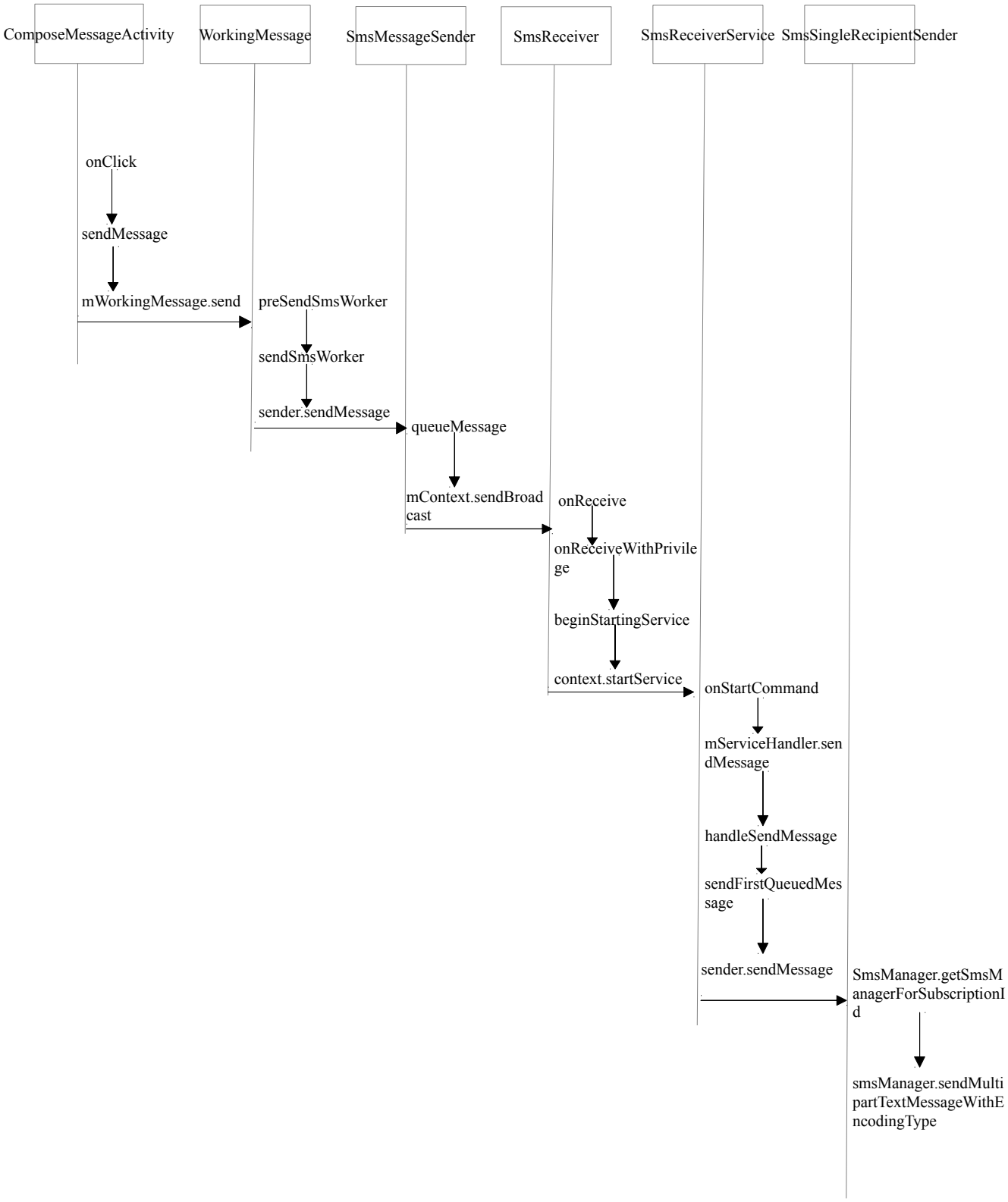
短彩信收发流程

黄辉

1.短信发送流程：

APP 层流程图：



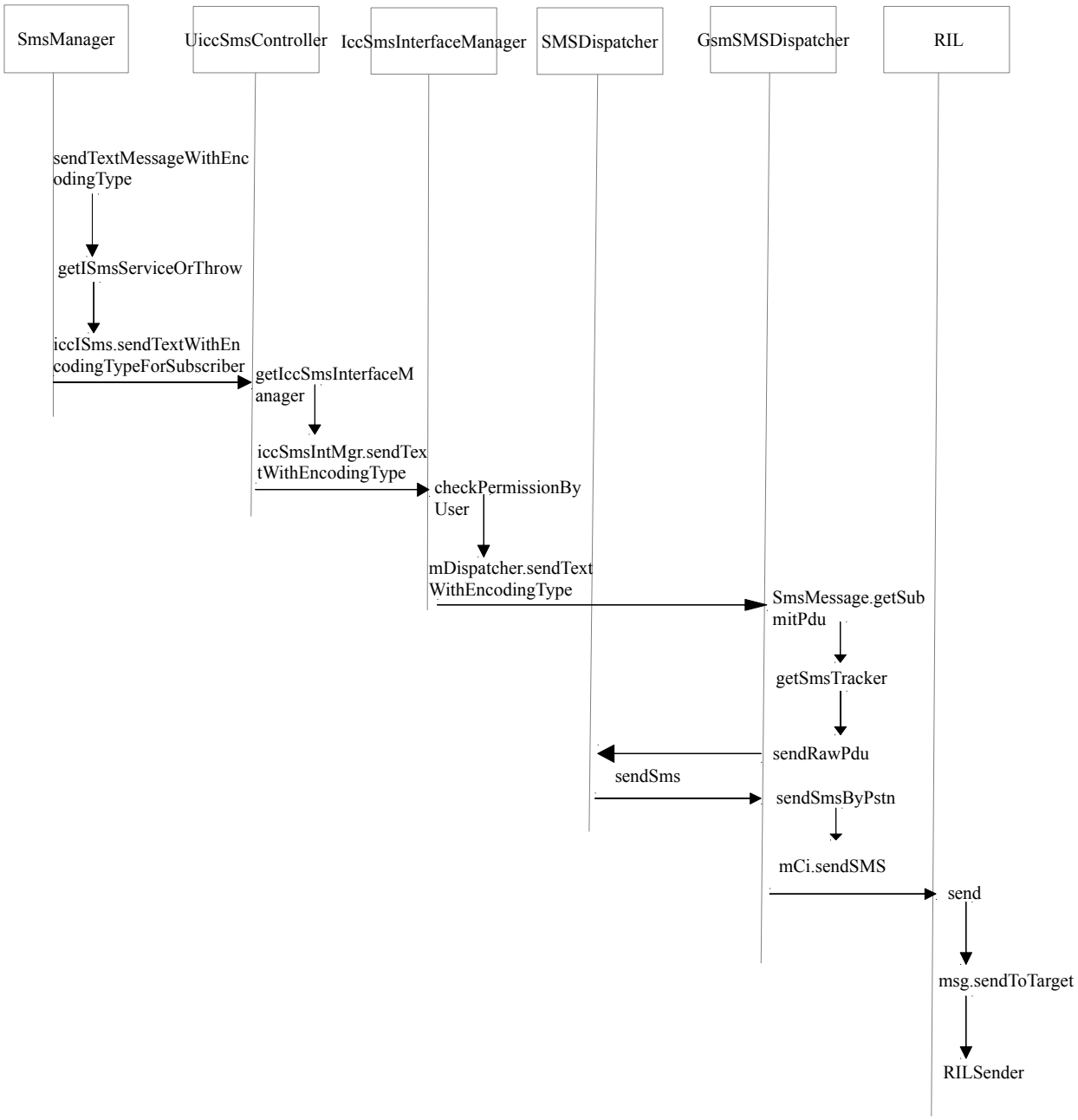
1. ComposeMessageActivity.java 短彩信的会话界面。是短彩信接收和发送的主要界面。当输入联系人和短信内容后，通过 WorkingMessage 的一些列判断，通常会判断是否有联系人，是否有内容，是否有手机卡。将短彩信的发送按钮置为可按。点击发送后。会调用 ComposeMessageActivity 中的 sendMessage。sendMessage 中判断了一下彩信附件大小等，如果不符合，就重新设置发送按钮状态并且退出这个方法。如果符合一些列要求，就调用 WorkingMessage 中的 send 方法。
2. WorkingMessage.java 的 send 方法中，判断是彩信还是短信，短信走短信流程，彩信走彩信流程。我们这边是短信，就调用 preSendSmsWorker。在这个方法中首先调用 sendSmsWorker 发送信息，然后调用 deleteDraftSmsMessage，删除草稿。在 sendSmsWorker 方法中，将联系人通过“;”截取为联系人数组。然后调用 SmsMessageSender.java 的 sendMessage.
3. SmsMessageSender.java 的 sendMessage 调用 queueMessage。QueueMessage 中通过联系人数组循环逐条发送给每个联系人。首先将短信信息插入短信数据库，然后发送广播通知服务，发送信息。
4. SmsReceiver.java 是短信的广播接收器，接收到广播后通过 *beginStartingService*(context, intent)将信息传递给 SmsReceiverService.java。
5. SmsReceiverService.java 短信收发的服务类。OnCreate 中先初始化一个 Handler.**new** ServiceHandler(*sSmsTHandler*.getLooper());然后在 onStartCommand 中，将接收到的 Intent 信息发送给 ServiceHandler。在 ServiceHandler 对不同的 Action 进行不同的处理。一般会有 *MESSAGE_SENT_ACTION* 短信发送完，成功或者失败等各种状态的返回处理，最终显示到界面上。*ACTION_BOOT_COMPLETED* 开机后对短信的处理，高通的是自动发送 pending 中的短信，MTK 的是将 pending 队列中的短信都置为 failed。*ACTION_SEND_MESSAGE*，发送短信。我们这时候是 *ACTION_SEND_MESSAGE*，调用 handleSendMessage。

```
private void handleSendMessage() {
        MmsLog.d(MmsApp.TXN_TAG, "handleSendMessage()");
        if (!mSending) {
            sendFirstQueuedMessage();
        }
      };
```

handleSendMessage 中先判断是否有短信在发送中，没有就发送 pengding 队列中的第一条。这时候调用 sendFirstQueuedMessage。sendFirstQueuedMessage 方法中，从数据库拿出第一条短信，然后传递给 SmsSingleRecipientSender.java 的 sendMessage，进行发送。

6. SmsSingleRecipientSender.java 上层和 framework 层的接口，在 sendMessage 中调用 framework 层中的 SmsManager.java。将数据传递下去。这边调用了 SmsManager.java 中的 sendMultipartTextMessageWithEncodingType 方法。

Framework 层流程图：



7. sendMultipartTextMessageWithEncodingType 方法。
8. SmsManager.java 中的 sendMultipartTextMessageWithEncodingType 中先通过 ISms.Stub.*asInterface*(ServiceManager.*getService*("isms"));获取系统服务。UiccSmsController.java 注册了 id 为“isms”的服务。
9. UiccSmsController.java 中调用了 sendTextWithEncodingTypeForSubscriber 在该方法中调

用 IccSmsInterfaceManager.java 中的 sendTextWithEncodingType 方法。

10. IccSmsInterfaceManager.java 中的 sendTextWithEncodingType。首先判断调用他的 APP 是否有发送短信的权限。如果有权限就调用 GsmSMSDispatcher.java 中 sendTextWithEncodingType 进行短信的发送。
11. GsmSMSDispatcher.java 中的 sendTextWithEncodingType 首先将短信通过设置的编码格式，进行 PDU 打包。然后调用 SMSDispatcher.java 中的 sendRawPdu
12. SMSDispatcher.java 中的 sendRawPdu 回调 GsmSMSDispatcher.java 中的 sendSms 然后调用 RIL.java 中的 sendSMS。
13. RIL.java 中的 RILSender 接收到 *EVENT_SEND* 通过 Socket 将短信下发。

2.彩信发送流程：

APP 层流程图

1. ComposeMessageActivity.java 中点击发送按钮后发送。在发送之前进行附件大小，有无联系人的判断。然后调用 WorkingMessage 中的 send 进行彩信的发送。

2. WorkingMessage.java 中的 send 方法，判断为彩信。开启一个线程，处理彩信发送流程。首先调用 `final SendReq sendReq = makeSendReq(spliter.getMMSConversation(), subject, mmsCc);` 将附件，主题等打包成彩信可发送的 **Pdu_SendReq**，然后调用 sendMmsWorker 方法，接着会把彩信写入数据库，把要发送的 SendReq 也会写入数据库，后面会再从数据库中读取出 SendReq，并标识为草稿；然后会构

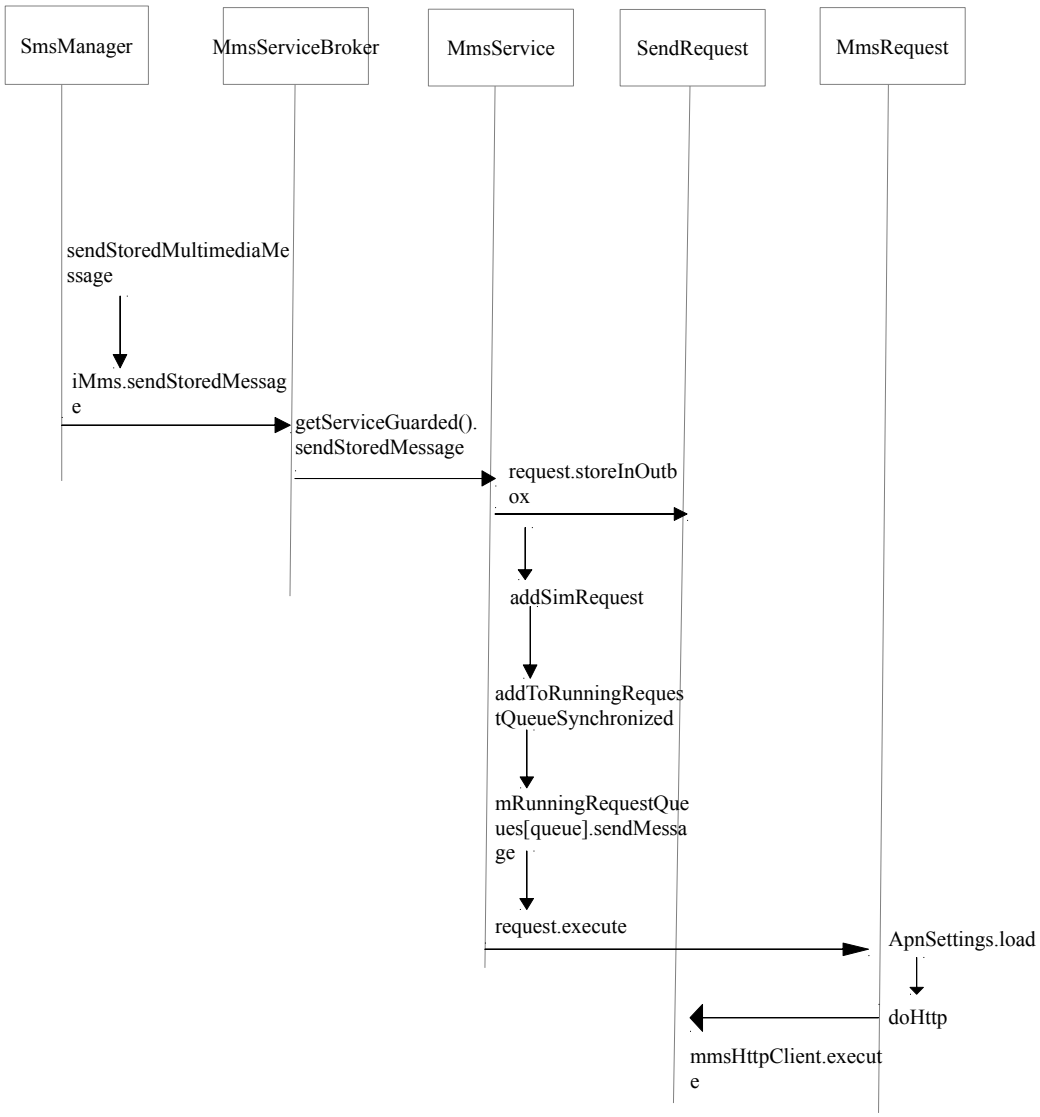
建 MmsMessageSender，传入收信人和彩信的 Uri，调用 sendMessage 让其发送。这期间也会回调 UI 一次，以初始化收信人编辑框和信息编辑框。

3. MmsMessageSender.java 中的 sendMessage 先从数据库中读取生成 Pdu 然后根据当前彩信的配置和其他信息对 SendReq 进行更新，比如设置 Expiration，Priority，Date 和 Size 等，把彩信移到 Outbox，然后启动 TransactionService 来处理彩信。sendMessage()就此返回。WorkingMessage 会再次回调 UI 的接口，因为此时彩 信已被在数据库中，所以 UI 会刷新信息列表，显示刚刚的彩信，这时的状态应该是正在发送中。

4. *TransactionService.java* 服务开始 onStartCommand 然后发送 message *EVENT_NEW_INTENT* ServiceHandler 接收到 message 调用 onNewIntent 中判断是否有网络然后调用 launchTransaction 启动事务来发送消息。ServiceHandler 接收到 *EVENT_TRANSACTION_REQUEST* 接着初始化 SendTransaction 并且传递一些参数过去然后调用 processTransaction 管理发送队列，调用正在处理的 SendTransaction 的 process 方法。

5. SendTransaction.java 的 process 开启一个线程处理发送服务。在线程的 run 方法中调用 SmsManager 中的 sendStoredMultimediaMessage 发送彩信。

Framework 层流程图：



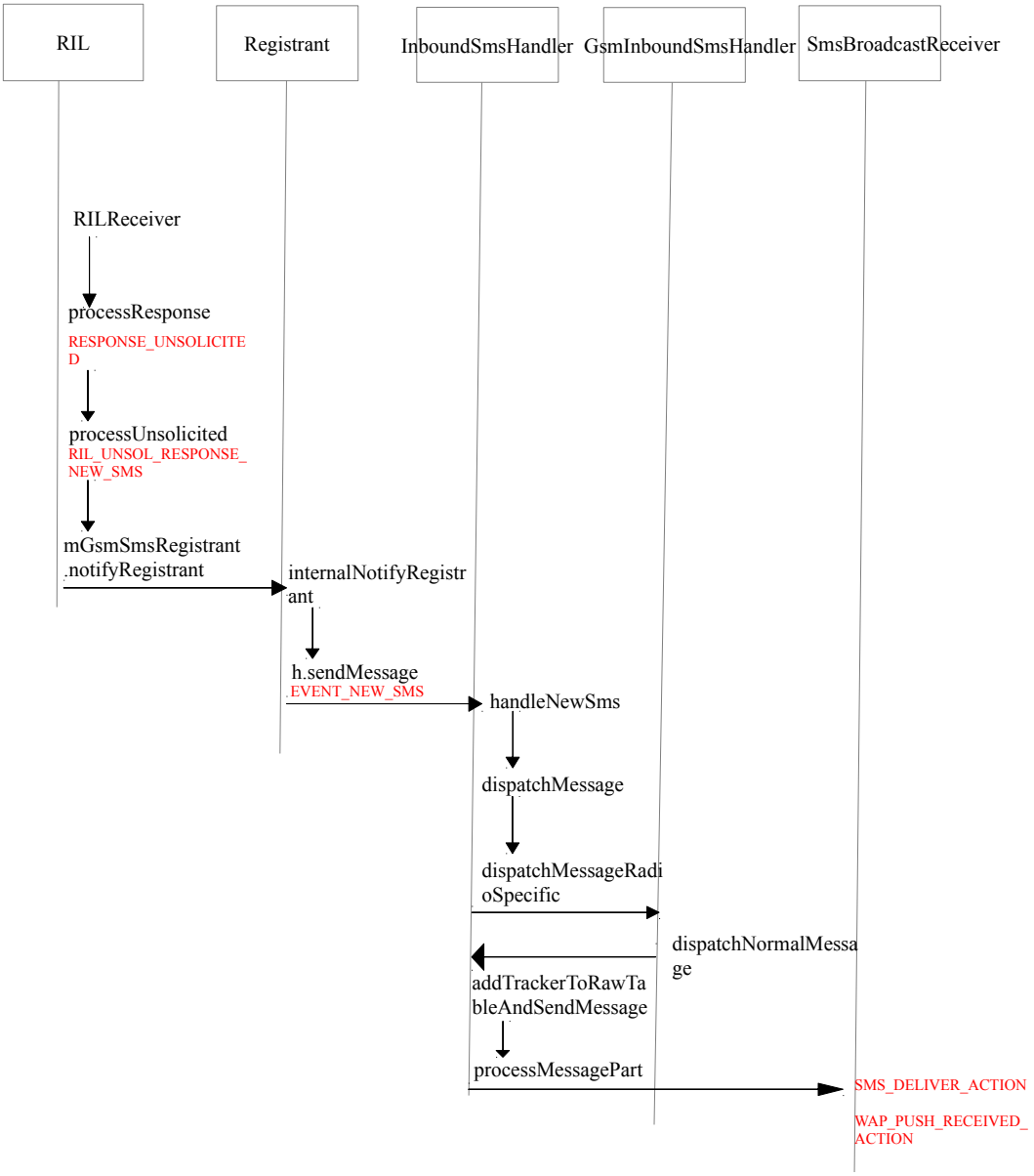
6. SendStoredMultimediaMessage.java 的 sendStoredMultimediaMessage 中首先获取 MMS 的系统服务 MmsServiceBroker 调用方法 sendStoredMessage 发送彩信。

7. MmsServiceBroker.javas 的 sendStoredMessage 方法中首先判断是否有发送彩信的权限。如果有权限就调用 MmsService 的 sendStoredMessage 方法。

8. MmsService.java 的 sendStoredMessage 方法中首先获取 pdu 数据，如果有数据就初始化 SendRequest.java 然后调用 SendRequest 的 storeInOutbox 更新数据库。然后调用 addSimRequest 将 MmsRequest 加入队列。第一条的话就调用 addToRunningRequestQueueSynchronized 加入发送队列，不是第一条加入等待队列。AddToRunningRequestQueueSynchronized 中发送 message 被 RequestQueue 接收。然后调用 MmsRequest 中的 execute 进行彩信的发送。

9. MmsRequest.java 中的 execute 先调用 doHttp 通过 APN 连接网络将彩信发送给彩信中心并且返回连接信息，然后 checkResponse 检测返回信息，最后将返回的状态通过 processResult 方法，通过 Intent 向上传递，最终显示到界面上。

3.短彩信接收流程



1.在 Ril.java 中定义了一个 receive 的框架：当接收到短信时，底层首先通过 rild 将接收到的短信通过 socket 传送给 Ril.java。RILReceiver 这个线程一直和守护进程 rild 进行 sorcket 通信，并获取守护进程上报的数据。然后调用 processResponse。

```
private void
    processResponse (Parcel p) {
        int type;

        type = p.readInt();

        if (type == RESPONSE_UNSOLICITED) {
            processUnsolicited (p);
        } else if (type == RESPONSE_SOLICITED) {
            RILRequest rr = processSolicited (p);
            if (rr != null) {
                rr.release();
                decrementWakeLock();
            }
        }
    }
```

通过 processResponse 方法的代码可以看出在上报数据进行了分类处理，*RESPONSE_UNSOLICITED* 表示接收到数据就直接上报的类型，主动上报，如网络状态和短信、来电等等。 *RESPONSE_SOLICITED* 是必须先请求然后才响应的类型。当然这里是短信接收肯定会走前者。processUnsolicited 该方法会根 据当前的请求的类型调用对应方法，如果是接收到短信则是 *RIL_UNSOL_RESPONSE_NEW_SMS*。调用代码如下：

```
case RIL_UNSOL_RESPONSE_NEW_SMS: {
                if (RILJ_LOGD) unsljLog(response);

                // FIXME this should move up a layer
                String a[] = new String[2];

                a[1] = (String)ret;

                SmsMessage sms;

                sms = SmsMessage.newFromCMT(a);
                if (mGsmSmsRegistrant != null) {
                    mGsmSmsRegistrant
                        .notifyRegistrant(new AsyncResult(null, sms, null));
                }
            break;
```

追溯该方法，mGsmSmsRegistrant 对象的创建过程 mGsmSmsRegistrant 是 BaseCommands 的成员变量，且在调用 setOnNewGsmSms（）方法来赋值的，BaseCommands 是 Ril.java 的父类。最后发现调用 setOnNewGsmSms（）该方法的地方是:GsmInboundSmsHandler.java 中

```
phone.mCi.setOnNewGsmSms(getHandler(), EVENT_NEW_SMS, null);
```

GsmInboundSmsHandler.java 又是 InboundSmsHandler 的子类因此 getHandler()获取到的是 InboundSmsHandler 中的 handler.

再看 mGsmSmsRegistrant.notifyRegistrant(new AsyncResult(null, sms, null));这个方法调用了 Registrant.java 中的 notifyRegistrant。

2.Registrant.java 中的 notifyRegistrant 有调用了 internalNotifyRegistrant。在

internalNotifyRegistrant 方法中发送了一个 message 给 handler 并且 message.what = *EVENT_NEW_SMS*。*EVENT_NEW_SMS* 这个是与上述 handler 一起传过来的。这边的 handler 就是上面所说到的 InboundSmsHandler 中的 handler

3.InboundSmsHandler 中的 DeliveringState 内部类中 processMessage。因为 msg.what 为：*EVENT_NEW_SMS* 因此调用

```
switch (msg.what) {
                case EVENT_NEW_SMS:
                    // handle new SMS from RIL
                    handleNewSms((AsyncResult) msg.obj);
                    sendMessage(EVENT_RETURN_TO_IDLE);
                    return HANDLED;
```

handleNewSms 方法中调用 dispatchMessage 它有调用 dispatchMessageRadioSpecific。该方法在子类 GsmInboundSmsHandler 中是一个回调。该方法有调用父亲类 InboundSmsHandler 中的 dispatchNormalMessage 该类中进行短信和彩信的区分。并将信息打包成 InboundSmsTracker。传递给 addTrackerToRawTableAndSendMessage 该方法中同过 message.*EVENT_BROADCAST_SMS* 将信息发送给 handler 接收：

```
case EVENT_BROADCAST_SMS:
                    // if any broadcasts were sent, transition to waiting state
                    if (processMessagePart((InboundSmsTracker) msg.obj)) {
                        transitionTo(mWaitingState);
                    }
                    return HANDLED;
```

handler 接收到信息后将 InboundSmsTracker 数据传递给方法 processMessagePart。该方法对应短信和彩信发送不同广播给 APP 层，进行短彩信的接收处理。短信的广播为 SMS_DELIVER_ACTION。彩信的广播为 WAP_PUSH_DELIVER_ACTION。短信中 SmsReceiver 接收到广播交给服务 SmsReceiverService 处理，进行铃声通知，将短信存入数据库等。彩信中 PushReceiver 接收到广播交给服务 MmsPushReceiveService 处理，进行铃声通知，将彩信的下载链接存入数据库等。

4.短彩信中的一些问题

1.MMS 和 SMS 的区别：

彩信是用 Http 协议进行发送的，短信是通过短信协议发送。彩信接收是先走短信协议收到一条 Http 链接，然后通过 Http 协议，下载彩信内容。

2.超时重连机制（如何判断失败）：

短信将信息传给 RIL 层发送后会后短信发送成功或者失败原因的状态码返回。然后通过上层传过来的 PendingIntent 将发送成功与否的状态返回上层，并把状态信息插入短信数据库，并且显示到界面。

彩信会设置链接网络的时间，超时，没有 SIM 卡等，会返回发送失败的状态码，也通过上层传送过来的 PendingIntent 将信息传给上层，并把状态信息插入数据库，并且显示到界面。