GSoC 2020 Proposal

Improve MMS support to the SMS client

Introduction

KDE Connect has recently launched an SMS Messaging app for the desktop which uses KDE Connect to synchronize all the existing conversations from the remote device. In its current state it can:- -

- Show a list of existing conversations and the corresponding conversation history.
- Send and display SMS.
- Send SMS to only existing contacts and conversations on the remote device.
- Display (not send) group messages.

However, it is very limited to MMS and can only support showing text. It cannot send and display pictures, audio, video and other kinds of attachments, also it cannot reply to multi-target conversations. Considering the importance of sharing multimedia files and group messages, the SMS client becomes very limited in use. This makes it very less useful than its commercial competitors. This project aims to improve MMS support with the integration of sending multi-target MMS by re-working the current implementation of sending SMS and UI support using Qt Multimedia and Kirigami-addons library (by Simon Redman).

By the end of this project, the SMS app will be capable of sending and receiving MMS, will be able to send multi-target MMS with the capability to view pictures, video/audio in the SMS app and the capability to download other attachments (.pdf, .txt etc.)

Goals

- Adding support to send simple plain text MMS.
- Support for multi-target MMS
- Support to view received pictures in the desktop app.
- Support to send Pictures from the desktop app.
- Support to receive videos/Audio in the desktop app.
- Support to send video/Audio from the desktop app.
- Support to download other attachments such as .pdf, .txt, etc.

Existing System

The current implementation of the SMS app has the flow of only plain/text messages from Android to desktop and desktop to Android by using KDE Connect's SMS plugin and contacts plugin, the former for the synchronization of conversations and the latter for the synchronization of contacts. The flow of messages from each end has different implementations, these two are summarized as

Phone to desktop message flow:- Desktop side requests for conversations list to the android side by sending the network packet of type PACKET_TYPE_SMS_REQUEST_CONVERSATIONS and in response the packet is received on the android end by KDEConnect's SmsPlugin which calls the handleRequestConversations(NetworkPacket) which get all the active conversations on the device with the help of SmsHelper class and constructs a bulk message packet of type PACKET_TYPE_SMS_MESSAGE and send it to desktop. The structure of the sent packet type contains an array of JSON objects corresponding to the "messages" field which in turn have various fields such as event field, body field having the text message, addresses filed, date field, thread_id etc. The point to note here is that it only contains a text message body and no multimedia or other attachments.

At the desktop end, the above-sent packet is received by SmsPlugin which calls its handleBatchMessages(NetworkPacket) method to create a list of ConversationMessage objects from the NetworkPacket and forwards the list to the DBus. The ConversationDbusInterface object on the other end of the DBus receives the request and caches the info of newly received messages using the addMessages(QList<ConversationMessage>) method and notifies the sms app through the DBus emitting corresponding signals such as ConversationCreated(QDBusVariant(QVariant::fromValue(message))), ConversationUpdated or ConversationLoaded(qint64 conversation, quint64 messageCount). Sms app in response updates the model/view data using ConversationListModel and conversationModel objects and displays the list of conversations in coversationList and text messages of a particular conversation thread in ConversationDisplay UI.

A similar process is followed for syncing all the text messages in a conversation by using the packet PACKET_TYPE_SMS_REQUEST_CONVERSATION for making requests to the android and SmsHelper's getMessagesInThread(this.context, threadID) method is called to fetch all the text messages belonging to requested threadID in the SmsPlugin.

Desktop to Phone message flow: - After clicking the event on the sendButton in the conversationDisplay UI the text from the TextEdit field is passed to the ConversationModel's sendReplyToConversation(QString& address) method or sendMessageWithoutConversation

(const QString& message, const QString& address) method if the conversation thread doesn't exist before. Either of the called method forwards the message to the ConversationDbusInterface object which in turn forwards it to the Smsplugin's sendSms(const QString& phoneNumber, const QString& messageBody) method on the DBus through conversationDBusInterface's replyToConversation(const qint64& conversationID, const QString& message) method or sendWithoutConversation(const QSring& address, const QString& message) method which ultimately prepares a network packet of the type PACKET_TYPE_SMS_REQUEST and sends it.

The current structure of packet PACKET_TYPE_SMS_REQUEST contains three fields required to send sms as represented below:-

*** { "sendSms": true,**

*** "phoneNumber": "1429048763213",**

*** "messageBody": "Hello from desktop" }**

For the SMS app to implement MMS support this packet type needs to be augmented or replaced. This packet when received at the Android end, SmsPlugin calls the SmsManager's sendMultipartTextMessage(...) method to send the sms to the specified phone number. However, this current implementation of sending messages from a desktop needs to be reworked to support sending MMS and multi-target messages.

The SMS app although in its current state can sense the difference between plain text and other types of files but is unable to send and display the MMS. This project plans to modify the SMS app's front end to support multimedia functionality and to change the current implementation of sending SMS on the Android side to support MMS.

Related Works

There are some works already done by Simon Redman in this direction. In one of his MR he has implemented the new structure of the PACKET_TYPE_SMS_REQUEST packet. Till now the new type labelled as version 2 consists of a "version" field, the "phoneNumber" field has been replaced by an "addresses" field which will now contain a list of addresses thus supporting multi-target messages, an "event" field and some optional fields such as "forceSMS" and "sub_id" are also added. This new version of the packet will have some more fields, like a "payload" field for carrying multimedia files and a "text" field for plain text SMSes and MMSes.

One of his other works in this direction is moving SMS app to kirigami-addons library . The kirigami-addons library will free us from the need to keep on hacking our local copy of the ChatMessage class which was originally copied from Kaidan and also this library will provide a

common point of development for all apps which want to provide a chat interface. This work will be very useful in developing the SMS app UI to support multimedia and other files.

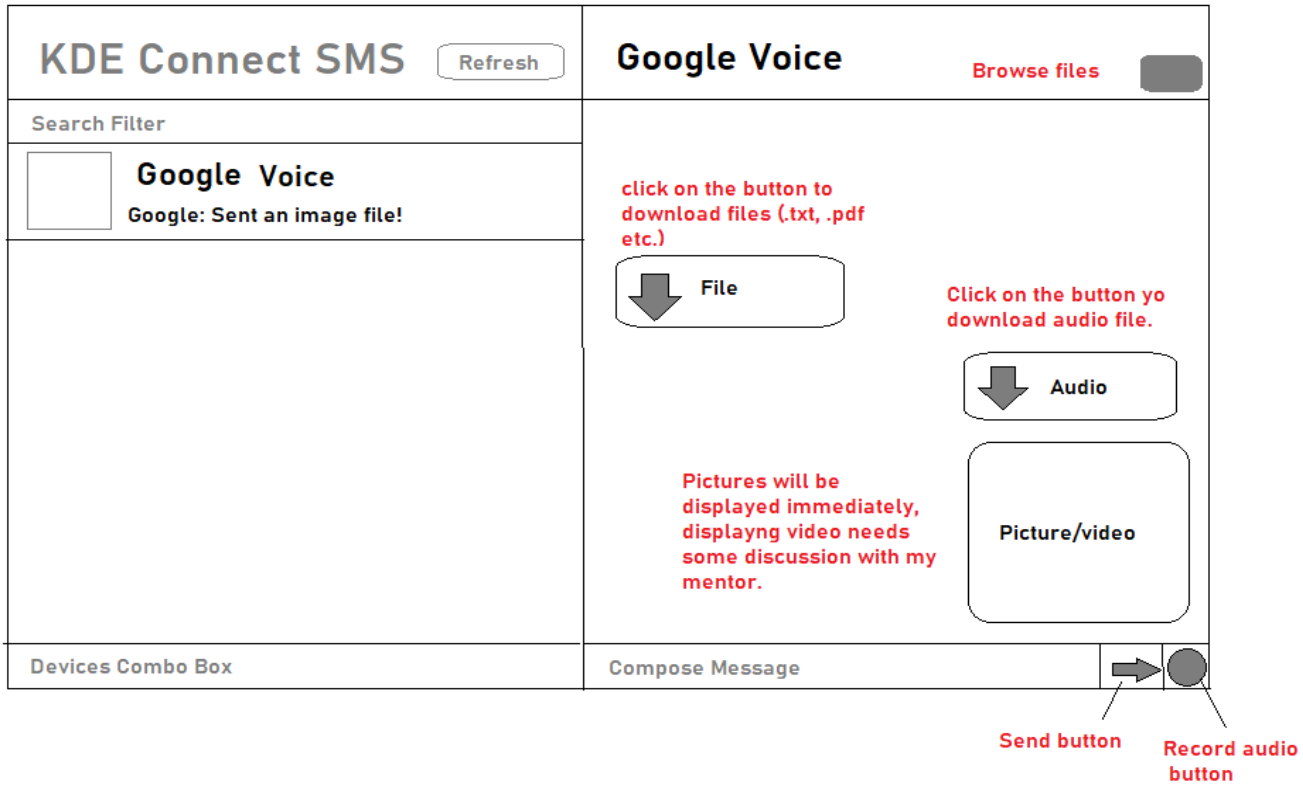
Proposed System

Various new features are queued to be implemented. In this section, I have explained how I will proceed to achieve desired results.

1. **Sending Simple plain text MMS: -** The current implementation of the SMS app sends plain text as multipart SMS, keeping the user's desire in view, whether they want to send the plain text as multipart SMS or as MMS the SMS app will provide a switch to the user so they can exercise their desire. When a user chooses to send it as SMS, SmsManager's sendMultiPartTextMessage method will be called or MMSPart will be used for sending plain text MMS.

2. **Synchronizing MMS attachments from Android device to SMS app: -**
   Synchronization of Multimedia files and other attachments of MMSes from Android to desktop will be mechanised as follows: -
   - Url of the particular MMS attachment will be extracted by calling the method **cursor.getString(cursor.getColumnIndex(Telephony.Mms.Part._DATA));** inside the SmsHelper's getMessages method.
   - This URL will be sent with the other fields of the PACKET_TYPE_SMS_MESSAGE packet to the desktop app.
   - If the attachment type is a picture, the sms plugin will send a low-resolution version of the image directly with the message packet. then will download the original image only when the user requests to download which would likely be sent using **FileTransferJob** class to a specified path on the desktop and the SMS app will read that file from that path and will immediately open it in the default app with the help of QDesktopServices class.
   - For video files, the thumbnail of the video will be extracted and it will be sent directly with the message packet to be displayed in UI as a still image. When the user will click on the thumbnail of the video, a download request will be sent to the Android device and it will be downloaded as described above for the original image files.
   - ConversationDisplay will display a button in place of the MMS file to download the audio/attachment. When the click event occurs a request will be sent with the URL of the attachment to the Android device to fetch the file to a fixed specified location from where the smsPlugin will read and open that file by clicking on that button in ConversationDisplay UI.

3. **Sending multimedia files and other attachments from SMS client to Android:-** This would be accomplished by augmenting the current PACKET_TYPE_SMS_REQUEST network packet to contain the payload field which in general will carry all sorts of files which can be sent as MMS. To send these types of files from the SMS app will need lots of modifications in the UI to attach files by using the QFileDialog class and selecting the desired file to send. An image holder to display images in the Conversation display widget and a voice recorder will be integrated to record and send voice messages immediately. All these multimedia support will be integrated using the Kirigami-addons library which is developed by Simon Redman and Qt5QML's multimedia library. The design of the UI would look something like this,



At the Android end, the received payload will be stored on the device temporarily or permanently before sending the file as MMS using the SmsHelper's **sendMultimediaMessage(Context context, Uri contentUri, String locationUrl, Bundle configOverrides, PendingIntent sentIntent)** method.

The need to store the file temporarily or permanently on the device will be part of the discussion with the mentor during the community bonding period. In my view, we need the file for later synchronization of MMSes when we restart the SMS app, but I will try to think of some clever way to avoid storing it on the remote device unnecessarily.

4. **Support to send multi-target MMS: -** This task will require some help from my mentor as I have little knowledge on how to get going with this feature. This task consists of sending group messages from the SMS app which on the Android side needs a bit deeper knowledge of Android's SDK. I will figure it out before the coding period begins.

Tools

I currently use KDE Neon 18.4 developer edition as my primary development platform. I use Qt-creator for my development purposes. I have KPeopleVcard and all other dependencies installed, which are required to build and run KDEconnect. For the Android codebase, I use Android Studio. For unit testing, I prefer using Qt Test. Using all these tools and my platform I have successfully submitted many patches to the KDEConncet source code which are merged. I will be using the same tools and configurations in the GSoC project as well.

Tentative Timeline

- **Pre-GSoC Period:** I have been contributing to KDE Connect for a month and have bonded with the community quite well. I have also added some features to the KDE Connect app and SMS app. During this period, my major focus will be on accomplishing my pending feature upgradation and solving some other bugs in the SMS app. My basic approach will be to get more familiar with the source code and will plan my implementation of the idea. During this period I will also work on sending simple Plain text MMS from the desktop app as it will serve as a pre-requisite for other MMS-related tasks.

- **May 4 - May 31:** During the community bonding the main focus will be to frame a roadmap for the project with the guidance of the mentor (along with improving bonding, which is what the period is for). The integration of the various parts of the project into one another will be framed, and a source code directory layout will be prepared to keep the flow smooth in the later stages of the development. Once into the coding period, I will be following the order as described below.

- **June 1 – June 15:** I will start by modifying the Message class to contain some reference to the file on the phone like the URL and MIME type of the file. Later on, I will use this reference in the desktop-side app to decide to request the file and will start the implementation of sending multimedia files and other attachments on the request from the SMS app. As for unit testing, KDE Connect is a bit hard so I will focus a little on it and will write unit tests only for possible modules during the implementation of that module. This approach will be applied to all the possible modules on which unit testing

would be possible throughout the coding period of three months. I will also dedicate some time to writing a blog, once every two weeks.

- **June 16 – June 30:** During this period I will extend the desktop side SMS app to receive MMS attachments from the Android device and save it at a specific path for later use by the SMS app using FileTranferJob class. For User testing, I have some friends at college who use KDEConnect and I will ask them for feedback at least once a month. This period will also be used as a buffer to complete documentation and fix bugs in the program and write a blog.

Midterm Evaluations

- **July 1 – July 14:** Work on replacing old-style PACKET_TYPE_SMS_REQUEST with new packet type to support sending MMS will begin by adding a payload field to the packet. Then sending attachments and multimedia files as MMS will be implemented on the Android side. Writing unit tests. Writing blog.

- **July 15 – July 30:** During this period the main focus will be on the implementation of sending Multitarget MMS. This period will also be used to write a detailed report and blog for work done during the second month and user testing.

Midterm Evaluations

- **July 31 – August 15:** This period will mostly focus on UI work. adding file browser and image viewer in the conversationDisplay UI. Adding buttons to download video files and play them. All UI work would be done with the help of kirigami-addons for chat messages developed by Simon Redman. Writing blog and user testing.

- **August 16 – August 23:** This week I will add audio support in the SMS app and will also Implement support in the SMS app's UI to handle other attachments (.pdf, .txt etc.).

- **Aug 24 – Aug 31:** Testing all the changes and finalizing the GSoC work. In this period I'll document all the changes and additions done to the SMS app during the GSoC period.

Personal Information

My name is Aniket Kumar and I'm currently enrolled as an undergraduate student at the DAV Institute of Engineering and Technology, Jalandhar. I'm in my third year pursuing a degree in Computer Science and Engineering.

Contact Information

E-Mail Id: anikketkumar786@gmail.com

Telegram Nickname: aniket_kumar7

GitLab profile: https://invent.kde.org/aniketkumar

Phone No. : +917508436347

Programming Experience

I have a programming background as my school courses introduced me to the basics of programming using Java. I developed a keen interest in application Development. I have been programming since my school days and continued it since. My main focus at the start was on Competitive programming and I used it to perfect my coding skills and DS/algorithm implementation. I have mostly used C++ in my competitive programming tasks.(hackerrank)

I have developed many small projects as university assignments or as personal projects (GitHub link) in C++ and some using the Qt framework as well. I have also developed one small gallery app for Android. I have basic knowledge of Android development. I have created a PR for KDEConnect- android's share plugin feature as well. I started contributing to KDEConnect nearly a month ago and since then I have made some contributions to the codebase of KDEConnect and the SMS app.

Contributions to KDEConnect

Link	Description	Status
GoTo	Start a new conversation in messaging app. (feature)	Merged
GoTo	Open sent files on remote device Desktop->Desktop (feature)	Merged
GoTo	Start a new conversation with any arbitrary phone number in SMS app. (feature)	Under review
GoTo	Moved ConversationsSortFilterProxyModel class from conversationlistmodel files to its own files	Merged

GoTo	Open sent files on remote device Android->Desktop. (feature)	Under review

Availability, during the GSoC project,

My semester will end by the second week of May. Hence, I will be able to dedicate all my time to my GSoC project at KDE. Internet Connection won't be a problem for me and therefore, I can remain in touch with the KDE Connect team via IRC channels. I will have no other commitments during the GSoC period. I will be able to devote at least 40-50 hours per week on my task and can extend my involvement if the need persists. I'm confident that I will complete my project in a timely manner.

Why me?

Most of my recent contributions to KDEConnect have been in the SMS app development and two related to the share plugin. I have added some features to the SMS app one is to fetch all the contacts available on the desktop and make it available to the user to start a new conversation. The second feature I have worked on is adding arbitrary contacts in the SMS app. While implementing these features I got familiar with its codebase, it's the flow of messages through various interfaces and how the app is structured. Hence I'm applying for GSoC in the SMS app-related project **"Improve MMS support to the SMS client".** I have successfully implemented these features and have shown an understanding of the SMS app's codebase and the ability to read and understand others' code and modify it accordingly.