Ujumbo: A Generic Development Toolkit for Messaging Based Workflows

Dept. of CIS - Senior Design 2012-2013

Archit Budhraja
arcb@seas.upenn.edu
Univ. of Pennsylvania
Philadelphia, PA

Bob Han
hanb@seas.upenn.edu
Univ. of Pennsylvania
Philadelphia, PA

Sung Won Hwang
sunghw@seas.upenn.edu
Univ. of Pennsylvania
Philadelphia, PA

Jonathan J. Leung
jonleung@seas.upenn.edu
Univ. of Pennsylvania
Philadelphia, PA

Sean Welleck
wellecks@seas.upenn.edu
Univ. of Pennsylvania
Philadelphia, PA

Boon Thau Loo
boonloo@cis.upenn.edu
Univ. of Pennsylvania
Philadelphia, PA

ABSTRACT

Many organizations, especially in developing countries often lack technical expertise, resource, and financial capital necessary to build applications with complex logic flows. Looking at the different kinds of products on the market, there seems to be a lot of commonality between them yet each of them is developed independently of each other, requiring programming ability and a substantial investment in time and financing for each application. Our project aims to reduce these additional costs in building a new application with common communication features by creating a generic platform that users with non-technical abilities could build off of. This will lower the temporal, financial, and technical knowledge required to developing the additional application.

In the paper we will further discuss the motivations for our platform, the high level structure of our program, how it is implemented, and the work we have completed and the work that still remains.

1. INTRODUCTION

Communication and general management platforms have rapidly grown in recent years due to the increased reliance on cell phones and the rapid adoption of web and electronic management of data. Many organizations are leveraging this technology by building complex services on top, as well as relying on these technologies to manage general day to day operations.

At the same time, many small scale organizations as well as non profits, often lack technical expertise, resource, and financial assets necessary to build such web and mobile applications. Looking at the different kinds of products on the market, there seems to be a lot of commonality between them. Yet, each of them is developed independently of each other, requiring programming ability and a substantial investment in time and financing for each application.

Our product aims to reduce these additional costs in building a new application with common communication features. We have created a generic platform that others with non-technical abilities could build off of, lowering the temporal, financial, and technical knowledge required to develop the additional application.

We began development by consulting local and international nonprofits and small scale organizations that may require custom platforms in their operations. We gathered data on their proposed workflows, needed functionality, as well as surveyed their available financial resources. After talking to several resource-constrained and non-profit clients, it was observed there was far too much manual communication. Most organizations we interviewed are physically entering and processing data on day to day operations, often in simple spreadsheet software. They also invest many hours manually calling and text messaging their members for recurring events. Due to their rapidly expanding user-base, many have expressed an exigent need to automate these processes. This motivated us to create an efficient and general toolkit of such tools.

Our finished product has the ability to read spreadsheet and CSV input data from Google Docs or outsourced CSV files; Bulk SMS/Email sending features using Twilio for SMS and SendGrid for Email, and scheduling ability using Resque scheduler. By integrating all these features into a comprehensive platform, we can let different organizations save time, effort, and money from repetitive manual work by allowing them to use our platform to create a customized application suited to solve their particular problem.

In order to implement such system with different inputs and communication outputs, we developed an architecture called “Pipeline”, a chain of modules that perform a custom series of actions on the data dictionary input. The idea of this architecture is motivated from Rack Middleware(Rack) [7], an interface between Ruby web servers. We will talk more about the details of what a pipeline is and how it works later in the System Implementation section.

2. RELATED WORK

We investigated several services and products that are similar to our project in terms of functionality, target audience, and/or software architecture.

2.1 Generic SMS Application Platforms

RapidSMS [8] is a general web framework for data collection, logistics coordination and communication, using basic
SMS technology. An application built using RapidSMS is able to receive input from basic mobile phones, and can display collected data online. For example, ChildCount is a product built with RapidSMS that allows health workers to register patients and send health reports to a web dashboard by sending text messages. RapidSMS is an extension of the Django web framework, and thus requires programming ability for setup, use, and customization. RapidSMS differs from the proposed project since it requires the end user to have technical expertise. However, RapidSMS addresses a similar problem space: creating a generic platform for building SMS based applications.

Frontline SMS [2] is a generic, mass SMS-based platform that allows users to send, receive and manage SMS over a mobile network. Frontline SMS eliminates the need for an internet connection, and allows easy development of forms that incorporate SMS. However, the platform does not work with many phones, and does not support any Android phones. Including Frontline SMS in an application requires programming ability; however, simple SMS based applications such as surveys or polls can be created without programming. Frontline SMS is open-source, so we will their product can be used for non-Android SMS communication within our platform.

Another product that is similar to our project is CommCare HQ [1]. CommCare HQ is a general platform for mobile data collection that allows creation of forms and questionnaires that use mobile data as input. The platform helps with application creation and deployment, as well as managing the collected data. CommCare differs from our project in that it does not provide a GUI for nontechnical users to develop applications.

2.2 Non-Technical Application Development

One existing product that opens SMS and voice-based application development to non-technical users is SendFlow [5]. SendFlow allows users to create interactive applications through a graphical interface. SendFlow abstracts various functions such as SMS messaging and data source operations into drag-and-drop blocks, which can be arranged into logical flows. SendFlow’s user interface will be a useful guide for designing the interface for our project.

Another product that allows non-technical users to create an application is Microsoft SharePoint Designer [9]. Microsoft SharePoint Designer, a member of the Microsoft Office suite, contains a Workflow Designer that allows users to create an application based on actions and events. Example actions include sending an email, performing a calculation, or moving a document; example events include clicking a button, changing a document, or deleting a spreadsheet row. Using the Workflow Designer, a user can create a flow of actions and events using a graphical interface, without programming knowledge.

The inspiration for our process flow comes from a product called If-This-Then-That (IFTTT) [4]. This web-based service utilizes a similar trigger/action workflow. Users can specify recipes consisting of a single condition or trigger which will cause an action to occur when activated. It provides a simple and intuitive interface for non-technical users at the cost of efficiency. We have incorporated a portion of this process flow into the development of our platform, and expanded upon this concept by developing the ability to chain multiple triggers and actions into full workflows. In addition, we have also developed the ability to batch process large amounts of data for each trigger or action block.

2.3 Related Software Architecture

In addition to applications with related functionality, we also did research on applications that we could use to help design the structure of our platform. The architecture of Rack [6] middleware is similar to our platform’s structure. Rack is an interface between Ruby web servers. Using Rack, one can build a modular, minimal web application that takes a defined environment as input, and outputs an HTTP response. Rack is designed such that multiple applications can be chained together, with each application using the response information outputted by the preceding application. The concept of “chaining” is used as a model for our ‘Pipeline’ architecture, which links modular actions together to create an action sequence. Another design inspiration derives from Rack’s Decorator pattern, which establishes a common input and output format for applications, thus allowing general chaining. Namely, all Rack applications input and output three element arrays. In our platform, we established a common input and output type by using Ruby hashes to pass parameters across pipes. Such a design allows for dynamic, flexible input and output to pipes while allowing pipes to be linked together.

2.4 API’s

Another part of our platform involves creating general modules for SMS messaging, emailing, document input and output, and scheduling. We have incorporated several APIs into our platform in order to help accomplish these functions.

2.4.1 Communication API’s

Twilio is a communication API that we use for sending and receiving SMS messages and voice calls. For email sending and filtering, we use the SendGrid API, which provides functionality for sending custom emails, filtering incoming emails, and defining responses to various email-related events.

2.4.2 Data I/O API’s

Our platform modularizes interaction with Google Docs. An existing Google Docs API wrapper, called Google Drive Ruby [3], is used to implement this module.

3. SYSTEM MODEL

Several difficulties that campus organizations face in their operations, according to our research, have commonalities: disorganized communications and manual tasks that could be automated. The specific challenges these entities face come down to three points: Data Input(I/O) and Modeling, Messaging or Communication, and automating a custom series of these items. The system model of the platform/toolkit Ujumbo is designed based upon these components.

3.1 Data Input and Modeling Modules

To readdress the issue, many of the student groups face inconvenience in bulky or frequent data input and conversion. For example, Preceptorials committee, a student organization that runs non-credit seminars at University of Pennsylvania, gets user registration data from the registrar
in the form of CSV files. Some groups get data in embedded email forms, or other groups need and get data from Facebook Events or just simple plain text emails. All of the groups in our research pool manually transfer all the data into Google Docs and manually modify information for changing schedules. These random changes are followed by manual modification of related attributes as well. The first part of our model addresses such series of tedious work.

In order to automate such functionality and reduce the manual information conversion, two components are needed: Data IO Component and Data Modeling Component. The Data IO component would be externally facing and gathering information from data sources such as Google forms, email, or CSV files, etc., and a Data Modeling component can keep track of the various attributes that change over time.

Since the Data IO component needs to take in data from several sources, the interface needs to be generic enough such that the interface for interacting with Facebook Events would look the same as a CSV file. This interface will be a simple CRUD functions (create, read, update, and destroy) which will interact with a single “row” of data for the particular source. Once the data is inputted into Ujumbo, it is synced to the Data Modeling layer. If something is modified in the Data Modeling layer, it synchronizes back to the original data source.

### 3.2 Messaging

The small organizations also commonly face the workflow that require them to communicate with their constituents in bulk both over EMAIL and SMS. For this workflow messaging layer is needed where they can write template emails or messages, fill in the correct parameters for each user (such as name), and then send them out. Such communication tools are essential in every action that occurs in the platform.

### 3.3 Action

Many of the described procedures of the target groups also required being able to chain a series of tasks. This requires the following aforementioned modules. As an example, the Preceptors committee would undergo following series of action repeatedly:

- Emailing a person for the availability (Messaging)
- Noting that person’s availability (Data IO => Data Modeling)
- Given the available people, randomly choosing people (Data Modeling)
- Informing selected students over email (Messaging)

The platform requires a series of various actions to occur to replace current target users’ series of manual task. As such, we put various pieces of functionality into modules and we can string them all together. Upon occurrence of some associated trigger, the platform is required to execute a series of associated actions to kick off sequentially. Such series of actions are implemented using pipelines.

As seen above in the example of tasks, such actions can be anything. It can be a form creation, emailing or SMS, or updating data source automatically. Thus, briefly put, the pipes can be anything. Figure 1 below illustrates a series of action in a particular case where each row in the data source represents a person that the user wants to send a message to, illustrated as a pipeline.

![Figure 1: A sample Pipeline](#)

### 3.4 Trigger

A trigger is anything that causes some action(s) to happen. Submitting forms, typing some command on the data source (spreadsheet), receiving an SMS, or anything can be the trigger to ignite a series of desired actions customized in the platform. This trigger in our platform, due to the target users’ demand, is relatively centered around communication; the messaging layer kicks in, using the data modeling component.

### 4. SYSTEM IMPLEMENTATION

Now we examine the implementation the system model. First of all, the data sources are primarily Google Docs spreadsheets, which is the chosen one of many possible data sources, implemented with Create, Read, Update, Destroy (CRUD) functionality. Generally, we implemented the three components of our system using Ruby on Rails with MongoDB and Redis for database to interact with data source as well as pipelines and triggers. Let us dive into the implementations and conceptual details about each of said components of our system model: Data Source, Triggers and Pipelines.

#### 4.1 Data source with CRUD functionality

The data source component of Ujumbo platform is designed with the CRUD functionality, and all the data used for the platform is written, stored, and communicated through Google Docs and any convertible format of spreadsheet. In theory, Google Docs is just one of data sources. We may use CSV files but we chose Google Docs because it has been the most popular among the organizations we talked to.

#### 4.2 Trigger

A trigger is the event that kicks off the system and starts the pipeline, the series of action. The pipeline can “set” an event that would trigger the initial flow of data from data source through the pipeline. Trigger can be any kind of event, entered into our platform anywhere from an external API call to an internal Redis Pub Sub. For example, the creation of a new row in Google doc could trigger a set of pipeline by calling our API endpoint. In our platform, we have a fixed set of triggers:

- **On API call**
- **On Row Creation/Update/Destruct**
- **On Receipt of SMS**
- **On Receipt of Email**

For example, in NoteCast, if you have an email address under email column and type “Send”, this will be a trigger...
to the pipeline, the server will pick it up and convert it to “SENT” on the spreadsheet, and the next action will be sending the email to the passed on email address gotten from the trigger.

4.3 Pipeline

The series of pipes captures a series of actions that users desire the platform to take. Such pipes form a pipeline. Once the trigger kicks off a pipeline, the trigger passes a bundle of dictionary parameters to the pipeline. These parameters can be any information necessary for the series of subsequent cascading actions such as sending SMS or email, or auto-populating a spreadsheet row. Each pipe gets passed a dictionary of information about its preceding pipes, static properties formed by the trigger initially, and the pipelined properties that can come from any preceding stages in the pipeline; output of each pipe’s action get added to the dictionary that is passed onto the next pipe. While pipe can contain any action as we mentioned in the System Model section, the below is the list of pipes we actually implemented on Ujumbo platform:

<table>
<thead>
<tr>
<th>Pipe</th>
<th>Available Pipe Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google Docs CRUD Pipe</td>
<td>Create Row, Read Row, Update Row, Destroy Row</td>
</tr>
<tr>
<td>SMS Send Pipe</td>
<td>Used to Send SMS to a given phone number</td>
</tr>
<tr>
<td>Email Send Pipe</td>
<td>Send Email to a given email address</td>
</tr>
<tr>
<td>Template Pipe</td>
<td>Create custom texts based on various attributes from the pipelined dictionary with fields similar to that of MailMerge</td>
</tr>
</tbody>
</table>

Note that Ujumbo has a fixed set of Pipes in it, although it is very easy to build more. Users don’t create new kinds of pipes but users can instantiate new pipes and customize their properties for their own needs. When a given action in a pipe is taken, static properties are the ones you know when the pipe is initially created, and pipelined properties change from trigger to trigger.

For example, in NoteCast, the showcased application built on top of Ujumbo platform, the user who sends emails through the platform is always the same - static property - that is declared at pipe creation. However, the persons the email gets sent to changes based on the data on data source, the passed-on properties of the trigger.

4.4 Implementation of data source as a Flexible Component

Since Google Docs spreadsheets are a primary data source for our application, we needed a powerful library for interacting with such spreadsheets. We needed one that:

1. Could create ‘real’/online Google spreadsheets
2. Interacted with the online Google spreadsheet: add rows, delete rows, modify rows
3. Notified us when something was changed, added, or deleted from the ‘real’ Google doc
4. Tracks, and responds to, changes in the Google doc
5. Capture all the above functionality for a given doc into a single object that could be used by our backend

To encapsulate all of the above functionality, we created a GoogleDoc class. When a GoogleDoc object is created in the Ruby program, a real online Google spreadsheet is also created on Google Docs. The GoogleDoc object allows the programmer to interact with the ‘real’ spreadsheet: it supports creating, reading, updating, and deleting rows within the doc, finding rows, and adding or removing columns. The GoogleDocs API allows us to make callbacks to our server whenever the contents of a document is changed with the given changes as a dictionary. Another powerful feature is that each spreadsheet has an embedded script that sends an http request to our API when the spreadsheet has been modified. The GoogleDoc object then tracks these changes, and can set off a trigger if necessary. To implement the all of the above, we used the Google Drive ruby gem to interact with the Google Doc spreadsheet. We programmatically attach a custom Google Script to each doc that detects changes. The Google Doc object itself is stored in MongoDB.

5. USE CASE

One example workflow that is easily built using Ujumbo is called NoteCast. NoteCast works with a Google Spreadsheet interface and allows you to create customized spreadsheets with a form, just like one would do in a regular Google Form Spreadsheet. NoteCast provides 3 tabs in the spreadsheet by default where data is populated - Users tab, SMS tab, and Email tab. The Users tab just has the columns from the form and rows are added to this tab whenever a new user fills out the form. The SMS and Email tabs allow us to utilize Ujumbo to send and receive email and text messages from within the Google Spreadsheet - no email client or phone is required. All you need to do is type in the body of the message in the same row as the user’s email address or phone number, type ‘Send’ in another cell, and press ‘Enter’. Members’ (recipients’) responses to these messages are also automatically populated in the spreadsheet, and hence you never need to leave the Google Spreadsheet for anything.

![Figure 2: NoteCast: auto fill-in of response to the sent text](image)
Using NoteCast as an example, we can now see what exactly data sources, triggers and actions are. A data source is essentially a Google Doc - a document or a spreadsheet in which data is populated which can then be used for various tasks/purposes. A trigger is anything that causes something to happen. An action is the 'something' that happens when a trigger is triggered. So, in NoteCast, our data source was a Google Spreadsheet. A few examples of trigger/action pairs in NoteCast are:

- **Trigger**: Typing information into a form, and clicking 'Submit'.
  **Action**: A row in the Google Spreadsheet is updated with the information that the user filled out in the form.

- **Trigger**: Typing ‘Send’ in a row containing a user’s phone number and a message body (in the SMS tab), and pressing ‘Enter’.
  **Action**: An SMS is sent to the user with the message in the spreadsheet.

- **Trigger**: The user types a response SMS and sends it to Ujumbo.
  **Action**: A row in the Google Spreadsheet is updated with the user’s response.

So, as we did with NoteCast, we can use any combination of triggers/actions along with Google Docs to create an application with some required functionality using the Ujumbo platform.

6. RESULTS

The primary result of this project was the Ruby API component of the Ujumbo Platform, which developers can utilize to build customized applications like NoteCast. The API allows developers to harness full potential of the Ujumbo Platform and put together customized flows using various triggers and actions along with Google Docs as a data source to build applications that provide the required functionality. NoteCast was built using just 38 lines of code, which is a testament to utility of the Ujumbo platform.

The Ujumbo platform has shown that it is possible to abstract many of common and even complex workflows into a generic platform and that this indeed can reduce total number of lines of code needed to write and ultimately the development time. Once the platform was complete, it only took a few hours to construct the NoteCast example.

7. FUTURE WORK

There still remains work to be done on the Ujumbo Platform, and the biggest goal for us moving forward is to implement a GUI for non-technical users that makes it extremely easy and efficient for them to create applications like NoteCast without having any knowledge of the underlying API or any technical expertise. We envision a drag-and-drop interface as in Figures 4, 5, and 6 that would allow individual users as well as organizations to put together custom flows to achieve some functionality.
8. ETHICS

As a functioning application creating platform, our product does not have inherent ethical issues. However, steps must be taken to ensure that issues that may arise from our users’ created applications are kept to a minimum while maintaining functionality as well as the privacy of the users. First of all, the integrity of the individual applications must be maintained, that data or settings from one application can not be accidentally or maliciously accessed by a user without the proper permissions. This is done through custom permission levels explained above, as well as our encryption measures.

Our platform also supports massive and iterative emailing and text messaging functionality, which lends itself to inappropriate spamming or advertisements. Care must be directed at the consumers of our application in the form of notices both in the agreement and when an excessively large amount of messaging is detected. In addition, opt out procedures for the recipients of messages sent by our platform should be implemented in a simple and effective manner.

Lastly, because our application may handle sensitive contact information, care must be made to ensure that we ourselves do not accidentally leak or access such information. Because we may need to access users data during debugging procedures, or service requests, we will do our utmost to let the users be aware of any access to database information, as well as garner sufficient permissions as appropriate. We must also make the users aware of the limitations of our security procedures. Although we have implemented basic encryption and security, there is still a chance of sensitive information leak, and the user should be aware of such a possibility before any use of our platform.

9. CONCLUSION

Throughout the year, our team shaped up the way to model the existing problems for the target users of our platform. Parsing each of the issues and their correlation resulted in the system model with three components: Data source, triggers, and actions. The implementation of the platform’s system design centered around Google docs as the primary data source, and desired actions were implemented extensively using communication tools such as SMS and EMAIL. The most significant conceptual breakthrough for our team was the use of cascading pipeline with a dictionary passed through, as an implementation of such series of actions. These series of thoughts and implementations resulted in the platform that lets you easily build a group data organization app like NoteCast, which shows off all the pipes available in Ujumbo. There is intriguing future work to be continued further over the year.

10. REFERENCES