Bridge
For the messaging gap between your devices

Sam Riester – Kyle Morley – Shawn McKay

‘Bridge’ is what will become the connection for messaging between your computer and phone, regardless of the devices you use. The goal with this project is to have a messaging app that relays SMS messages to and from your computer and phone using Bluetooth instead of through a server like other similar apps. This will help us provide a responsive and private experience to the user.

I. INTRODUCTION

Bridge is our cross-platform app that enables the user to send and receive SMS (and potentially MMS) messages on their computers via a Bluetooth connection between a phone and the computer (with a focus on Linux). The app on the phone side would be very minimal, having potentially one activity to setup the app as the user likes, and having it only run when it receives relevant broadcasts. On the computer side, the application would mimic a messaging app on a phone where the user can see all current conversations on the side, and the main conversation focused on the other side with the option to respond to messages using a text box at the bottom of the window. This application will be very useful because currently there is no app/application for messaging across a phone and a Linux based computer. Also, other messaging apps that are available sync user data, including messages, on their servers which can become a privacy issue for users. ‘Bridge’ will be a light weight, intuitive and private messaging app for everyone.

II. RELATED WORK

The main application that we are trying to bring to the android users is iMessage and most of our inspiration came from it. iMessage is only available to users of iPhones and mac computers so it is not at all a competitor to bridge. The direct competitors to bridge are the other applications tailored to android that have attempted to bring iMessage functionality to android users. There are 3 of these applications. These applications are Airdroid, Samsung SideSync and KDE connect.

Airdroid: full featured messaging app but is browser based. It lacks native feel and integration. It also requires that you are connected to the same unrestricted WIFI network on your phone and laptop.

Samsung Side Sync: Only available for users with android phones and windows laptops. This is also not a full featured messaging app, it is actually just remote desktop for your phone.

KDE connect: The only option which uses blue tooth. This means KDE connect will work without an internet connection. However, the application has a poor UI with no conversations or contacts.

Bridge attempts to take the best features of the 3 listed applications above and develop an application similar to iMessage.

III. APPLICATION DESIGN (MOBILE UI)

As you will see in the screen-shots at the end of our report, the UI of Bridge on the mobile side is very simple. This is because in day to day use the user will never actually have to open the mobile app. The first time the user uses Bridge they must accept the permissions within the app but that is the end of any configurations. The mobile UI follows material design rules, even the Icon that we have chosen is material. The icon comes from Icons8.com where all icons are free for commercial use as long as there is a reference in the description on the app/play store. Under the Icon the user simple can view the status of the connection and whether any contacts/messages are being synced at that time.

IV. APPLICATION DESIGN (MOBILE LOGIC)

The back end of the mobile application is very complex. There are a lot of moving pieces including blue-tooth, content providers, and broadcast receivers. I used the android classes BluetoothAdapter, BluetoothDevice, and BluetoothSocket. I first create a BluetoothAdapter, then use that to get a reference to my BluetoothDevice. After I have this reference, connections(sockets) can be created and managed using the createRfcommSocketToServiceRecord(UUID) function of the BluetoothDevice class. The Input streams and output streams for the data link can now be opened through that socket.

There are several content providers being used as well. On initialization, the content providers are used to retrieve all of the user’s messages as well as contacts. These are then synced with the desktop using JSON. A class called Gson was used for JSON object conversions.
There are also two broadcast receivers listening at all times for new text messages. This is because Google uses a different broadcast receiver for SMS messages being sent and received. These broadcast receivers are important because if new messages are received or sent from the phone, these new messages must be reflected on the desktop side. The mobile side has two objects defined, Message and Contact. These match the same definitions on the desktop side and are used to send back and forth as JSON over Bluetooth.

V. Application Design (Desktop UI)

The desktop UI for this project needed to be fully featured while still being user friendly. We decided to go with a typical messaging app layout. We wanted to have all current conversations listed on the left hand side, showing the people you have been talking to with a short preview of what the last message sent was. The user can easily switch between conversations by clicking on the desired conversation. On the right portion of the screen, the user can see his/her current conversation. This portion has all of the previous messages in a window so the user can look back at what has been talked about, as well as a text box at the bottom to allow the user to send a message from the computer to the current conversation. Last, we have a button below the current conversations for adding a new conversation to the list. When clicked, the ‘current conversations’ box clears and repopulates with a list of all of your contacts. The user can then select a contact to begin a conversation with, then they are added to the current conversation panel.

VI. Application Design (Desktop Logic)

The backend of the desktop side of Bridge consists of two main parts: the Model and the SPPServer classes. The SPPServer class manages the serial port profile for sending and receiving data from the phone. When the desktop java application launches, the SPPServer class is launched in a background thread that waits for a mobile device to connect. Once a mobile device has been connected the program launches another thread that listens for JSON strings sent from the phone. The server converts the JSON into java objects of Messages or Contacts and sends this data to the Model. The server also handles the responsibility of writing data to the phone. The server sends data back to the phone over the same SPP interface as a JSON string.

The model class handles all the business logic for the desktop end. The model contains the data received by the SPP server. Contacts are sorted into a contact list, and Messages are sorted into conversations by the Model.

VII. Results

The goals for this project were:
1. Compatible on Window, Linux, and Mac: Bridge is written entirely in java and interacts with the system’s bluetooth through the java library Bluecove. Bluecove is compatible on Windows, Linux, and Mac allowing Bridge to run on all 3 platforms.
2. Work without requiring Internet Access: Bridge only requires you to pair a desktop/laptop with your Android phone over Bluetooth and has no need for an internet connection to ever be present.
3. Full-Featured SMS capability: Bridge uses contacts and messages sent from your phone, giving you all the features expected from an SMS application.

We were able to meet all of the goals we set out to Complete.

VIII. Future Work

Now that we have a functioning application, we can start to focus more on the extra features of the application. Originally, we set out to create an SMS application, but we would like to include MMS messages as well, so the user can send and receive messages from their computer.

Another feature we would like to add is the ability to pair devices (the desktop and the phone) through the app. Currently, the user needs to pair the devices externally. In order to do this completely, we would need to implement some security measures during the pairing process to keep other people from being able to intercept the connection. For example, we could pop up a code for the user to enter when pairing the first time.

As with any other application, we would like to optimize ours for battery efficiency. There is always more work to be done on an app, ensuring that the radios are only used when necessary, and that data is transferred efficiently.

Lastly, we would like to give the user the ability to ask for a pin before opening up the messaging app on the desktop. Because we are dealing with potentially sensitive information, we want to be able to ensure that the messages and other content does not get into the wrong hands.

IX. Discussion

As a final project we are happy with the results we achieved. We are looking forward to continuing our work on Bridge. Right now Bridge functions well and we are using it for personal usage and hope to scale up for a full release.
REFERENCES


Figure 1: Mobile Application UI

BRIDGE

Bluetooth is enabled...
Connection established...
Updating Contacts...
Updating Messages...

Figure 2: Desktop Application in a conversation

Conversations

Stephanie Galen
It's magical!

Unknown number
Bank of America: Debit...

Dad
It is Reed's Birthday. It ...

Shawn Mckay
Gonna come print your...

Sam Riester
I like it.

Unknown number
Test Message from my ...

Conversation with Sam Riester

Is it on Craigslist

Yeah tell me what you think

Orbea asphalt?

Yeah