ShopGuru

An SMS-based Shop Detail Discovery Service

Bandara U.K.J.U.
Department of Computer Science and Engineering
University of Moratuwa
Sri Lanka
bandaraukju.11@cse.mrt.ac.lk

***Abstract*—This research paper describes ShopGuru, a Short Message Service (SMS)-based service developed as an attempt of solving some of the issues faced by customers in discovering the locations, details and availability statuses of shops around them as well as by shopkeepers in communicating their shop details to customers. It outlines the approach followed and major decisions taken during the conceptualization, design, implementation and evaluation of the application, and points out key concerns of the evaluation such as latency and accuracy of the responses, along with suggestions for future improvements in making it more attractive, user-friendly and available to a greater audience.**

Keywords—IdeaMart; SMS; Location Based Services (LBS); geolocation; neighbourhood

I. INTRODUCTION

The purchase-based economy of modern times requires customers to frequently visit and communicate with different shops. Two main problems arising from this model of customer-shop interaction can be identified: 1. A customer may need to know the current status of a particular shop (e.g. whether the shop is open, whether some of the goods are out of stock, etc.) before actually travelling there, 2. A person in an unfamiliar neighborhood may need to find a shop of a particular category, in the close vicinity (e.g. a pharmacy).

If there existed a service via which the shop owner could communicate the 'status' of his shop to its customers (as and when required), it would be quite helpful for both parties. The medium should be asynchronous so that the shopkeeper will not have to get involved in every customer query, as in the case where the customer gives a phone call to the shop, and simple and versatile enough to be used anywhere, anytime.

ShopGuru is an SMS-based service app that attempts to address this requirement under minimal commitments and infrastructure requirements from both the clients' and the shop owners' perspectives. In addition to this primary function, ShopGuru plays the role of a location-based shopping guide, where a customer can send a query to the application regarding the type of shop he/she is looking for, and receive a list of shops and addresses located closest to his/her current location.

II. BACKGROUND AND MOTIVATION

Most of the shops currently in Sri Lanka still do not have websites or other public communication media via which they can communicate with customers without real-time human intervention. Voice communication (direct or over phone) is inconvenient in this regard as it is synchronous (both parties should be involved simultaneously) and unicast (each customer should be contacted separately). The next best choice is SMS communication, provided that it is made asynchronous and multicast (shop owner updating the shop status once, and multiple customers being able to view it whenever required).

While it is possible to use a smartphone application for communication, it should be kept in mind that the majority of mobile phone users in Sri Lanka are still using regular ('feature') phones [1, pp. 6-7]. Furthermore many of the small- to medium-scale shop owners are accustomed to using feature phones primarily due to their low price and ease of use. Thus, a bare-bones mobile service would have a larger customer reachability than a smartphone app in this case.

Dialog [2] is one of the leading mobile service providers in the country [1, Fig. 1], and its IdeaMart developer Application Program Interface (API) is available free of charge for any app developer. If an application is built (with proper decoupling and modularity features) to work with the IdeaMart APIs, it can eventually be adapted to run with any other mobile subscriber's API specifications (e.g. Mobitel [3] also has similar APIs, but they are available only for commercial use and it is necessary to register and sign an agreement in order to start using them).

Based on the above facts it was decided that the intended application should be an SMS-based app compatible with all mobile phones, utilizing IdeaMart APIs so that it would be accessible for all Dialog subscribers, covering a significant portion of the mobile phone user community [1, Fig. 1].

III. LITERATURE REVIEW

SMS is widely used as a communication medium among the general public, mainly due to its simplicity, reliability and speed. With the massive growth and inadequate infrastructure in many countries, voice and where available IP are just not feasible unless there is massive investment to bring up the networks. SMS will get through even when the 'network is busy' for hours [4].

As a result, SMS is quite popular as a communication medium not only in developed countries but even in underdeveloped ones in Africa, and in developing ones like Sri Lanka. For example, in Zambia, a farmer's union uses SMS messages to distribute market prices while in Uganda, Malawi and Benin, health education messages are sent via SMS [4]. SMS query-response systems are widely used in academic

contexts as well, such as at Kenyan universities for checking exam results and fees balances, and reporting emergencies [5].

Many researchers have pointed out the possibility of SMS being used as marketing tools. For example, as with the internet, SMS advertising can introduce shopping in a specific store, or driving in close proximity to a retail outlet [4]. Although ShopGuru is more of a service, a potential is offered to shops to market their products via their status updates.

Mobile phones fall into two major categories as feature phones and smartphones. The former ones have a limited, fixed set of functionalities, whereas the latter ones generally have more advanced features and offer more customizability and expandability, often achieved via externally installed applications which are named 'apps'. Although SMS applies for both classes in pretty much the same way, location based services can be quite diverse as smartphones often support Global Positioning System (GPS) and wireless network (Wi-Fi) based location identification. GPS and Wi-Fi typically provide more accurate location results than regular mobile network based locations [6].

In fact, GPS- and other kind of location-based recommendation and guidance apps are now in plenty. Popular smartphone apps like Google Maps, Google Places, Foursquare and ShopKick can be cited as examples [7]. However, such apps are not widely available for feature phones, especially those which do not have GPS or mobile data (General Packet Radio Service, GPRS) capabilities. Hence a majority of mobile phone users in countries are still confined to the use of lower-accuracy network-based LBS, and do not enjoy the benefits of many of the advanced location based services (e.g. apps).

Location-dependent Data (LDD) has great promise for mobile and pervasive computing environments [6]. Location based application often provide its services (LBS) by using location based queries (LBQs). LBQs can be categorized into four types; range queries, nearest neighbor queries, navigation queries and geo-fence queries. For example, nearest neighbor queries deal with the entities closest to the point of query origin (similar to location-based queries in ShopGuru) [6].

Location-based services, such as the ability to provide features like weather forecast, restaurants guides, hotels maps, address finders, and traffic update have been cited as consumer friendly features of new media [4]. For example, in a location based shop query system, the user just searches for a particular grocery shop, and the user location is automatically detected by the mobile phone and encapsulated along with the query. This enhances the way the query is processed in the back-end, as based on the location, the query works only on a subset of the back-end database of shops [6]. It also returns a response that is more meaningful for the user, as details of a nearby shop are quite valuable than those of one which is several miles away.

Although location-based recommendation and assistance systems are quite popular in the world today, such systems have not been widely adapted in Sri Lanka when it comes to the mobile service provider. For example, even under Dialog's IdeaMart app developer campaign [8], there are only a few apps such as Mega Mind [9] and WikiLanka [10] that offer location-based support and recommendations.

In conclusion, considering the potential benefits that can be enjoyed by the general mobile phone user population (including both feature phone and smartphone users), and the lack of such services in the current Sri Lankan mobile service field, the development of a service like ShopGuru can be considered as a useful, timely and profitable venture.

IV. Outline of the Solution

Services provided by ShopGuru are based on four basic user activities (operations) which are described in detail follow:

A. *Shopkeeper Subscription*

Shopkeepers can subscribe to the service using a Dialog mobile phone (providing shop name, location (address) and shop category (e.g. pharmacy, bakery, grocery etc.)).

B. *Shop Status Updates*

Subscribed shopkeepers can update the statuses of their shops (e.g. "open", "closed", "closed from 10AM to 2PM" etc.) at any time (via SMS). If the shopkeeper has not updated his status within a predefined time period, the service notifies the shopkeeper to do so. Each update has a maximum validity period of one day, which can be configured by the shopkeeper.

C. *Customer Queries*

Any regular subscriber can send an SMS query to the service, to retrieve the best matching status results for a given shop category, in SMS form. Relevance of the results can be improved using Dialog's LBS API, and Google Maps APIs for decoding textual addresses to geographical coordinates. For example, the query "pharmacy" might return up to 3 status results for pharmacies, in the form "Pharmacy A [location]: open, Pharmacy B [location]: closed".

D. *Customer Registration under Shops*

Additionally regular subscribers can 'register' under a given shop via SMS. Results related to such registered shops would appear with higher priority in the response, whenever applicable. For example, if a user has registered under the hardware store X, if he sends a "hardware" query from a location in close proximity to X, the status of X would appear at the top of the response SMS.

V. Design Considerations

A. *Basic Architecture*

Although the service uses a client-server architectural pattern, no specific client interface is available because the SMS mechanism is handled by Dialog, the mobile network service provider. A layered approach is used to achieve better levels of modularity, maintainability, reliability and extendibility. The system interfaces with IdeaMart's Idea Pro [11] web APIs, and Google Maps Geolocation API [12] for address manipulation.

ShopGuru exhibits a mix of client and server characteristics, this is due to the fact that it acts as a server for its subscribers and at the same time as a client of Idea Pro (IdeaMart) and Google Maps APIs. ShopGuru also communicates with a MySQL database which, although hosted on a different MySQL server, is part of the application itself.

Clients of ShopGuru indirectly communicate with it via Dialog's service architecture, using SMS and LBS protocols. ShopGuru utilizes HTTP (and HTTPS) to communicate with external APIs.

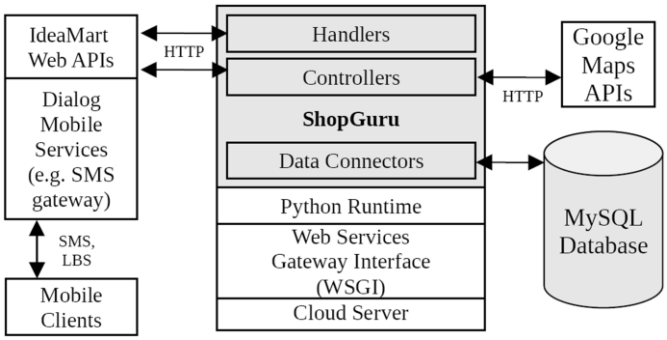

Fig. 1. Basic architecture of ShopGuru application.

B. System Tiers

Since ShopGuru is more server-oriented and has no visible client interface, the traditional 3-tier, presentation-application-data access layered architecture is not directly applicable. Hence a slightly different approach has been followed in modelling the tiers of the application.

Handlers (in Fig. 1) interface with client requests (queries). Since there is no customized interface, these merely listens for incoming queries and passes them to lower layers, and transforms information generated by lower layers into more human-friendly responses to be sent to the clients. Hence this can be considered partially as a presentation layer and partially as a communication layer.

Controller tier (in Fig. 1) is analogous to the business logic layer in traditional 3-tier architecture. It processes client requests received via handlers to produce required responses, during which it communicates with the database (via data connectors) and with external APIs (LBS and Google Maps).

Data Connector tier is analogous to the data access layer of traditional 3-tier architecture. It is responsible for entity management (retrieval, persistence and discarding of entities) and the consistent handling of database queries via a persistent database connection.

C. Performance Metrics

In order to serve multiple client requests concurrently, a client handler factory implementation is used to spawn new client request handlers whenever new requests are received. Nevertheless, singleton implementations like the database connection (necessary for avoiding concurrent updates and maintaining database integrity) can sometimes induce a bottleneck effect. Hence the development of an efficient database schema and set of queries is crucial for maintaining the application at desired performance levels.

VI. IMPLEMENTATION

A. Package Hierarchy

ShopGuru source [13] contains several packages for providing different operational and maintenance functions.

TABLE I. PACKAGES IN SHOPGURU SOURCE

Package	Contents
adapter	classes used for interfacing internal web request handlers with external web server API classes (e.g. Google App Engine software development kit (GAE SDK) vs. web services gateway interface (WSGI) server)
config	app configuration parameters (e.g. app ID, password, API endpoint URLs, database connection parameters)
exception	exception hierarchy of the app, spanning from a base *ShopGuruException*
launcher	modules for launching the app on different platforms (e.g. GAE SDK, WSGI)
lbs	handler module for LBS request handling
maps	Google Maps Geocoding API handler
model	entity model, entity handler and datastore connection modules for the app; contains submodules *db* for database-level operations (with a MySQL connector and data manager) and *entity* for entity classes used by the app
sms	SMS receiver, sender and parser modules
task	modules related to handling of user queries

B. Entity Model

Following are the distinguishable entities for the app:

TABLE II. ENTITIES IN SHOPGURU APP

Entity	Description
Shop	contains shop details (phone, name, location, registration time, category), current status and last status update time (for status expiry notifications)
Customer	contains customer details (phone, name, location)
Subscription	subscription of a customer for a shop; contains references to the shop and customer, time of subscription initiation, and time at which the last query related to the subscription (i.e. from the particular customer to the particular shop) has been made

In addition, two SMS-related entities were introduced for logging and statistical purposes: *IncomingSMS*, representing an incoming message (query or otherwise), and *OutgoingSMS*, representing an outgoing SMS response. Each has an associated phone number, timestamp and message content.

Some non-persistent entities are defined for internal tasks:

TABLE III. NON-PERSISTENT (HELPER) ENTITIES IN SHOPGURU APP

Entity	Description
Message (BaseSMS)	parent of *IncomingSMS* and *OutgoingSMS*; message persistence is handled via polymorphism, where each type gets persisted in the appropriate table
Location	a geographical location inside the application; persisted as part of a customer or shop entity
Query	a user query; in addition to the original message entity, contains query-specific parameters (e.g. for a shop registration, the shop name, address and category)

C. Basic Algorithm

The following sequence of steps takes place when a new SMS is received by the app:

1. Message arrives on the server, and it is routed to the SMS receiver module as a POST request.
2. A new *ClientHandler* (class for representing and handling a client request) instance is created using a

ClientHandlerFactory, and launched. *ClientHandler* is a thread on its own, so the SMS receiver gets released for handling further queries without remaining blocked.

3. *ClientHandler* first persists the message, and then passes the message into a parser module.
4. The parser parses the message (in a sequential manner), and returns a *Query* object. In parsing, case insensitivity of keywords is honoured but actual parameter values (e.g. customer and shop names) are taken as case sensitive. Also, the order of parameters is not strictly enforced ("reg SHOP n:Amila Hardware C: hardware" and "Reg Shop c:hardware N: Amila Hardware " would produce the same *Query* with *type* = *SHOP_REGISTER* (shop registration), *name* = "Amila Hardware" and *category* = "Hardware")
5. Each subsequent parser method thoroughly checks the incoming message for illegal parameters (which is currently limited to too long and too short parameters, such as very short or very long names). If such a discrepancy is faced, an appropriate error and instruction response is thrown, which is subsequently caught and handled by the *ClientHandler*.
6. The *Query* object is processed by the *ClientHandler*. It has a series of dedicated methods for each type of query. Each method returns a response string that can then be composed into an SMS and sent either as acknowledgement or as query results, to the client. If any of the methods fail, an appropriate exception is thrown to the calling (driver) method, which then generates an appropriate error or instruction response. *ClientHandler* also contains a series of predefined message templates that can be used for response generation. In case a particular response has to be modified, the programmer simply has to change the message template at the beginning of the file.
7. The returned reply string is composed into a message, and sent to the customer. The response is also persisted in case it might be useful for further analysis.

D. Notable Features

For name-based shop queries, it is necessary to retrieve the nearest set of shops (relative to the customer's current location), sorted by proximity. For this, a Euclidean distance calculation is performed in the relevant database query itself.

For location-based shop queries, shops to which the customer is already subscribed are given a higher priority in the response list, as a potential benefit of subscribing under shops. Due to the high level of decoupling between the entity model and the back-end SQL query implementation, it is not feasible to write a SQL join query to handle this, since it would result in the query specification leaking into the application logic. Hence it had to be handled via a somewhat complicated list intersection algorithm in the application itself.

E. Communication

SMS reception and delivery is handled over standard HTTP, by *sms_receiver* and *sms_sender* modules. SMSs are received as JavaScript Object Notation (JSON) payloads in POST requests to the SMS receiver endpoint URL that is defined during the hosting of the application. The module parses this JSON object and generates a *Message* entity, which is then bound to a new *ClientHandler* instance.

In *sms_sender*, messages are encoded as JSON payloads and posted to the relevant API endpoint URL.

Similarly, the LBS request handler module sends out LBS requests to the relevant API endpoint URL, and parses the received responses (in JSON format) into *Location* entities.

The Google Maps Geocoding API handler, *geocoder*, also uses standard HTTP to process address queries. The resulting JSON responses are parsed and converted into *Location* entities to be returned to the respective *ClientHandler* instance.

MySQL database queries are executed via the *MySQLdb* Python library interface, which then interacts with the database hosted on the app hosting domain itself.

All internal communications take place over standard Python method (API) interfaces.

VII. EVALUATION

A. Unit Testing

Since the application is triggered by, and solely dependent on, incoming text messages, correct message interpretation is very important. Several internal test cases have been derived in this regard, covering all possible types of queries that the application may receive.

The PyUnit [14] unit testing framework that comes bundled with the Eclipse PyDev [15] extension was used for evaluating unit tests. Python's unit test package was used for defining test cases. The tests generally focused on evaluating the message parser against different valid and invalid messages, and asserting the outputs and errors generated, respectively.

B. System Testing

While unit testing is important, more emphasis must be placed on system testing when it comes to a system like ShopGuru with multiple modules working in collaboration.

System testing was performed using the IdeaMart Simulator provided as part of the IdeaMart Developer Bundle. This simulator provides a mobile phone-like interface via which text messages can be sent to the application from preferred mobile numbers. Responses sent by the application are displayed in the simulator itself.

Since the simulator runs on the local machine (localhost), it can be accessed via a web browser. This allowed the use of Selenium IDE [16], the popular website functionality testing framework, to be used for overall testing of ShopGuru via the simulator. Test cases were written to send a predefined sequence of messages to the application, and assert that the expected responses are received.

As the data currently existing in the database might interfere with the testing process, two (*Preconditions* and *Postconditions*) test cases were written such that *Preconditions* backs up the data currently in the database and 'cleans' it (by

truncating all tables), and *Postconditions* restores the old data back into the database, both using the phpMyAdmin database management interface.

For convenience and modularity, two custom functions *doSendSMS(phone*, *message)* and *doSetLBSLocation(latitude*, *longitude)* were written and added to the Selenium Core.

All test cases were combined into a test suite, which could be run whenever required to verify that the overall application functions as expected, conforming to the required quality of service parameters (e.g. timeout limits could be defined for an SMS, in order to measure the latencies in serving queries.)

C. Performance and Accuracy

Duration between a user query and the corresponding response was measured to be approximately 6 seconds. While the actual query processing did not usually take more than 2 seconds on the server side, communication overheads such as delays in message delivery and database queries accounted for the remaining time difference. Significant deviations were observed in case of customer and shop registrations, where location geocoding using Google Maps API was performed over the internet, which was clearly due to the network communication overhead. Due to the same reason, actual performance of LBS-included queries could not be measured.

The application produced accurate responses for all cases considered in the system tests (e.g. location- and name-based shop queries, ordered by subscription priority and by location) for different forms of client (customer and shop) queries.

VIII. Future Enhancements

A. Smartphone Interface

While ShopGuru is available as a regular SMS application for all Dialog subscribers, it is also possible to introduce an Android and/or iOS interface for the app. This would allow non-Dialog users to use the service, while allowing smartphone users to enjoy more advanced features (like Maps-based display and navigation to nearest shops, and richer shop status notifications including banners, icons and formatted text).

B. Item Reservation Capabilities

Via Dialog's eZcash (CAAS) API, it would be possible to extend ShopGuru to a point where a customer can not only query a shop status, but also place an order or reserve an item on the shop (to be purchased on arrival at the shop).

C. Natural Language Processing (NLP)

Rather than depending on a specified querying format, it is expected to make ShopGuru understand natural language queries (such as "register me as a shop"). IdeaMart Apps like Mega Mind already possess this capability.

D. Expansion to Other Mobile Service Providers

Dialog is only one of the several mobile service providers in the country. It is considered that Mobitel has acquired an even larger share of the mobile subscriber community. Since ShopGuru's source code is not tightly coupled with IdeaMart, it would be possible to introduce the same service under Mobitel and other service providers. It might probably involve the necessity to enter into contractual terms, unlike in case of IdeaMart which is totally free for independent app developers.

IX. Conclusion

ShopGuru attempts to address a shop information retrieval problem commonly encountered by both the customer and seller community by providing a query service for mobile devices via the simple, easy-to-use and widely available SMS service. It can be considered as a platform where shopkeepers can post and update their shop information, which can then be queried by interested customers on demand. While it has the advantage of being able to reach a wider audience, it has some inherent disadvantages such as the lack of a user interface and the need to memorize some simple yet relatively long query syntaxes. Details regarding actual usefulness, effectiveness and customer acceptance of ShopGuru would be known once it is placed in active production as a public IdeaMart app.

Acknowledgment

The author would like to acknowledge all lecturers and colleagues who advised, evaluated and assisted him, members of the Dialog IdeaMart community who provided him guidance and assistance, and the authors of all online and offline resources consulted during the course of the project.

References

[1] N. Jain, T. Hatt, and A. Wills, "Country overview: Sri Lanka," GSMA Intelligence, London, Oct. 2013.

[2] Dialog [Online]. Available: https://www.dialog.lk [Accessed: Sep. 16, 2014].

[3] Mobitel [Online]. Available: https://www.mobitel.lk [Accessed: Sep. 16, 2014].

[4] P.K. Jaiswal, "SMS based information systems," Master's thesis, School of Computing, Univ. of Eastern Finland, FI, Sep. 2011.

[5] P. Gichure, "SMS query & response system: the power of SMS" [Online]. Available: http://www.slideshare.net/gichurepaul/sms-query-response-system [Accessed: Jul. 5, 2014].

[6] R. Rajachandrasekar, Z. Ali, S. Hegde, V. Meshram, and N. Dandapantula, "Location-based query processing: sensing our surroundings," Dept. of Comp. Sci. & Eng., The Ohio State Univ., OH.

[7] "20 hot location-based apps and services you should know about" [Online]. Available: http://www.slideshare.net/socialtech/20-hot-locationbased-apps-and-services-you-should-know-about-12841489 [Accessed: Jul. 5, 2014].

[8] Dialog, "IdeaMart" [Online]. Available: http://www.ideamart.lk [Accessed: Jul. 5, 2014].

[9] Zhvillues, "Mega Mind" [Online]. Available: http://zhvillues.tumblr.com/post/103883950726/mega-mind [Accessed: Sep. 10, 2015].

[10] WikiLanka code base on GitHub [Online]. Available: https://github.com/ridwannaruto/WikiLanka [Accessed: Sep. 10, 2015].

[11] Dialog, "Idea Pro" [Online]. Available: http://www.ideamart.lk/idea-pro [Accessed: Jul. 5, 2014].

[12] "The Google Maps geolocation API | Google Developers" [Online]. Available: https://developers.google.com/maps/documentation/geolocation [Accessed: Jul. 3, 2014].

[13] ShopGuru code base on GitHub [Online]. Available: https://github.com/janakaud/shopguru [Accessed: Sep. 16, 2014].

[14] "PyUnit – the standard unit testing framework for Python" [Online]. Available: http://pyunit.sourceforge.net [Accessed: Sep. 16, 2014].

[15] "PyDev" [Online]. Available: http://www.pydev.org [Accessed: Jul. 5, 2014].

[16] "Selenium IDE plugins" [Online]. Available: http://www.seleniumhq.org/projects/ide [Accessed: Sep. 16, 2014].