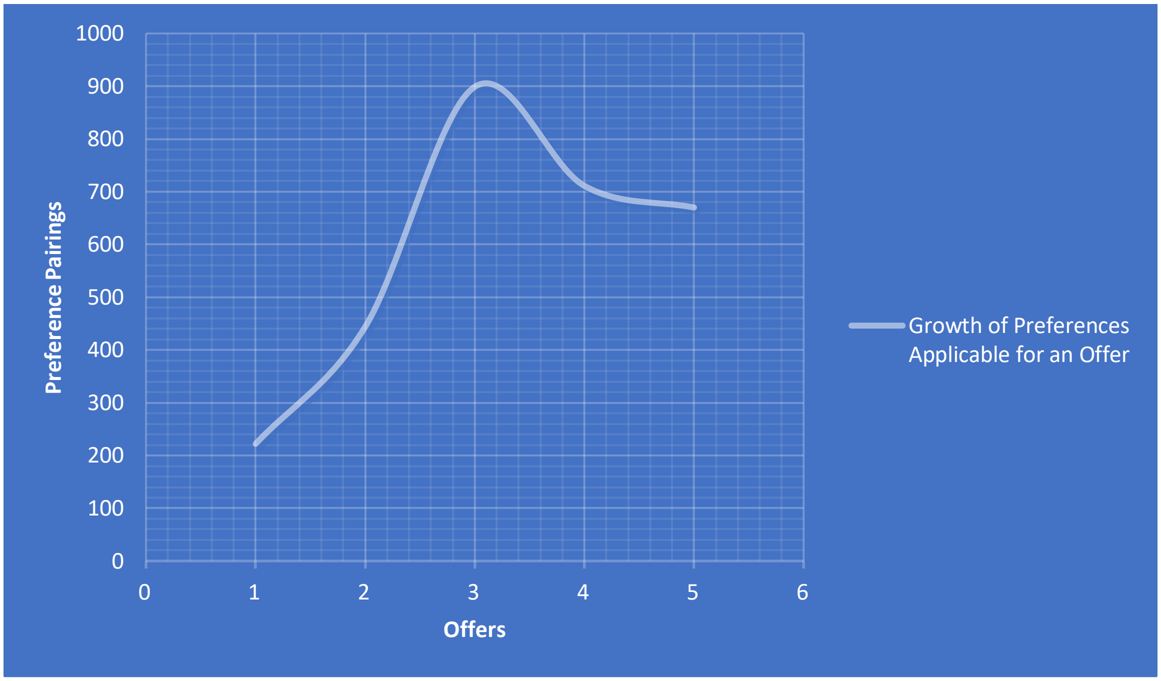


Figure 2. 9 – Effect of grouping on the pairing process

Once the chief process of 'pairing' is settled with a technology to uphold its efficiency and smooth operations, an investigation into another critical aspect of the system, 'the semantics' is necessary. As mentioned earlier, the system caters a range of business domains whose preference specifications vary between each other with a dynamic number of fields, data types and data variations. For example, in the lands business the terminology is often in land sizes in acres and the locations while in the vehicle market, it would rather be number of owners, models, brands and transmission types etc. These terminologies are domain specific, therefore as a neutral system catering all the fields it cannot be expected from the system to interpret these terminologies for pairing with the offers. As a remedy, it was decided to consider a mathematical/logical form of notation, notably Boolean expressions for the purpose. Such expression form would not carry any domain semantics, but rather mathematical and logical operators to denote the relations between the operands, easily representing the preference. E.g. a preference with a criteria of a car with brand as Toyota and the maximum price as 200000 can best be represented with Boolean as '*brand' == 'Toyota' && 'price < 200000.* This is yet another major milestone of the system that was achieved.

Another critical newly evaluated aspect of the system is user accounts. From the system's perspective, any genuine user is valid, and it does not care about the context of the user, whether in institutional level of as a member of a group. On the other hand, the system which is poised to grow significantly as a communication facilitator does not intend to keep a bulk of user context data in its arsenal, which will become a burden in the future. Creating user accounts burdens the system by having it to manage the security workflows as well as discouraging users to interact with the system. Upon considering all these facts, an ideal solution for this problem is to use OAuth

2 based OpenID signing services provided by major players such as Google, Facebook, Twitter and GitHub. The advantage of using such service is that the system must retain a minimal set of information while the auth provider certifies of the user's context such that the email address is valid. From the user's perspective (consumers and merchants) it is much convenient from them to login using the social media account.

Acknowledgement of the receipt is a critical workflow of the system, which manifests into higher usability and attraction of merchants. By acknowledging the merchants of the number of consumers who has 'seen' the offer, the system assures the usefulness of it to the merchants. A critical aspect of such notification is that upon clicking the link of the offer by a consumer a message must arrive at the system, which will in turn update the view count for that offer. However, implementing such mechanism introduces several key challenges to address.

- It should not double count, meaning that the same message clicked twice, should not make the system increment the view count twice.
- It should not accept obsolete ones, which were posted long time ago
- It should not accept artificially generated acknowledgement traffic.

The extensibility is an utmost concern of the system design. The system is poised to grow and take on more functionality. For example, adding new notification platforms and sharing of an event of one service across the others, which perform a subsequent action. Any architecture defined should support such scenarios. For example, there could be some action that would be triggered after a new preference has been registered an existing group. Figure 2.10 demonstrates such a sharing of update scenario.

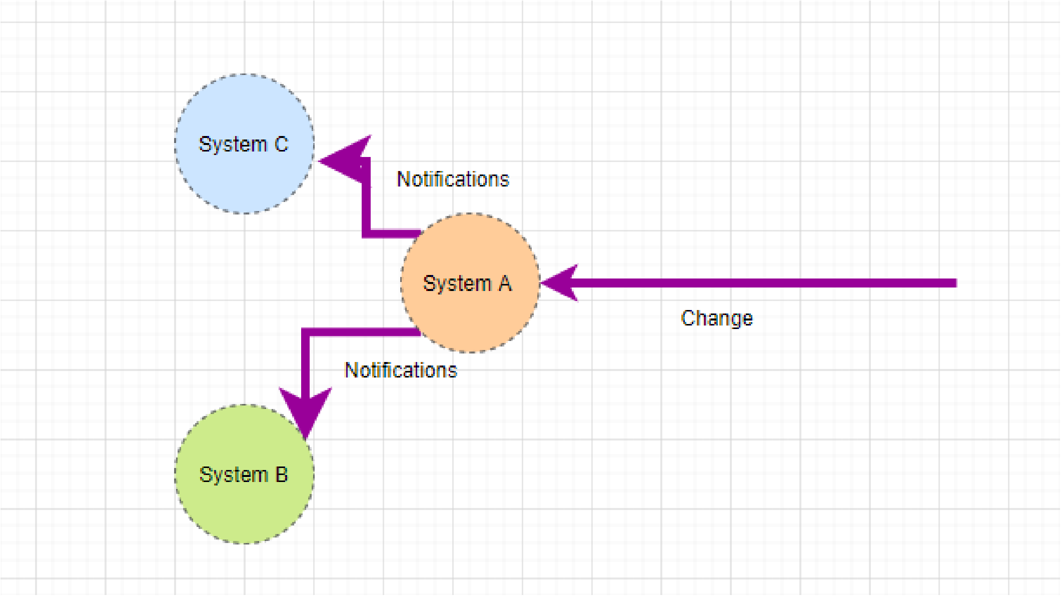


Figure 2.10 – Broadcasting updates throughout the system

Finally, record cleanup is a secondary concern, that has to be addressed in a design. Unlike in a traditional static system, this solution calls for a sophisticated clean up mechanism that takes care of many concerns. For example, the performance impact of delete, the consistency of data residing on multiple services after the deletions etc. It has to be a solution comprising of a mix of soft and hard deletes together with events and signals where the right listeners will carry out total deletion.

Based on all the observations specified in the analysis as depicted before it can be stated that the system boils down to the major set of requirements specified below as summary.

Core business workflows:

- **Registration of Preference**
  The system stores the consumer preference data in its data store

- **Registration of Offer**
  The system stores the offer data in its datastore

- **Preference and offer normalization (flattening)**
  The preferences and offers being registered have to be encoded to uniform rich code ideal for evaluation.

- **Preference classification and assignment to distribution lists**
  The preferences are classified based on the content to group with other preferences having the same content.

- **Offer-preference pairing**
  Link the preferences with the offers.

- **Update detection of the distribution lists**
  The system sends notifications when an update happens to a distribution list such as when a new preference is added to the list as a result of a new user registering his preference.

- **Updatable consumer batch selection from the distribution lists**
  In the selected distribution list, the subset of consumers, which should receive the update are selected.

- **Selection of delivery medium for updatable consumer batch**
  The delivery method registered with each consumer in the subset of consumers to be notified is selected for forwarding.

- **Dispatch to the updatable consumer batch through the registered delivery medium**
  Dispatch the update to the selected consumers through the designated medium such as email.

- **Dispatch the notification when the consumer views the offer**
  When the consumer clicks on the offer a message should be dispatched to the system for processing

- **Process the view count for each offer**
  Process the stream of view events and increment the view count of the offers

Additional workflows:

- Offer-preference unpairing
- Offer deactivation
- Preference Deactivation

2.2 Review of Similar Systems

Google Adsense and Facebook ads

An advertisement service that tracks the user (consumer) behavior and performs classifications and profiling to forward the ads (offers). This method is ideal to distribute open-ended advertisements to a large audience with high relevance, but neither the consumer nor the seller has control over them to satisfy their specific needs. The needs of specifying their offer and preference characteristics and delivery media is not an option and the period the consumer receives the offer is at the discretion of Google or Facebook. The delivery method is not generic either, since the consumer has to be logged into either of the platform to receive the offer.

Therefore, this methodology does not address the problem, the ‘right offer for right consumer at the right time through the right medium.

Mailing Lists

The once popular mailing systems that are used to send newsletters implies a similar but much basic functionality. It is a simple registration system of email addresses that are notified when an update needs to be broadcasted. It does not take the specific user needs into account.

Adhoc notifications in sites

Some websites have implemented adhoc notification systems such as selection of a basic topic and then registering the email address to receive updates when some update to the topic happens. This again is a solution with a basic scope and not extensible to accommodate the complexity of the user needs. Further, the businesses that possess these systems do not dedicate the complex infrastructure and processes necessary to handle such a need as their line of business is different.

Some features of the system were inspired by the operation of some already existing technologies in communication. The main example being Message Oriented Middleware such as RabbitMQ facilitating a producer-subscriber interaction. Such model of communication is ideal to support event driven operation. Apache Kafka has the ideal infrastructure for processing streams and the concepts surrounding it would help immensely to design the project such as the ‘continuous flow of messages and handlers. Google has the GDL (Google Distribution Lists), the concept of which was used as a building block in the project to design notification groups listening for updates.

Chapter 3. Methodology

3.1 Distribution List Design

Based on the observations noted in the analysis phase the design of the system is constructed as described below. In the analysis the key concerns were surrounding the pairing process which relied on a grouping method of preferences in order to boost the system performance and evade bottlenecks. The so called ‘groups’ is technically worded as ‘distribution lists’ from here onwards. Distribution lists form the heart of the system, which contains groups the preferences under an identification that represents the preferences and the associated offers paired. Figure 3.1 below is an illustration of a distribution list.

Distribution List ID: Base64 string
Preferences [] : array
Offers [] : array
Criteria: String

Figure 3.1- Skeleton of a distribution list

As the figure 3.1 above depicts the distribution list ID is a Base64 string representing the criteria of each member preference. This means that at any given time, using the criteria of an individual preference the associated distribution list can be retrieved. Also, the preferences and the paired offers are contained as arrays as shown.

Criteria String is a Boolean expression representing the preference criteria no matter how complex it is. A typical criteria expression can be denoted as below.

$price < 10000000 && $color==’black’

As shown, a criterion would include both the placeholders for the aspect of criteria indicated by $ marks and a condition denoted by an operator such as ‘<’ and a value. An offer that wants to meet the criteria should replace the placeholder with its corresponding value and evaluate the expression. For example, a vehicle offers marked with price 20000 having black color will replace the $price with 20000 and $color with ‘black’ respectively which derives the expression such as below.

20000 < 1000000 && ‘black’ == ‘black’

Upon evaluating the expression, the outcome would be derived as true essentially pairing the offer with the distribution list. This mechanism sets the cornerstone of the system. A real distribution list is presented in the Appendix B.

3.2 Pairing Process Design

Deriving from the previous section, the pairing process carries out few steps to achieve its goal.

1. Retrieves the offers from the database in the given business domain in order to pair with a newly created distribution list or retrieve the distribution lists from the database in order to pair with a new offer.
2. Replace the placeholders of the criteria of the distribution list with the corresponding values of the offer.
3. Evaluate the Boolean expression to determine whether the two should be paired and construct the pairing.

In order to boost performance, an additional two steps are defined; index look up and paged retrieval. Iterating through all the distribution lists or offers in the database will create a huge performance impact. Therefore, the system will load an index containing a list of criteria of all the distribution lists available and perform evaluation to determine the ones that should be paired. Paged retrieval is a mechanism to perform pagination of records retrieving a specified number of records into the memory, rather than the whole bulk, which prevents a possible memory crash. Also, the it is important to note that the process is designed to utilize the multiple cores of the processor, essentially using parallel processing techniques such as parallel streams. Figure 3.2 below showcases the pairing operation.

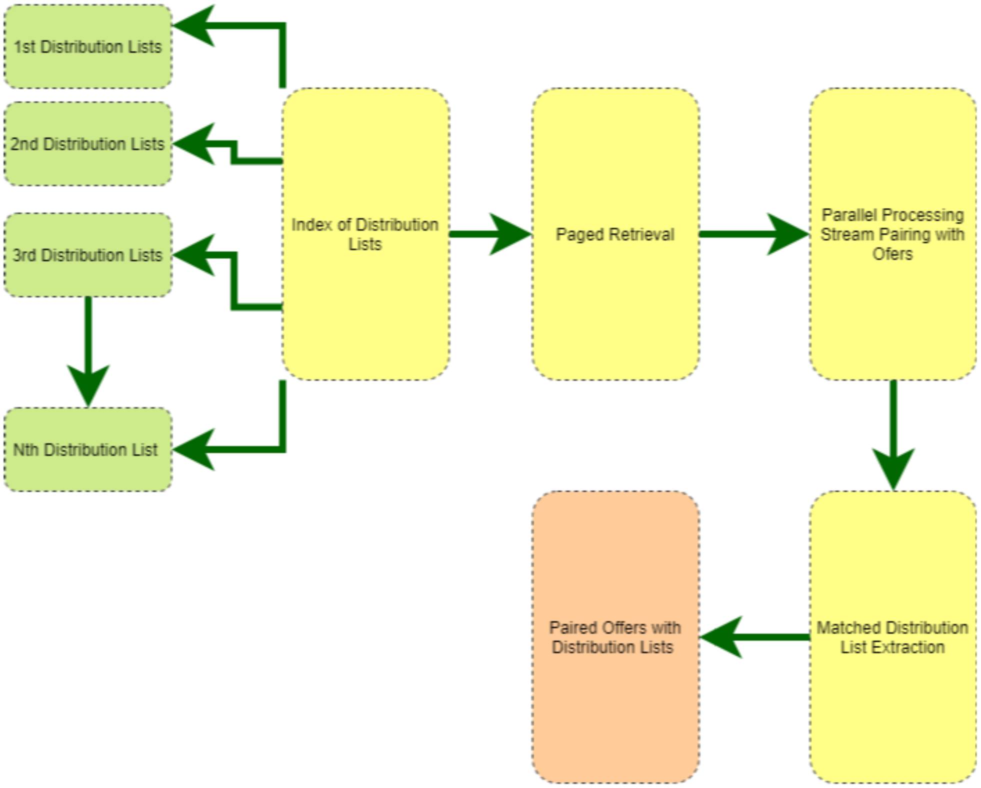

1st Distribution Lists
2nd Distribution Lists
3rd Distribution Lists
Nth Distribution List
Index of Distribution Lists
Paged Retrieval
Parallel Processing Stream Pairing with Ofers
Matched Distribution List Extraction
Paired Offers with Distribution Lists

Figure 3.2 – Pairing process in nutshell

3.3 Acknowledgement Design

Acknowledgement process is designed with a unique philosophy innovated in this project. It is called the envelope system. An envelope can only be broken once to retrieve the letter inside it. Similarly, a notification that is clicked on by consumer can only be counted once as opening and any subsequent clicks will be ignored.

In this scenario, the envelope is a static webpage containing the link to the offer, which will self-re-direct upon rendering in the browser. Just before the redirection it will send an acknowledgement that the envelope has been opened, which triggers a chain reaction to increment the view count for the given offer.

The notification link presented to the user will contain the following elements.

<envelope base URL>/<offerId>/?timestamp&signature

Envelope base URL is the URI of the static page which loads up to the browser. Timestamp is the time the link was generated, and the signature is an MD5 signed string which helps the system to identify whether the request is a forged one or a genuine envelope opening.

Figure 3.3 below illustrates his process.

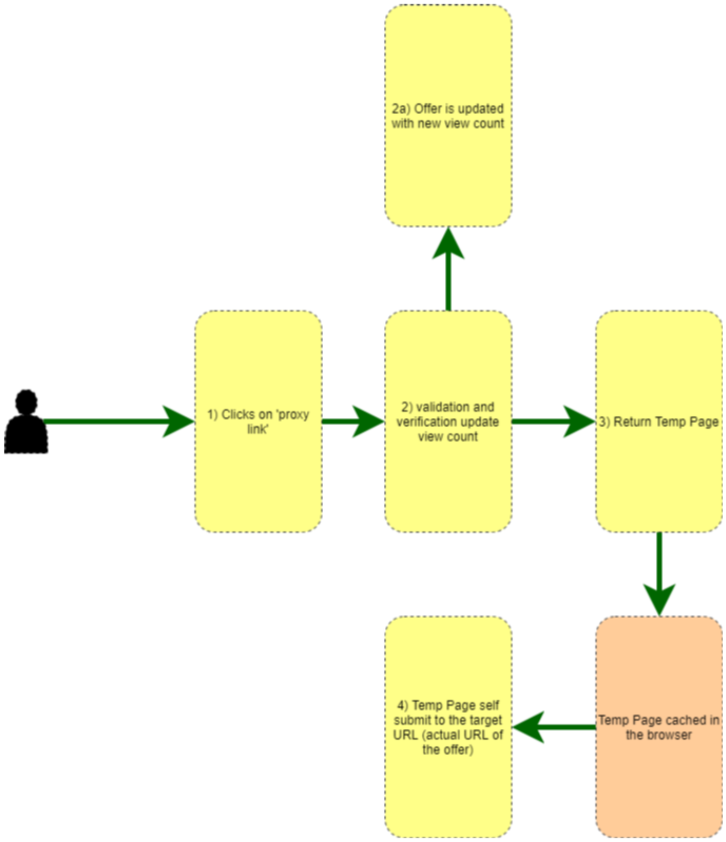

2a) Offer is updated with new view count
1) Clicks on 'proxy link'
2) validation and verification update view count
3) Return Temp Page
4) Temp Page self submit to the target URL (actual URL of the offer)
Temp Page cached in the browser

Figure 3.3 – Acknowledgement process in a nutshell

3.4 Full process overview

As a cloud service, the system is expected to receive large number of preferences, for which matching them to each of the offers is next to impossible. Therefore, a protocol had to be designed to meet this notification requirement as well as respect to the scalability, extensibility factors of a cloud service. Below is the 9-step notification process described as a user story, which is the core of the system and exclusively developed to deliver the offers to the subscribers efficiently.

The protocol is as follows.

1) System receives the user input with a special metadata called “domain”
2) System finds and loads the domain interpretation model, which contains the rules to interpret the user preferences to form a Boolean expression
3) System converts the created Boolean expression into Base64 key.
4) System looks up for existing distribution lists and if available add the new preference to the subscribers’ section of the loaded distribution lists.
   4a) if no existing distribution list is available for the given key, system creates a new distribution list
   4a1) system adds the new preference to the new distribution list
   4a2) system loads the existing offers and add the matching offers to the distribution list
5) System updates the preference as active and turns on the listening state
6) System creates a notification package from the distribution list to include the latest preference and all the offers
7) System duplicates the notification package that include one offer for the full set of subscribers
8) System splits each of the notification package from the previous output into further sub packages based on the notification medium
9) System routes the notification packages to the notification media.

   The same process is illustrated graphically in figure 3.4, with a flow chart.

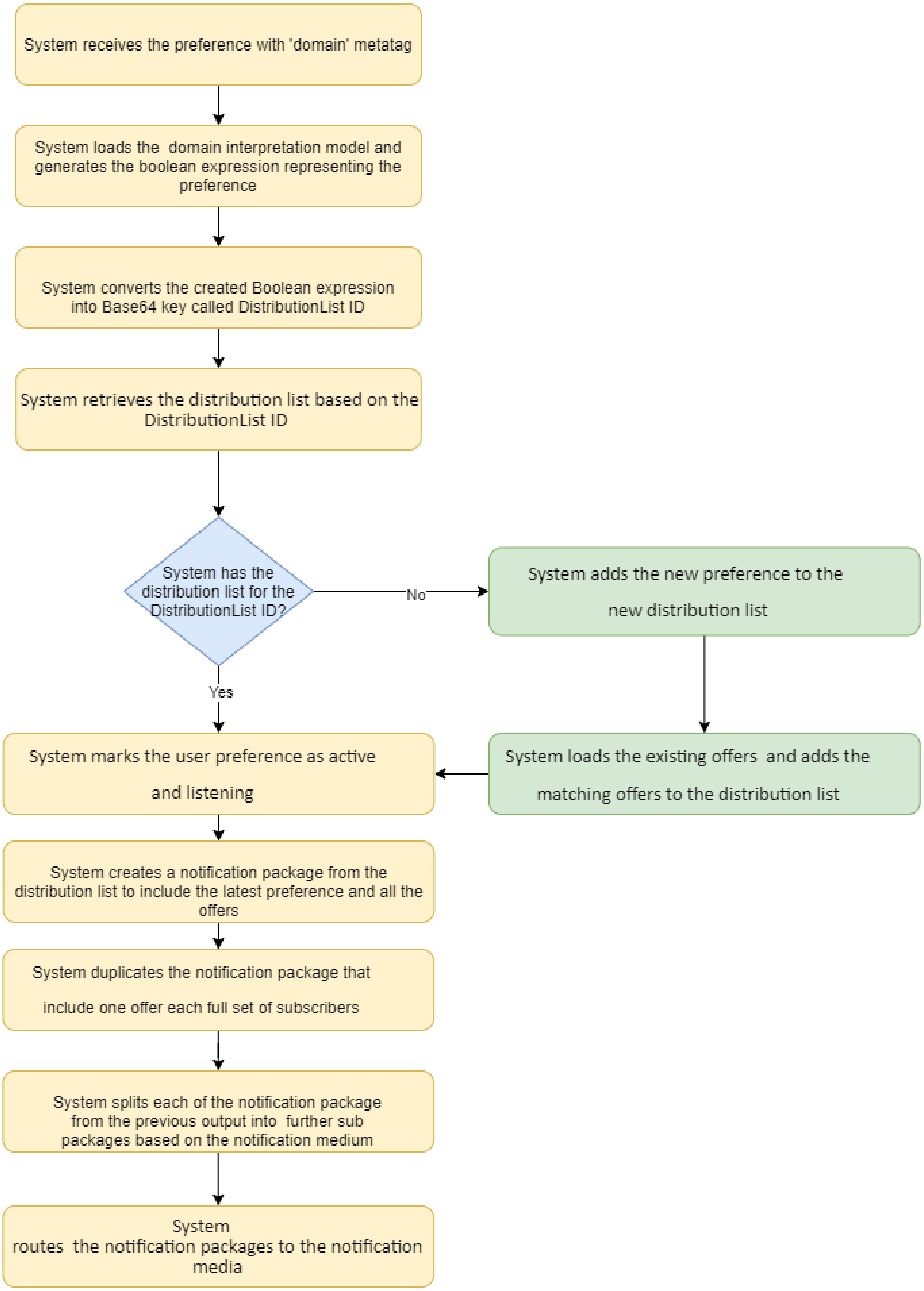

System receives the preference with 'domain' metatag
System loads the domain interpretation model and generates the boolean expression representing the preference
System converts the created Boolean expression into Base64 key called DistributionList ID
System retrieves the distribution list based on the DistributionList ID
System has the distribution list for the DistributionList ID?
No
System adds the new preference to the new distribution list
System loads the existing offers and adds the matching offers to the distribution list
Yes
System marks the user preference as active and listening
System creates a notification package from the distribution list to include the latest preference and all the offers
System duplicates the notification package that include one offer each full set of subscribers
System splits each of the notification package from the previous output into further sub packages based on the notification medium
System routes the notification packages to the notification media

Figure 3.4- Full system process illustration

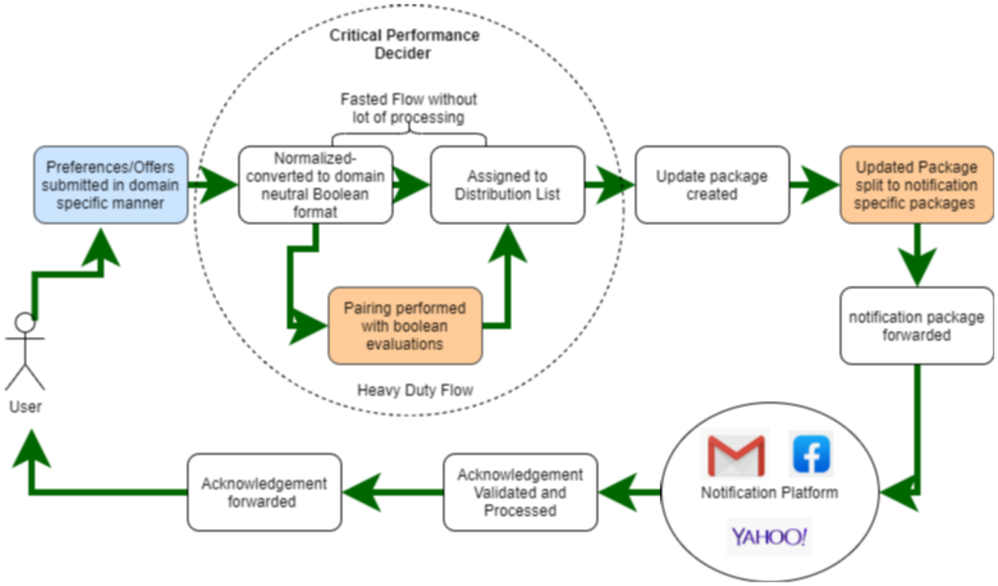

Critical Performance Decider
Fasted Flow without lot of processing
Preferences/Offers submitted in domain specific manner
Normalized- converted to domain neutral Boolean format
Assigned to Distribution List
Update package created
Updated Package split to notification specific packages
Pairing performed with boolean evaluations
Heavy Duty Flow
User
notification package forwarded
Notification Platform
YAHOO!
Acknowledgement Validated and Processed
Acknowledgement forwarded

Figure 3.5- Conceptual view of the system

Figure 3.5 above shows the process in conceptual organization ready to be implemented. The illustration identifies the blue box as the interfacing process processing user inputs and registering. The orange color boxes are identified as heavy-duty components with lot of processing calling for optimization techniques and precision. As indicated by the dotted circle the pairing process is the critical performance factor and optimization of the processes in the circle would greatly enhance the capabilities of the system. Figure 3.6 displayed below showcases where the business domain separation should take place with the yellow strip. This also means that from this point onwards all the domain-oriented data becomes independent as well as they become domain neutral with the Boolean expression encoding mechanism mentioned in the analysis phase.

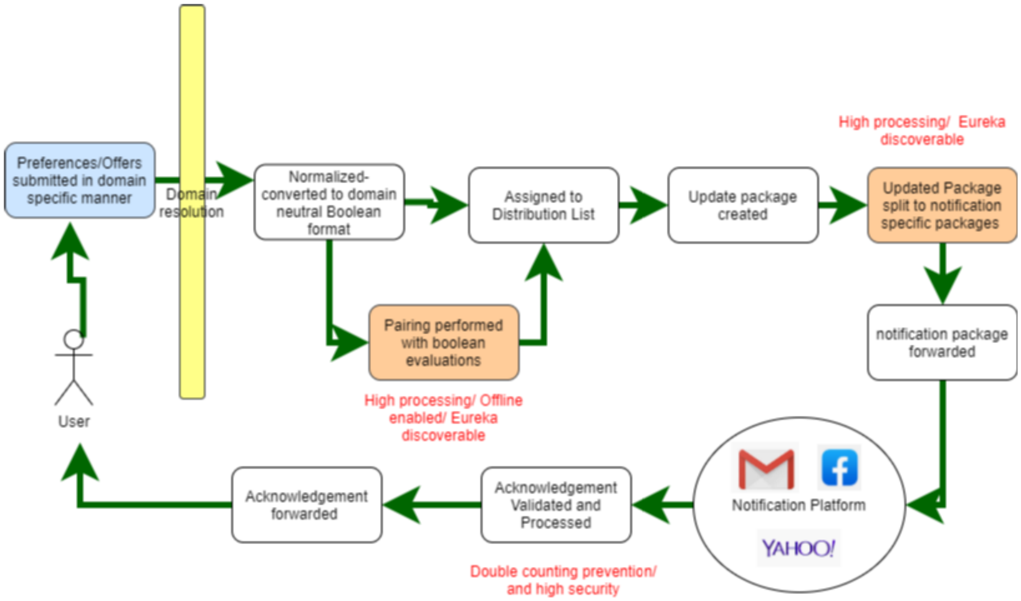

High processing/ Eureka discoverable
Preferences/Offers submitted in domain specific manner
Domain resolution
Normalized- converted to domain neutral Boolean format
Assigned to Distribution List
Update package created
Updated Package split to notification specific packages
Pairing performed with boolean evaluations
notification package forwarded
User
High processing/ Offline enabled/ Eureka discoverable
Acknowledgement forwarded
Acknowledgement Validated and Processed
Notification Platform
YAHOO!
Double counting prevention/ and high security

Figure 3.6- Domain separation method in the system

Based on the in-depth analysis and the observations as specified in the previous chapter, the system's main behavior driving force is set to be events, which means it'd be an event driven design. The choice for the event driven approach was selected to this type of application because the interaction of the end parties with the system are best represented as events which need to be handled in the order they occur. It also allows the system to scale and handle better throughput with optimized set of resources. From an abstract point of view, the system would behave as a stream with each offer entering and being dispatched continuously without any impedance.

3.5 Designed Software Architecture

The philosophy of the designed architecture is to ensure continuous flow of operation in a single direction in the form of a flowing stream, rather than a matrix of crisscross communications. It also greatly emphasizes a reduced database calls in along the operation. This was driven by the intension of increasing the throughput and reducing the latency and resource contention as much as possible. In order to facilitate this philosophy several architectural patterns are at play in the solution.

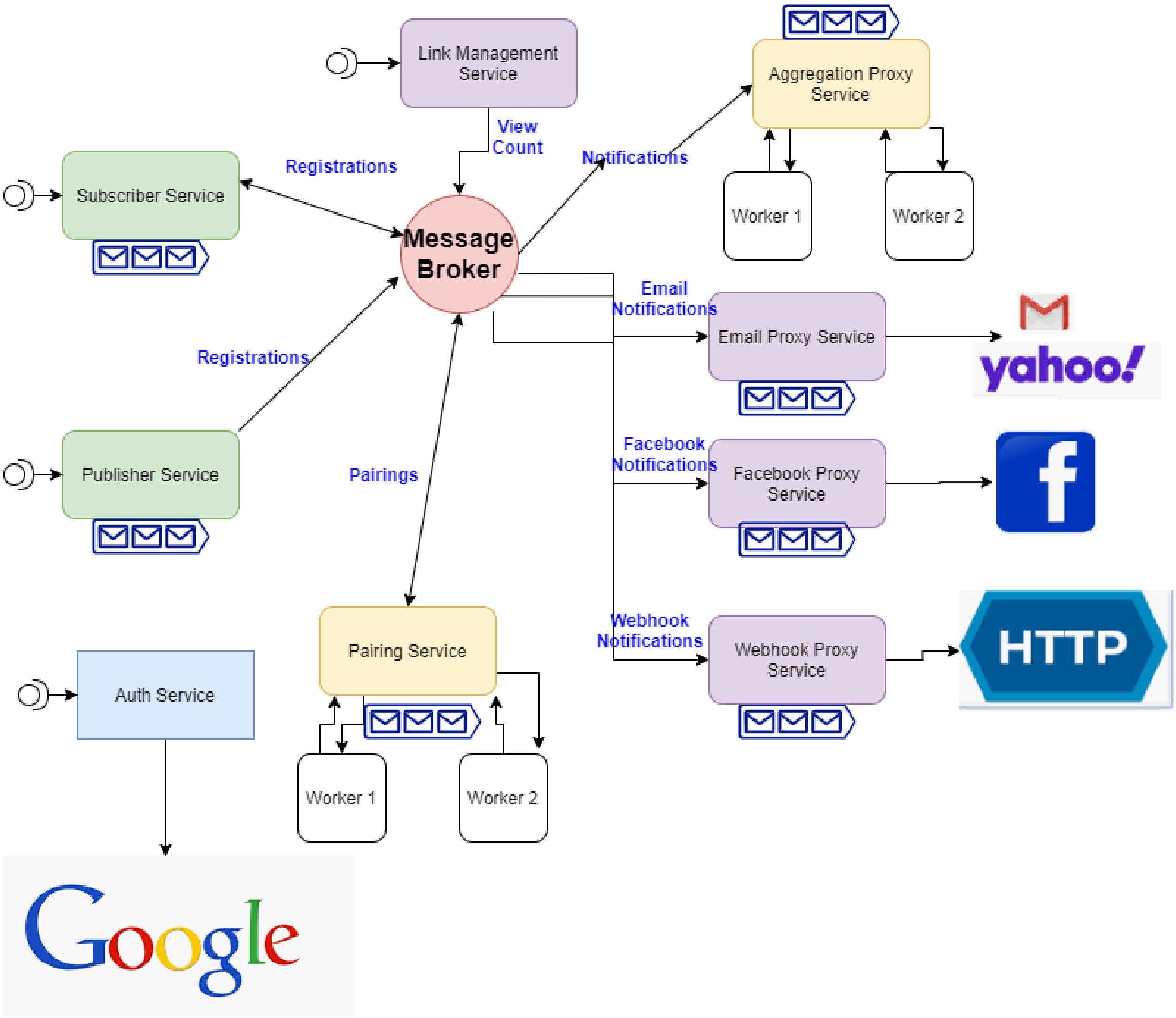


Figure 3.7 -Architecture of the system.

Communication Diagram for the notification process when a new preference is added.

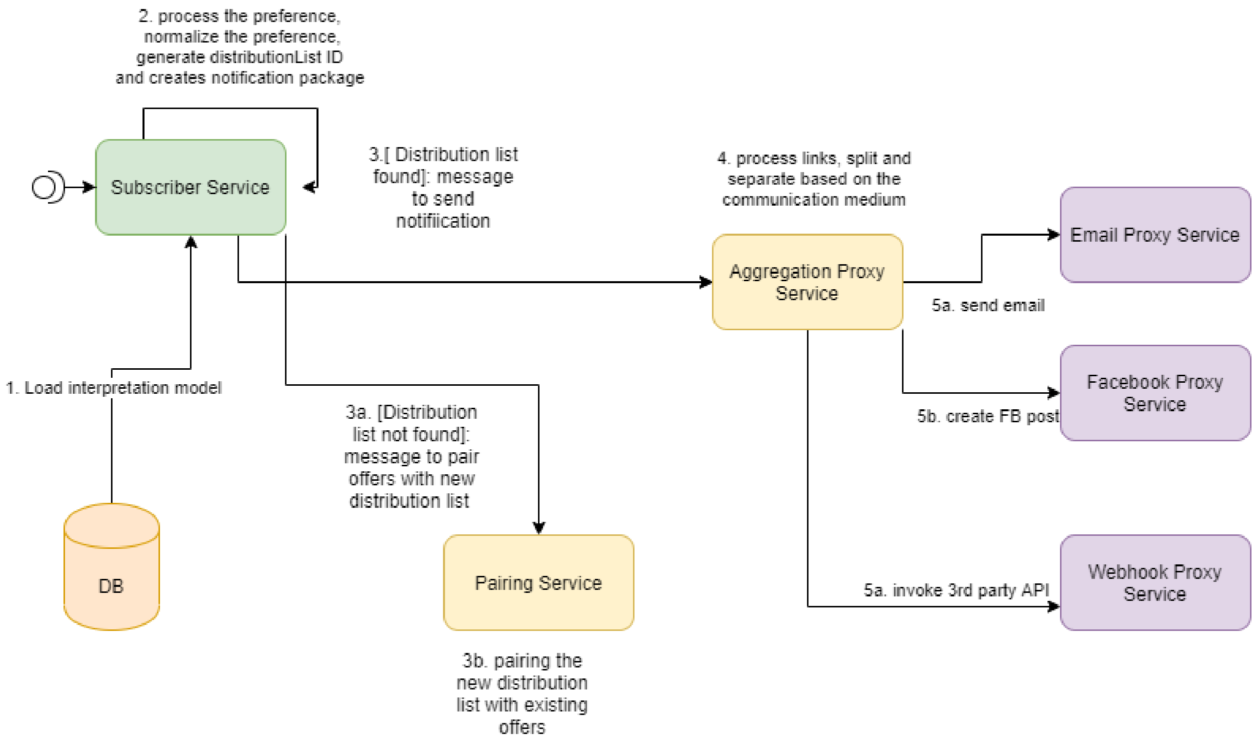

2. process the preference, normalize the preference, generate distributionList ID and creates notification package
Subscriber Service
3.[Distribution list found]: message to send notifiication
4. process links, split and separate based on the communication medium
Email Proxy Service
Aggregation Proxy Service
5a. send email
Facebook Proxy Service
5b. create FB post
1. Load interpretation model
3a. [Distribution list not found]: message to pair offers with new distribution list
DB
Pairing Service
5a. invoke 3rd party API
Webhook Proxy Service
3b. pairing the new distribution list with existing offers

Figure 3.8- Communication diagram for the new preference added scenario.

Microservices architectural pattern

As indicated in the figure 3.7, the solution is developed on micro services architecture. It will ensure that the services are independently scalable to cater to the load. For an application with high-throughput, microservices is not a luxury, but a criticality, meaning the expected load cannot be delivered otherwise. A prominent use case in the application that calls for micro services architecture is pairing service and aggregation service workers, which carry out heavy processing using a large amount of computational processing power. In this case, without many instances of the given service to compensate for the load, the entire system might slow down.

The microservices architecture of this solution, comprising of multiple instances of the same heavy-duty service, make use of load balancers. The Netflix Ribbon load balancers dish out the requests to the heavy-duty instances in the round robin fashion streamlining their operation without causing resource contention.

Message Broker Pattern

One other prominent architectural pattern used is the message broker pattern. Such a pattern became useful in scenarios where the speed of data transmission had to be controlled when there are possible mismatches of speed between the producer of the data and the consumer of the data. Implementation of this architectural pattern was materialized using Spring's AMQP classes which make use of a MOM (Message Oriented Middleware), called RabbitMQ. In figure 4.1, there are many queues illustrated, which were implemented for the purpose described above. A clear example in this context is the queue between the notification proxy and the email service. The speed difference between these services can vary very much.

In the solution the message broker acts as the orchestrator of the flow essentially decoupling all the component interactions and allowing the solution to be extended easily. Figure 3.9 demonstrates how the microservices are connected through the message broker.

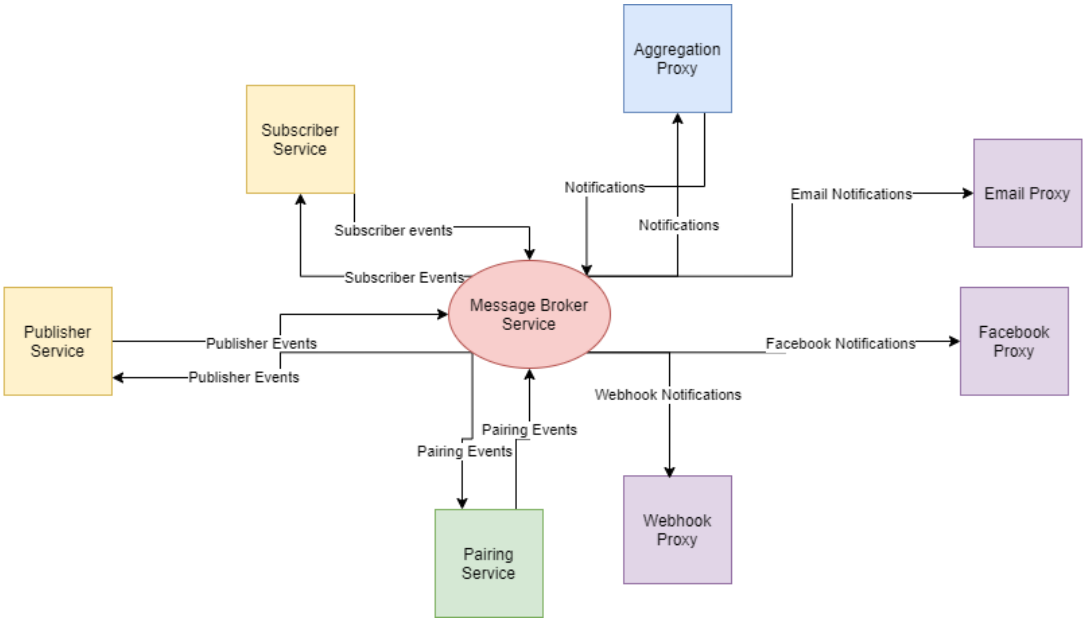


Figure 3.9- Message Broker connectivity throughout the system.

In the figure 3.9 the services are color coded just as in the figure 3.7 denoting their functional area or the zone. In this message broker architectural pattern, the message broker acts as the hub in the center. As depicted, there is no direct integration of components except for through the message broker. A message crosses into different zones (color coded regions) based on the wiring done at the message broker. One of the biggest advantages of using the RabbitMQ message broker for this purpose is that it can act as a cluster enabling the cloud service to meet its operational demand.

As the figure 3.10 shows, the messages emitted by each micro service are events identifying a business scenario. The broker has the responsibility to route it to the appropriate receiver. The 'routing key' feature, which is provided out of the box by the broker enables this dynamic wiring.

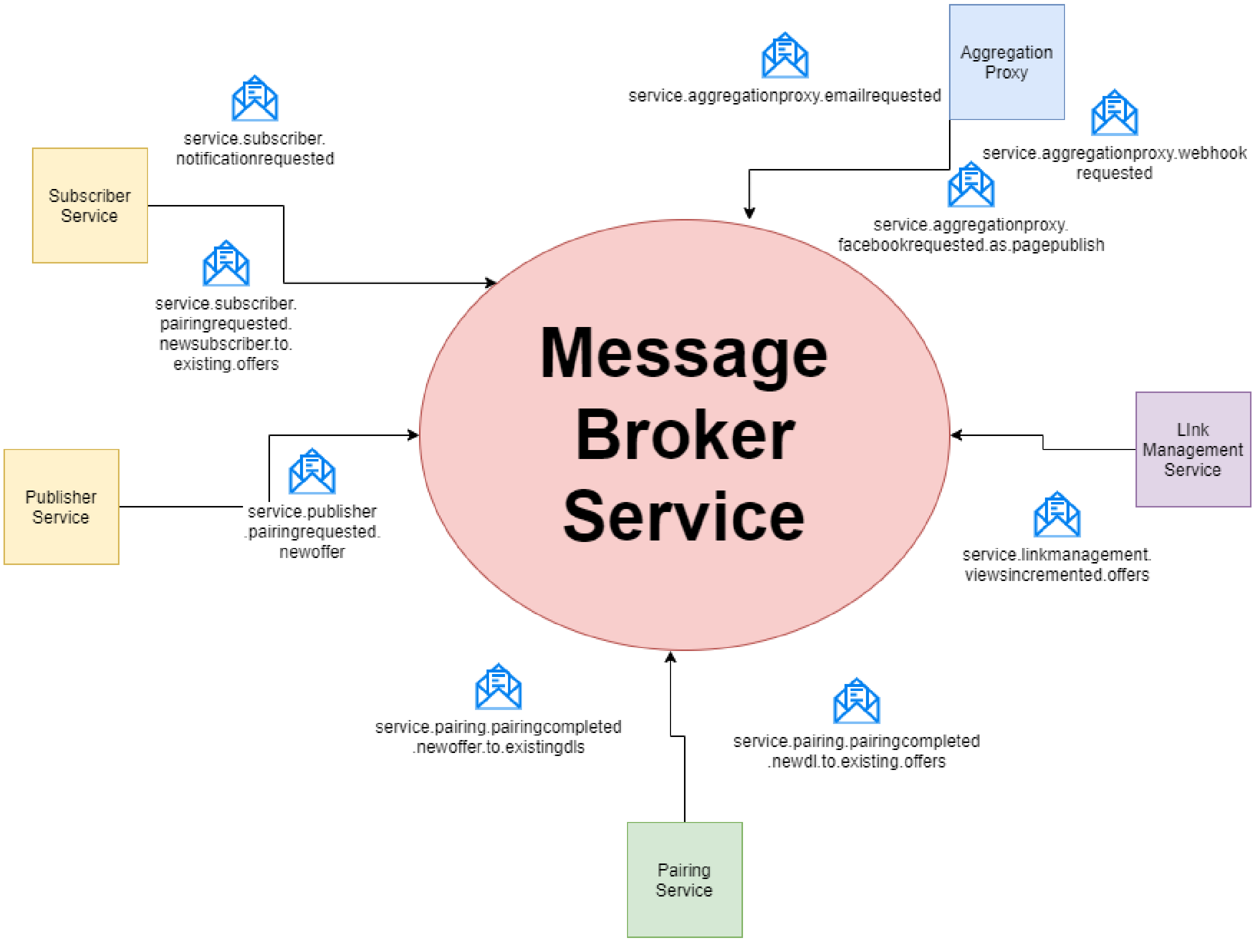


Figure 3.10 - The context diagram of the message broker with the messages it receives as events.

Master-Slave Architecture for Heavy Duty components

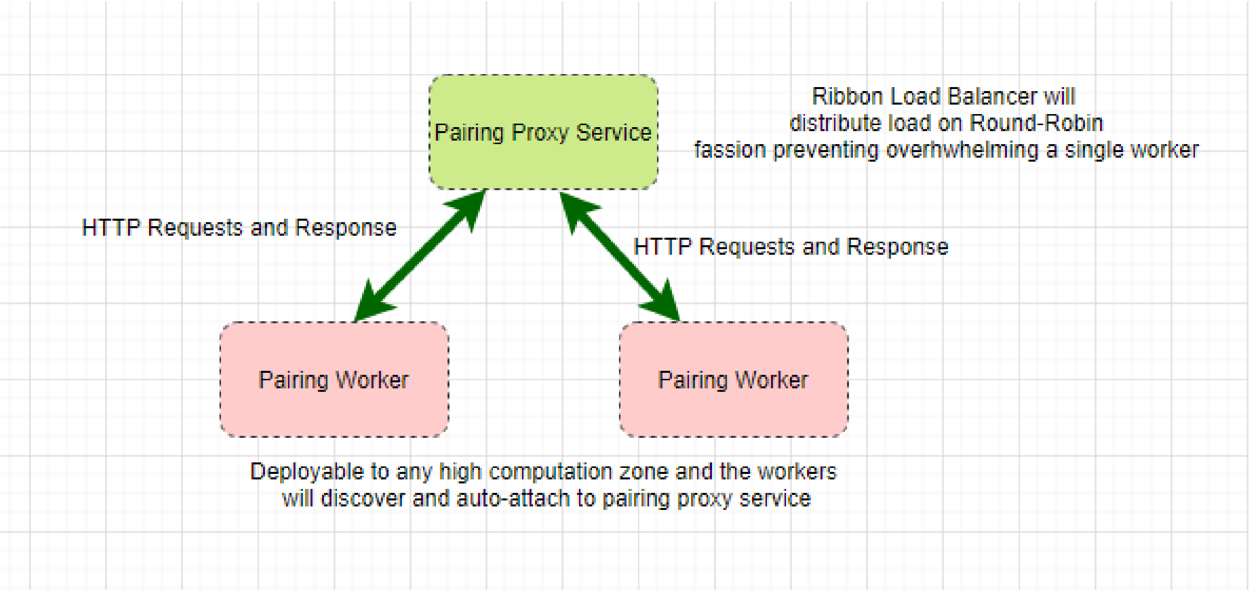

Pairing Proxy Service
Ribbon Load Balancer will distribute load on Round-Robin fassion preventing overhwhelming a single worker
HTTP Requests and Response
HTTP Requests and Response
Pairing Worker
Pairing Worker
Deployable to any high computation zone and the workers will discover and auto-attach to pairing proxy service

Figure 3.11- Master-slave architecture set up

As shown in figure 3.11, the master slave architecture suits best for this purpose more than any other. Primarily because of its flexibility in scaling as the workers can be spawned at will and placed in any platform powerful enough to provide the required computation power such as Amazon's EC2 instances or a Docker container. The components, no matter the location will attach and subscribe to the proxy service via Eureka discovery component. The master service will communicate through HTTP calls using a Ribbon client-side load balancer, which will dispatch the requests in a round robin manner without overwhelming the workers with a bloated load.

Multi-tenant architecture

In order to facilitate the multiple business domains, the system houses a multi-tenant architectural behavior. This is achieved by loading up a configuration file for each of the business domain, which dictates the interpretation logics and behaviors. A sample domain configuration file is given in the Appendix D.

Event Driven Activities and message passing

The architecture features a significant even-driven processing and message passing capabilities. Such event driven design was used to leverage a high degree of decoupling between the components, which allows adding more events to be handled in the future building more functionality. Each event can be treated independently, and its payload can be acted upon. In figure 4.4, the emitted messages by each service are events organized based on a domain hierarchy.

For example: “Service.pairing.pairingcompleted.newoffer.to.existingdls” can be illustrated as the pairing service has completed its new offer to existing distribution list mapping. It can be noted that this routing key follows domain hierarchy allowing easy configuration.

Joining hand in hand with event driven design is the message passing technique. This means the participating systems coordinate and maintain the interactions between them by passing messages between them. This enables the system to scale up easily and allows better maintainability because of the ability to modify the functionality seamlessly.

Model View Controller pattern and REST architectural style

Some of the services expose REST based interfaces, which are implemented based on the Model-View-Controller architecture. The REST services allow building a simple inter-service HTTP communication strategy rather than resorting to complex inter-process communications such as RMI or SOAP. The MVC style allows the separation of views from the business logic.

Subscription Service and the Publisher Service make use of this architecture exposing several REST based interfaces to the end users.

3.6 Application Design Philosophy

Paged Database Access

The application expects large bulks of data coming from its users, consumers in particular who subscribe to the cloud to listen for the offers. As such, full blown data reads into the memory can crash the system and cause catastrophe. As a remedy, the application makes use of paginated data access from the MongoDB stores.

No SQL Data

The application makes use of MongoDB as opposed to a SQL database due to the nature of the operation of the application. This means the application needs fast data reads and the structures of the data can vary at any given moment. Therefore, it’s difficult to comply to hard and fast schema as required by SQL database. Also, since the system makes use of message passing, MongoDB’s JSON based document storage capability stands as a good solution to store the messages in their natural form.

Dependency Management

The application was built to quickly adapt new changes and employ technologies that are best suited for the purpose. As such the seamless integration of the technologies and the robustness thereafter Is paramount.

Maven is used to integrate dependencies to the application. All of the services and libraries developed for this application uses Maven as a backbone to build and deploy the dependencies. In addition to that, Spring Dependency Injection is used to create instances of the classes as opposed to hard coupling of the dependencies ensuring a right inversion of control.

3.7 Techniques for performance boost

Distribution Lists

This application employs the concept of distribution lists to classify the disparate consumer preferences into groups. Each group has an identical set of preferences, essentially meaning the group of preferences listening to the same set of offers. This greatly improves the performance of the system, specially with the maturity of the system. It also helps the better manageability of data. An example is when two consumers set the preferences as “Toyota Prius with price<5500000” , both the preferences are classified into the same group by obtaining a hash representation of them.

Browser Cached pages and checksums

Some responses to the browser requests from the end user are forced to be cached by the application in an attempt to reduce the hits and safeguard the availability of the application. For example when the end user (consumer in this case) clicks on the link to obtain the real market offer, the browser makes a request to the Link Management Service, which produces a self-submit form, which will eventually redirect to actual location where the market offer resides. This self-submit form also makes a simultaneous request to the Notification Proxy Service, telling it to increase the ‘view count’ of the offer. However, triggering this process every time the end-user clicks on the link is a costly operation, prompting for a solution, which is a web cache.

The web cache will cache the self-submit form permanently until forcibly removed by the server. Therefore, the subsequent clicks on the offer link will be handled by this cached page and they will never reach the backend servers improving the efficiency tremendously.

The same process uses checksums in the form of timestamps and hashes to verify the validity and integrity of the request.

3.8 Coding Design

There are number of design patterns that have been employed with the purpose of improving the maintenance and also the performance.

Singleton Pattern

Most of the classes designed to offer a generic service in the application are singleton classes. This is because these applications do not maintain any state for a given context. Among these some service classes interface with resource consuming third party applications such as message queues and databases, which are inevitably designed to be singleton to prevent resource contention.

Proxy Design Pattern

Proxy is a notable design pattern used to abstract the complex processes of a given class and simplify the interface used. An example use case from the application is the creation of a class known as Paginator Proxy to provide paginated access to MongoDB, as opposed to using the direct connection to fetch all of the documents at once, potentially burdening the system memory. The proxy fetches page by page in a seamless manner, even though the client is not aware of it.

Façade Design Pattern

Façade is used to provide a simple interface to the complex subsystems that are invoked by each respective service. The façade classes abstract the complexity of the interfacing with the subsystems. A notable example in this application is the message broker façade to communicate with the message queues.

Factory Pattern

Factories are used to instantiate the objects based on a criterion. The factory class knowns the type of object needed to handle the given scenario while the client is unaware. Such factories are used in the core data module developed in this application which decide whether to instantiate an executive query handler or a nonexecutive query handler.

Template Design pattern

Template pattern is used in the aggregation proxy to define the pipeline process in which the order of execution is of utmost priority. Therefore, a future developer could override the process without affecting the order of execution.

Interpreter design pattern

This pattern is used to delegate the interpretation of a statement down to the elementary level interpreters who can handle the type of statement. This pattern is used to generate the criteria expression for the distribution lists.

Builder Design Pattern

Builders are used throughout the project to construct payloads and other objects which are built in with complex inputs and conditionally.

Service Locator Design Pattern

Service locator is used to load the configuration files for a given domain, which looks up a cache to load first and only the absence in the cache will force the system to load from the database, driving the performance up significantly.

3.9 Testing approach

The application compels to carry out certain categories of tests. Primary test strategy being the unit tests, which will be written for all of the complex functionalities, specifically the ones that execute critical processing. However, the less trivial functionalities will not be unit tested due to the time constraints.

Component and integration testing will be carried out in the latter stages of the project. Each service will be treated as a component and will be subjected to individual testing. Upon completion a full integrated test will be performed on the entire application.

Beta testing will be an optional testing to measure the success of the application in terms of the user experience. This will be carried out if the time permits.

3.10 Authentication

The system is built on OpenID authentication provided by OAuth 2.0 client credential flow. This authentication mechanism was chosen because the OpenID provider certifies the user details provided. For example, the email address and other data are verified. Also, the system was built in such a philosophy that anybody having a valid email address can use it without any limitations. There are many OpenID providers such as Facebook, GitHub and Twitter, but for the current state of the system only Google is enabled.

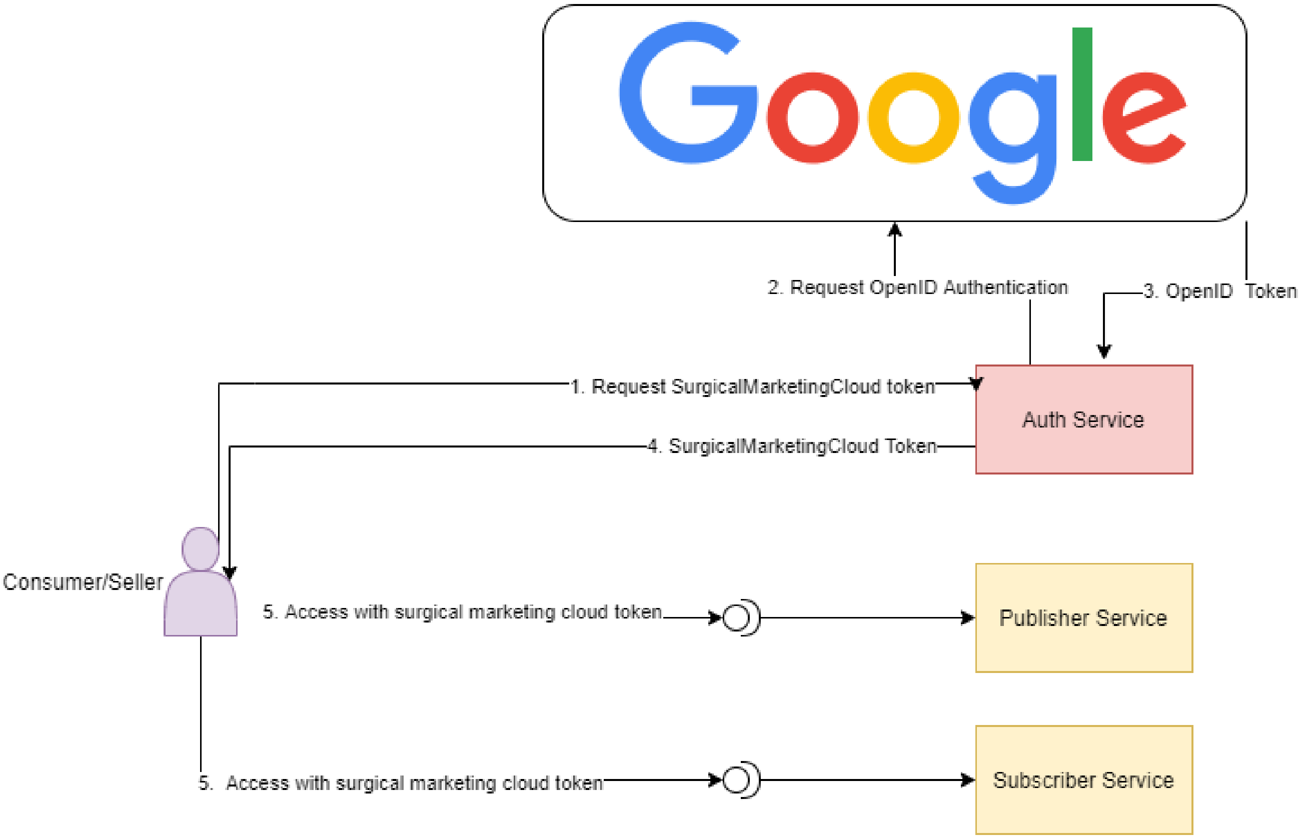

Google
2. Request OpenID Authentication
3. OpenID Token
1. Request SurgicalMarketingCloud token
Auth Service
4. SurgicalMarketingCloud Token
Consumer/Seller
5. Access with surgical marketing cloud token
Publisher Service
5. Access with surgical marketing cloud token
Subscriber Service

Figure 3.12- OpenID authentication in Surgical Marketing Broker

3.11 Comparison of alternative architectures

The initial investigation led to a type of architecture which is based on the polling concept where a central scheduler picks up the offers registered on a predetermined interval and runs the dispatching process. However, this architecture was abandoned in favor of the event driven model due to its less scalable nature under heavy loads. Following is a snapshot of the

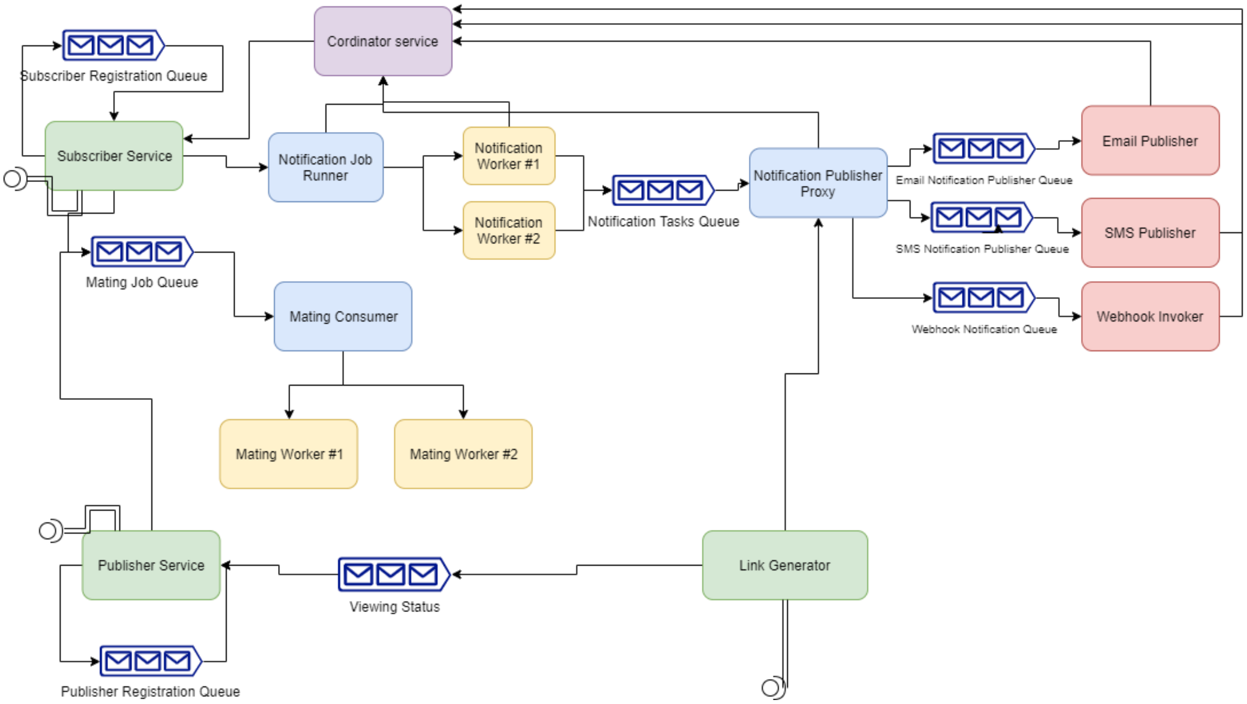


Figure 3.13- Alternate architecture

3.12 Software Development Process

The project is to follow an evolutionary- prototyping approach rather than a strict waterfall model due to its inherent exploratory nature. Most notably being in the areas of algorithm and process development for components that are workhorses of the application. This approach of developing an evolutionary prototyping reduces the risks of having to deal with complex issues in the new processed developed. An example case is the development of a 'pairing service' prototype to test the idea of evaluating consumer preferences using the expression evaluators.

Chapter 4. Evaluation

Upon reading the thesis in the prior sections it is understood that the project's objectives extend well beyond its business objectives. Therefore, the evaluation aims to provide how well the project has achieved all the categories of objectives.

4.1 Evaluation of meeting the primary objectives

Reiterating the main problem statement, 'delivering the **right message** to the **right consumers** at the **right time** through the **right medium'**; it is safe to claim that the solution answers this. The system works in an event-model where the consumer requests are serviced based on the order of receipt, meaning they would be served in sufficiently short amount of time assuring the 'right time' property.

In the pairing flow, the market offers are classified in a custom defined process and grouped into distribution lists, each of which contain the marketing offers together with the intended recipients assuring the property of 'right message'.

In the notification flow, the distribution lists are picked up based on the preference the consumer registered and the marketing offers are forwarded assuring the 'right consumers' property.

Finally, the offers that are forwarded in the notification flow will exit the system and arrive at the consumer through a notification proxy, i.e. email proxy, Facebook proxy, which was chosen by the consumer as the preferred medium to receive once again upholding the property of 'right medium'.

4.2 Evaluation of meeting secondary business objectives

Displaying the view count was achieved by means of using a combination of HTTP caching, HTTP redirection coupled to an event queue, together forming the concept was 'envelope' developed for the solution. This mechanism securely delivers the view count to the merchants who would like to know the view count.

4.3 Evaluation of meeting technical objectives

One of the main goals of the project was to deliver a technical excellence in terms of addressing the key performance areas that have the highest impact on meeting the business objectives. With the performance forecast outlined in the analysis section it became apparent that the consumer preferences are predicted to grow exponentially. As a fruitful response, a resilient and scalable architecture was developed which is agile enough to the increasing load and the demand for new functionality and interoperability. For the sake of evaluation few major architectural achievements can be highlighted. The nature of the operation where users interact with the system and produce requests in the form of messages, which needs to be processed in the order of arrival has best been addressed with the event-driven design cum message passing in the solution, creating a smoother and predicable operation. Message broker architecture allowing dynamic workflow pluggability and extensibility, which makes the application future-proof. The multi-tenant system design providing an opportunity to accommodate enormous number of domains, which again solidifies its usability in the future. The high processing components based on master-slave architecture intended to make the application robust and resilient in the face of a high load and maintain its throughout by allowing the heavy-duty components to be multiplied at will and deployed to powerful processing platforms. Furthermore, efficient techniques were used to make the solution modular and expandable such as design patterns and also, resilience measures such as database retrieval pagination to prevent memory run outs in the wake of a large dataset. Also, offline batch processing capability will provide

4.4 Evaluation of meeting process objectives

Serving the business objectives while keeping the performance with the technical objectives described above also means that the system follows a predictable, improvable, adaptable and clear process which guarantees future improvements and applications. Hence, a new protocol was developed with defined steps and a philosophy. Four key workflows were defined namely, registration, pairing, notification and acknowledgement, which clearly exhibit the nature of processing at first sight. These separable workflows contain innovative building blocks that were developed specifically to address the given problem. The primary building block being the 'distribution list', which houses the semantically equal preferences with paired offers which allow clear location of the target recipient consumer group as well as the relevant offers in the notification workflow. Also, it's noteworthy the registration workflow contains a normalization procedure, which translate the domain-specific preferences to a domain-neutral representation in Boolean form, a powerful concept making the system adaptable. Also, the system pairing houses a groundbreaking logic of matchmaking by the use of Boolean expression evaluation as opposed to simple if-then checks making the system infinitely extensible.

4.5 Results achieved after testing

In Appendix E, a screenshot of an email notification received after a consumer registration and a preference submission. Appendix E also shows a Facebook post automatically created after another preference submission. Both outcomes showcase the successful solution that was developed. The given mediums have been selected by a consumer as the ideal method for delivery for him and the messages include all that is necessary for him to distinguish the offer. It can be noted that in the email, the 'subject' having the domain the user has selected, which is vehicles in this case. The link that both the Facebook post and the email is an envelope, which loads an html page from the system, which runs a procedure to invoke the acknowledgement of the message, finally making a self-redirection to the target offer.

A critical evaluation would render certain objectives which were discovered latter at the project with architectural refinements but could not be implemented due to the time constraints. As depicted in the figure 3.2 in the design section, the record dictionary mechanism would severely cut down the data fetch iterations and save enormous disk I/Os. However, such a modification would compel to carry out many changes and lot of testing, which would not be viable option in the interest of delivering on time. Therefore, the project was implemented with a standard iteration mechanism to fetch offers and distribution lists. However, in order to satisfy the requirements of growing load, this should be treated as a tech debt for future implementations.

A key lesson learnt during the conceptualization, design and development of the project is that more time should be extended to architectural revisions. The primary reason being the fact that rather than a simple implementation project set out to cater a set of business requirements, this project had further objectives of creating a protocol to address the problem together with a supportive architecture ready to meet the future demands. Hence, the project was executed on an evolutionary prototype model as mentioned in the design chapter, which inevitably requires earlier start in order to meet the tight deadline.

The project had favorable comments from the users who participated in beta testing who tested the usefulness of the solution against the consumer needs. The remarkable feedback given was the ability of the system to do accurate matchmaking for right offers and distribute it in the chosen notification media. Also, the ability of the consumers to freely specify the criteria and domains were highly praised.

Chapter 5. Conclusion

The problem identified is a clear gap in the field of digital marketing which has not been addressed. This gap was further substantiated by the preliminary survey results, in which the sample answers demonstrate that people would love to use the solution as a method to be notified when their wishful item becomes available. Based on the nature of the problem and the potential extent of the use of the solution, it was decided that the solution best be a cloud service. The cloud service was intended to encompass certain characteristics for it to be usable under certain potential conditions which called for a supportive architecture that has been developed successfully. Additionally, a clear cut, well defined process has been developed as a baseline for future improvements as well as to facilitate communication and understanding and achieving that is a major success of the project. Finally, of the utmost importance are the meeting of the business objectives, which is testified to be successful with the provided test results in the Appendix E.

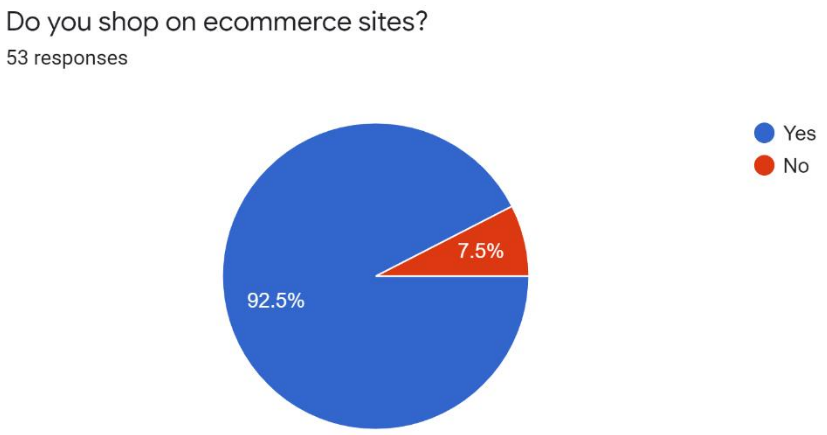
On a futuristic note, the project can be complemented with more features as enhancements or a remedy to the deficiencies. Few of the such points are noted below.

As an immediate improvement the project could be integrated with more types of social login such as Twitter and Facebook. Also, it could include an SMS proxy, essentially connected to the set of notification medium proxies and forward the SMS requests. Also, the portals could be made mobile friendly as this application finds its best use through the mobile phones.

Long term technical improvements can also be suggested such as indicated below. The application currently understands the preferences and offers through a specified structure, i.e json or form parameters. A key improvement in the future would be to enable Natural Language Processing (NLP), which captures casual text and converts it to an understandable format. Another very important technical upgrade would be enabling a token-authenticated API integration with its portals where a client application could become the merchant as well as the consumer interacting with the system and receiving services. Another, future technical improvement, in the long term would be to introduce Ribbon load balancers, as prescribed in the architecture depicted in figure 3.7, which would perform a round-robin based selection of servers providing an even load across all the instances.

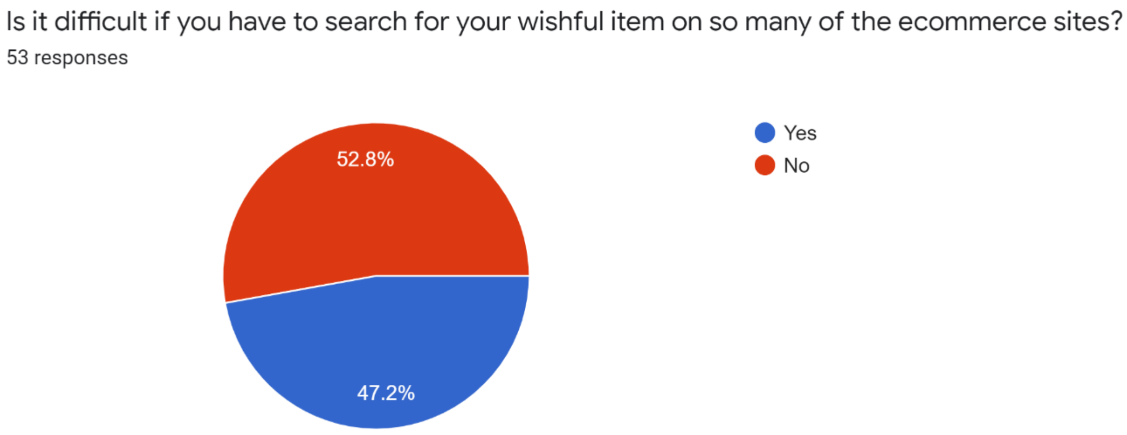
A conceptual improvement could also be suggested which would position the solution in a unique place. Currently, the solution holds data in its possession, namely, preferences and offers, which inevitably means that it has to maintain those. A conceptual improvement would be to designate the solution as purely a stateless machine providing the processing as a service, which connects to outside data sources to retrieve the offers and the preferences. Such a solution may compel the designers to create fine-grained data specifications and secure APIs to enable seamless data pushing from third party systems to the solution.

Appendix A-Feasibility Survey Results



Do you shop on ecommerce sites?

53 responses

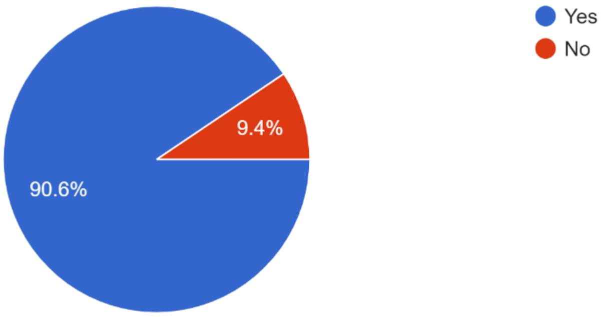


Is it difficult if you have to search for your wishful item on so many of the ecommerce sites?

53 responses

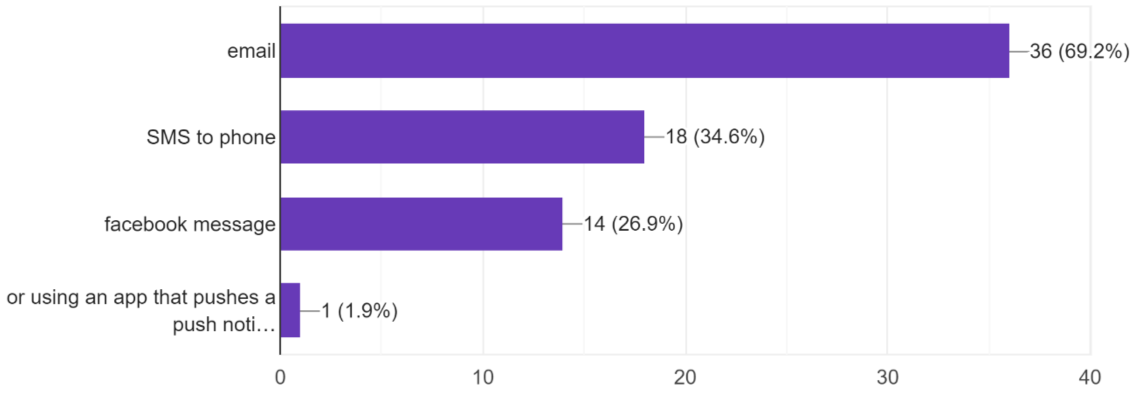
Are you happy if you were notified when an item matching your criteria appears on some ecommerce site?

53 responses

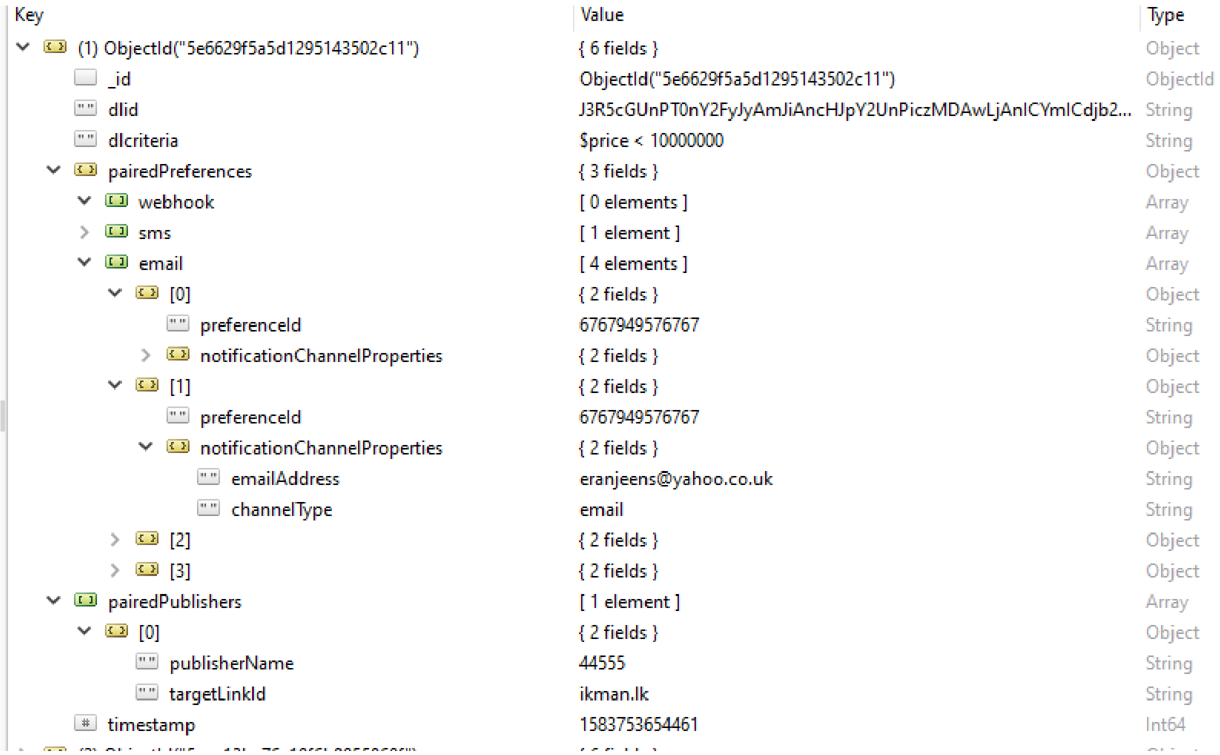


If yes to above, then in what method would you like to receive notifications?

52 responses



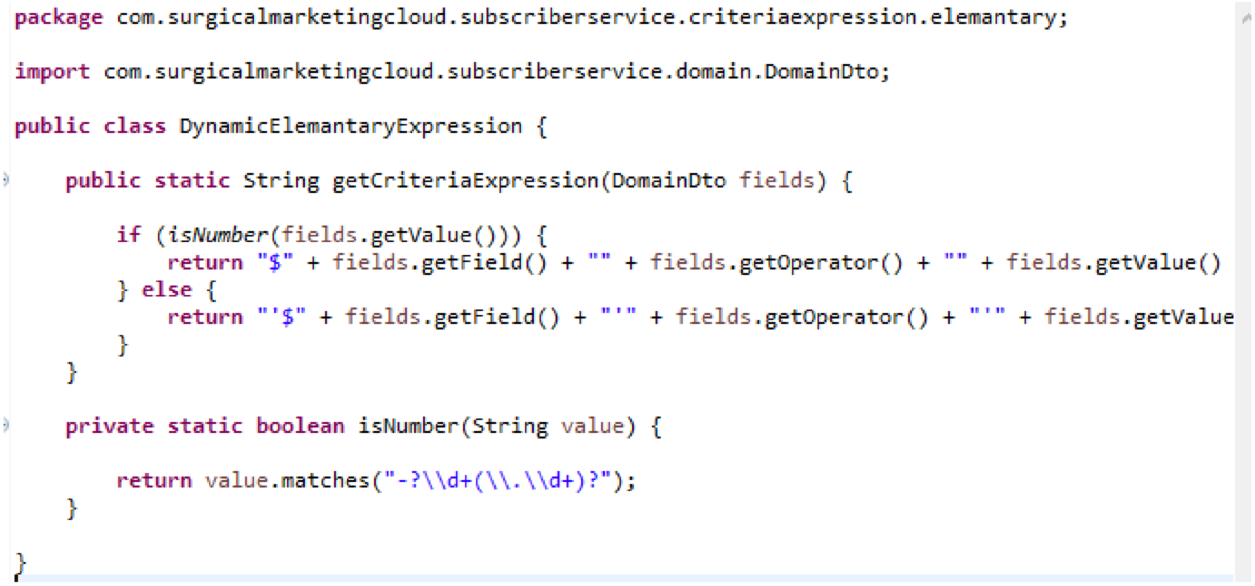
Appendix B-Sample Distribution List



Key	Value	Type
(1) ObjectId("5e6629f5a5d1295143502c11")	{ 6 fields }	Object
_id	ObjectId("5e6629f5a5d1295143502c11")	ObjectId
dlId	J3R5cGUnPT0nY2FyJyAmJiAncHJpY2UnPiczMDAwLjAnICYmICdjb2...	String
dlcriteria	$price < 10000000	String
pairedPreferences	{ 3 fields }	Object
webhook	[0 elements]	Array
sms	[1 element]	Array
email	[4 elements]	Array
[0]	{ 2 fields }	Object
preferenceId	6767949576767	String
notificationChannelProperties	{ 2 fields }	Object
[1]	{ 2 fields }	Object
preferenceId	6767949576767	String
notificationChannelProperties	{ 2 fields }	Object
emailAddress	eranjeens@yahoo.co.uk	String
channelType	email	String
[2]	{ 2 fields }	Object
[3]	{ 2 fields }	Object
pairedPublishers	[1 element]	Array
[0]	{ 2 fields }	Object
publisherName	44555	String
targetLinkId	ikman.lk	String
timestamp	1583753654461	Int64

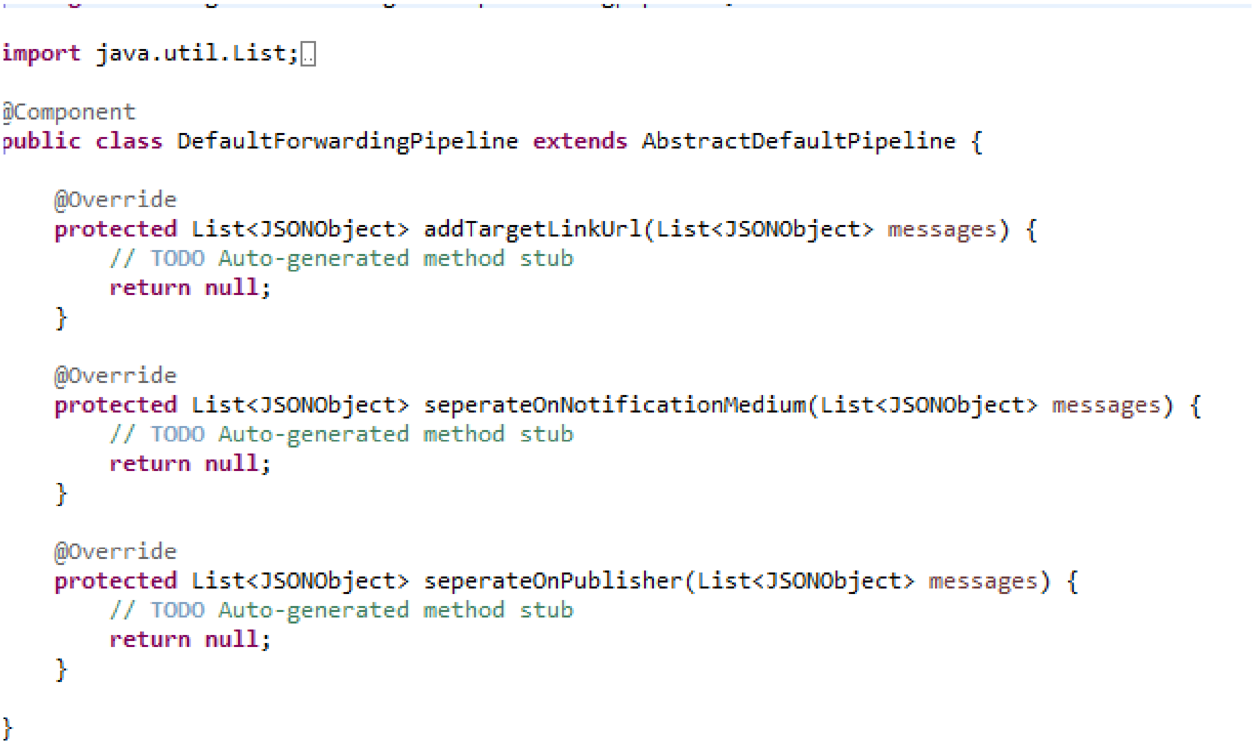
Appendix C-Code Samples

DynamicElemantaryExpression class of the interpreter pattern



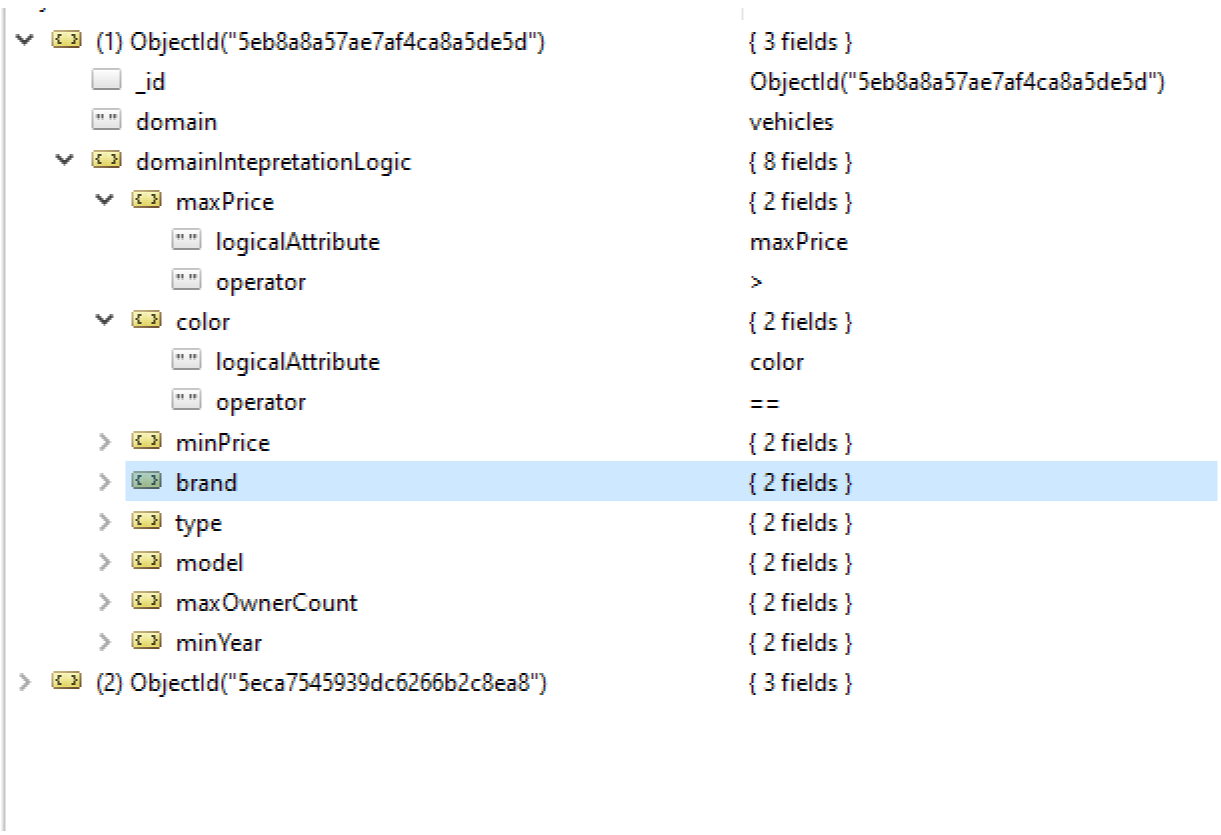
```
package com.surgicalmarketingcloud.subscriberservice.criteriaexpression.elemantary;

import com.surgicalmarketingcloud.subscriberservice.domain.DomainDto;

public class DynamicElemantaryExpression {

    public static String getCriteriaExpression(DomainDto fields) {

        if (isNumber(fields.getValue())) {
            return "$" + fields.getField() + "" + fields.getOperator() + "" + fields.getValue()
        } else {
            return "'$" + fields.getField() + "'" + fields.getOperator() + "'" + fields.getValue
        }
    }

    private static boolean isNumber(String value) {

        return value.matches("-?\\d+(\\.\\d+)?");
    }

}
```

DefaultForwardingPipeline class of the template pattern



```
import java.util.List;

@Component
public class DefaultForwardingPipeline extends AbstractDefaultPipeline {

    @Override
    protected List<JSONObject> addTargetLinkUrl(List<JSONObject> messages) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    protected List<JSONObject> seperateOnNotificationMedium(List<JSONObject> messages) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    protected List<JSONObject> seperateOnPublisher(List<JSONObject> messages) {
        // TODO Auto-generated method stub
        return null;
    }

}
```

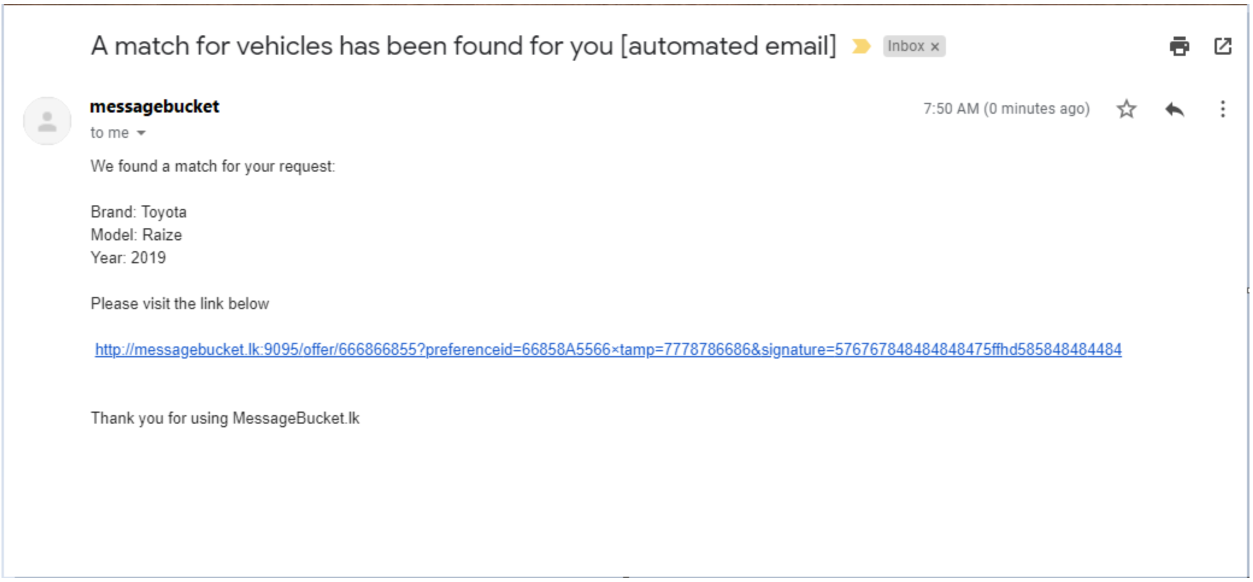
Appendix D-Domain Configuration File



```
(1) ObjectId("5eb8a8a57ae7af4ca8a5de5d")    { 3 fields }
    _id                                     ObjectId("5eb8a8a57ae7af4ca8a5de5d")
    domain                                  vehicles
    domainIntepretationLogic                { 8 fields }
        maxPrice                            { 2 fields }
            logicalAttribute                maxPrice
            operator                        >
        color                               { 2 fields }
            logicalAttribute                color
            operator                        ==
        minPrice                            { 2 fields }
        brand                               { 2 fields }
        type                                { 2 fields }
        model                               { 2 fields }
        maxOwnerCount                       { 2 fields }
        minYear                             { 2 fields }
(2) ObjectId("5eca7545939dc6266b2c8ea8")    { 3 fields }
```

Appendix E-Test Results

Email Received from MessageBucket.lk



A match for vehicles has been found for you [automated email]

messagebucket

to me

7:50 AM (0 minutes ago)

We found a match for your request:

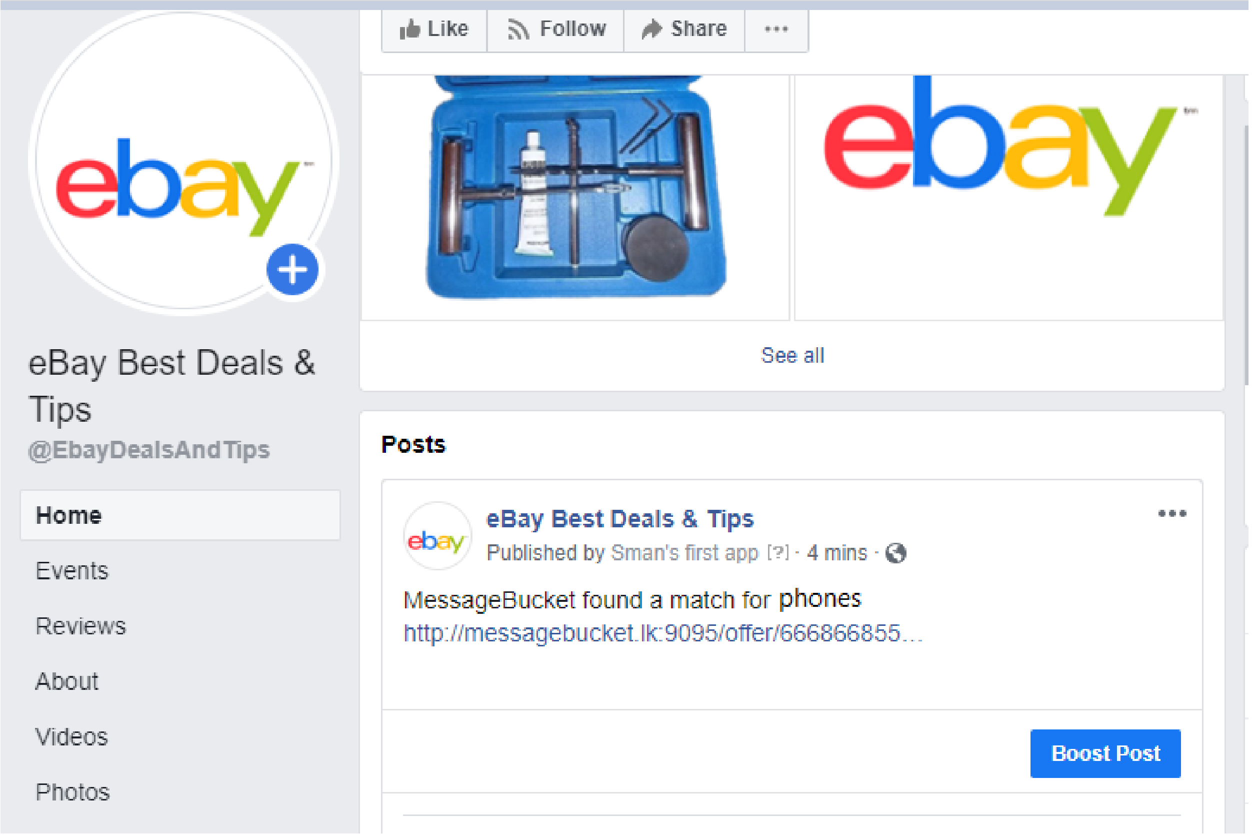
Brand: Toyota
Model: Raize
Year: 2019

Please visit the link below

http://messagebucket.lk:9095/offer/666866855?preferenceid=66858A5566×tamp=7778786686&signature=576767848484848475ffhd585848484484

Thank you for using MessageBucket.lk

Facebook page post by MessageBucket.lk



eBay Best Deals & Tips
Published by Sman's first app [?] · 4 mins ·

MessageBucket found a match for phones
http://messagebucket.lk:9095/offer/666866855…

References

[1] “Introduction to message brokers part 1 apache kafka vs rabbitmq” [online] available:https://hackernoon.com/introduction-to-message-brokers-part-1-apache-kafka-vs-rabbitmq -8fd67bf68566 [accessed: 21st October 2019]

[2]“publisher subscriber messaging” [online] available: https://dzone.com/articles/comparing-publish-subscribe-messaging-and-message [accessed: 26th October 2019]

[3] “effective digital marketing tactics and beyond” [online] available: https://mikekhorev.com/12-effective-digital-marketing-tactics-strategies-2018-beyond [accessed: 11th November 2019]

[4] “best email marketing campaigns” [online] available: https://www.campaignmonitor.com/best-email-marketing-campaigns/ [accessed: 21th November 2019]

[5] “azure publisher subscriber architecture pattern” [online] available: https://docs.microsoft.com/en-us/azure/architecture/patterns/publisher-subscriber [accessed: 21th November 2019]

[6] “Spring Security and OpenID Connect” [online] available: https://www.baeldung.com/spring-security-openid-connect [accessed: 21th April 2020]

[7] “Cache-Control” [online] available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cache-Control [accessed: 17th January 2020]

[8] “Understanding OAuth 2.0 and OpenID Connect” [online] available: https://blog.runscope.com/posts/understanding-oauth-2-and-openid-connect [accessed: 10th January 2020]

[9] “Dead Letter Exchanges” [online] available: https://www.rabbitmq.com/dlx.html [accessed: 11th February 2020]

[10] "How can I delete all cookies with JavaScript? [duplicate]" [online] available: https://stackoverflow.com/questions/595228/how-can-i-delete-all-cookies-with-javascript/595251 [accessed: 9th February 2020]

[11] "Third-party sites & apps with access to your account" [online] available: https://support.google.com/accounts/answer/3466521?hl=en [accessed: 11th February 2020]

[12] "Page Content" [online] available: https://developers.facebook.com/docs/pages/publishing/ [accessed: 15th April 2020]

[13] "The complete guide to using localStorage in JavaScript apps" [online] available: https://blog.logrocket.com/the-complete-guide-to-using-localstorage-in-javascript-apps-ba44edb53a36/ [accessed: 14th March 2020]

[14] "Spring Expression Language (SpEL)" [online] available: https://docs.spring.io/spring/docs/3.0.x/reference/expressions.html [accessed: 13th March 2020]