IJCSN International Journal of Computer Science and Network, Volume 4, Issue 3, June 2015
ISSN (Online) : 2277-5420 www.IJCSN.org
Impact Factor: 0.417

Inter-Process Dependencies in Lollipop Call Sequences

[1] Venkata Sanyasi Rao Sasipalli, [2] Zulfadhli Bin Zainuddin, [3] Krishnam Raju Gottumukkala

[1] Center for Excellence in Computer Technology
Hiroshima City, Hiroshima, 7300048, Japan

[2] RAMTEJ Technologies Corp.,
Hiroshima City, Hiroshima, 7300048, Japan

[3] Center for Excellence in Computer Technology
Visakhapatnam, Andhra Pradesh, 531001, India

Abstract - Dialer / Call application is very important among all the applications in any mobile phone and essential for all users. Call failures are common though, can be reduced by examining and tightly knotted with proper inter-process communications. In this paper we examined call sequence differences of Kitkat and Lollipop and their inter-process communications. Discussed the dependencies of inter-process communications by pointing the crucial changes, pitfalls, and risks in call handling and connectivity thereof. Concluded with some suggestions to reduce inter-process dependencies to improve call success rate.

Keywords - *Call success rate, Call Sequence, Inter-process Communication, Dependency.*

1. Introduction

In telecommunications a call attempt in call application invokes a call setup procedure, which, if successful, results in a connected call. A call setup procedure may fail due to a number of technical reasons. Such calls are classified as failed call attempts. In many practical cases this definition was expanded with a number of detailed specifications describing which calls exactly are counted as successfully set up and which not [1]. Call handling sequences differ from manufacturer to manufacturer, and are the focus elements to examine for success / failure of calls. Call setup success rate (CSSR) is the fraction of the attempts to make a call that result in a connection to the dialed number. This fraction is usually measured as a percentage of all call attempts made. In designing the call handling application, call sequences play an important role to send or receive a call successfully.

In this paper, we examine call sequences of call application that use open source modules of OS and binaries / libraries. Android Open Source Project (AOSP) is an initiative of Open Handset Alliance (OHA, led by Google. Android software stack is created for a wide array of devices with different form factors. The primary purposes of Android are to create an open software platform available for carriers, OEMs, and developers to make their innovative ideas a reality and to introduce a successful, real-world product that improves the mobile experience for users [2].

Today, carriers, device manufacturers and developers exploit this innovative platform to create applications. Figure 1 depicts the Android stack diagram with each block containing the libraries provided to support the hardware and application development.

With the advancements of network technology the Dialer / Call application (in Application block in Figure 1) itself provides wider scope to explore the implementation of the technology improvements of various network types. With the introduction of various network types, 3G, 4G, and their Variants, the carriers and OEMs responded aggressively to the architectural changes in the network implementations in Dialer / Call application. There are major improvements, like Look & Feel, Security, Recent Apps and Settings in the upgrade from Kitkat to Lollipop, we look into the call sequences in Dialer / Call application only. Note that in the Android stack diagram (Figure 1) Dialer / Call and Data Transmission requests / responses go out of the device and expect network involvement. We can control the call flow within the device by properly designing the logics.

There are many improvements from version to version though, we focus on the changes from Kitkat to Lollipop version only, as a large set of improvements on both the application side as well network side were taken place in Lollipop.

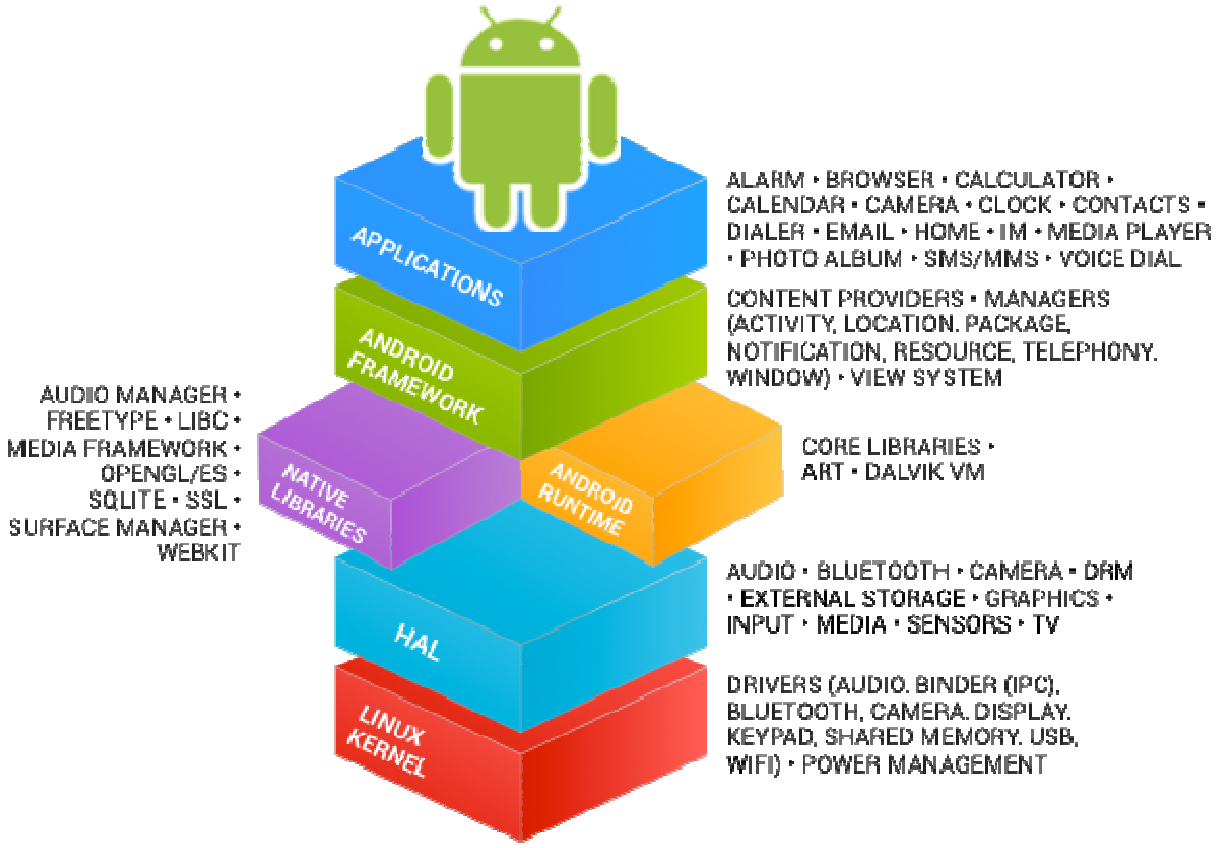

Figure 1: Android Stack

In this paper, we examined the inter-process communications and their dependencies in call sequences in both outgoing and incoming calls. Further, in this paper we provided some suggestions to mitigate the burden resulted from the inter-process dependencies, and also provided direction for further improvements. Our focus is on processes ported from Open Source Modules (OSM) that are available for public use, and the devices such as Nexus, which uses OSM.

2. Dialer / Call Application

Dialer / Call Application is essential among all applications that a mobile phone offers. This application is used very frequently to send or receive phone calls from / to users. Such of this application has to be very well designed and developed. This development aggressively uses or dependent heavily on the hardware component called modem, which sends requests to the network and receives responses from the network.

All the modem manufacturers provide carrier-specific apps to interact with their modem. Hardware component suppliers vary for each model of the device, so let us check a random model Nexus 6 and its hardware components in the below table for reference [3].

Table 1: Hardware Components for Nexus 6 for Android 5.0 (Lollipop)

Hardware Component	Offering Company
NFC, Bluetooth, WiFi	Broadcom
Media, Audio, Thermal, Touch Screen, Sensors	Motorola
GPS, Audio, Camera, Gestures, Graphics, DRM, Video, Sensors, Modem	Qualcomm

Depending on the hardware component the binary to be used changes and the programming logic changes thereof which, influences the inter-process communication and the dependency. It is clear for Dialer / Call application, two aspects influence the programming logic; they are Hardware components and Network types that are supported by Android version.

Upgrade from Kitkat to Lollipop brought a rich experience in many perspectives including the call application [4]. When upgrading, we are agreeing to accept the new behavior which is different from Kitkat. Below is a general call flow diagram from Users to Networks and vice-versa.

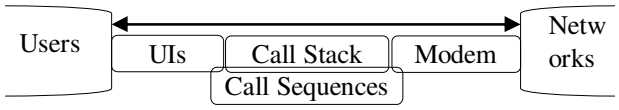

Figure 2: Call flow between User and Network

2.1 User & UIs

In Dialer / Call application, User is a caller or receiver. From the technical point of view, user is an entity that initiates an intent through User Interface (UI). Once an intent is initiated in the Dialer / Call application UI, instructions are passed to Call Stack through notifications. The flow of communication either synchronous or asynchronous starts from (User) UI component then to Call Stack and then from modem to network. Call success rate lightly dependent on User / UIs, but heavily dependent on later modules, Call Stack, Modem and carrier-specific modem support libraries.

2.2 Call Stack

From web definition, a call stack is mainly intended to keep track of the point to which each active subroutine should return control when it finishes executing [5]. Call stack we mean here (for brevity) with similar perspective but a bunch of intents that take place between UIs and Modem when a mobile user initiates a call intent. Why we use the work Call Stack is to give an imitative meaning of Synchronous and Asynchronous communication.

2.3 Vendor

Vendor is hardware supplier for a particular model of the mobile phone. Here our vendor is nothing but a modem of the mobile phone and its supporting libraries, in the software point of view.

3. Sequences

3.1 Kitkat Call Send or Outgoing Sequence

Once a user initiates phone call, a call intent is created in the Dialer / Call application and a notification is sent to com.android.phone process. Here, a broadcast intent is initiated and a notification is sent to com.android.internal.telephony. Calling in 3G or 4G or its variants is verified here. For 3G call, notification is sent to the modem straight from here. Otherwise, a notification is sent to IMS support carrier-specific module, for example, org.codeaurora.ims (for IMS call variant), and then the notification is sent to modem to place the call. This is a straightforward process without any synchronous or asynchronous inter-process communications.

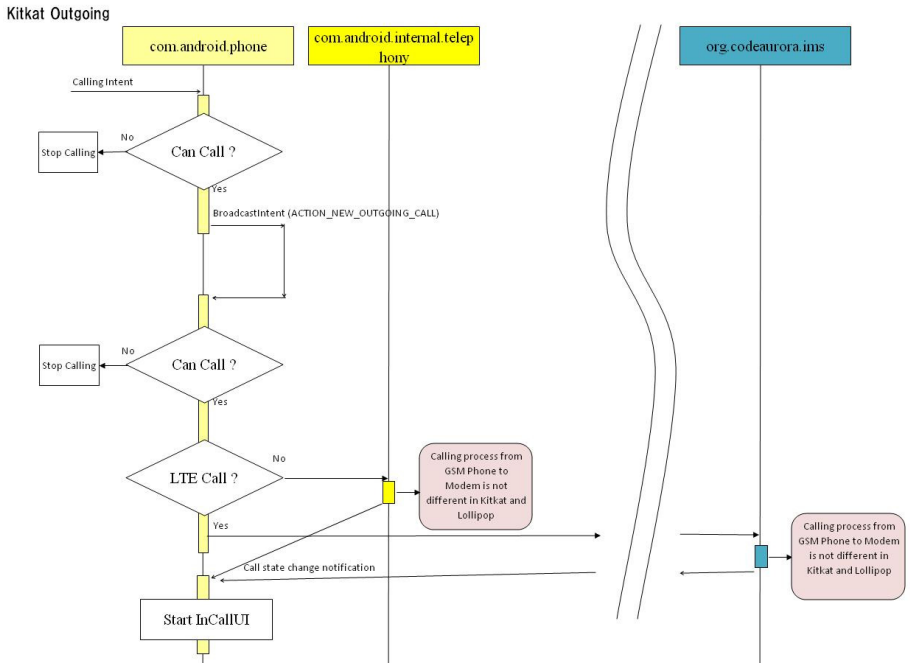

Figure 3: Kitkat Outgoing Call Sequence

However, we should consider minimum bugs for each traverse through a process. Note that there are many claims on phone process in the internet, such as, "Unfortunately the process com.android.phone has stopped" [6]. Therefore, at each process stage, fail probability is higher as carrier-specific application interleaves the call process.

3.2 Lollipop Call Send or Outgoing Sequence

In Lollipop call send or outgoing sequence, the call intent notification is first sent to com.android.server.telecom, where phone state is verified to check whether the call can be placed or not. Calling screen intent is initiated here itself.

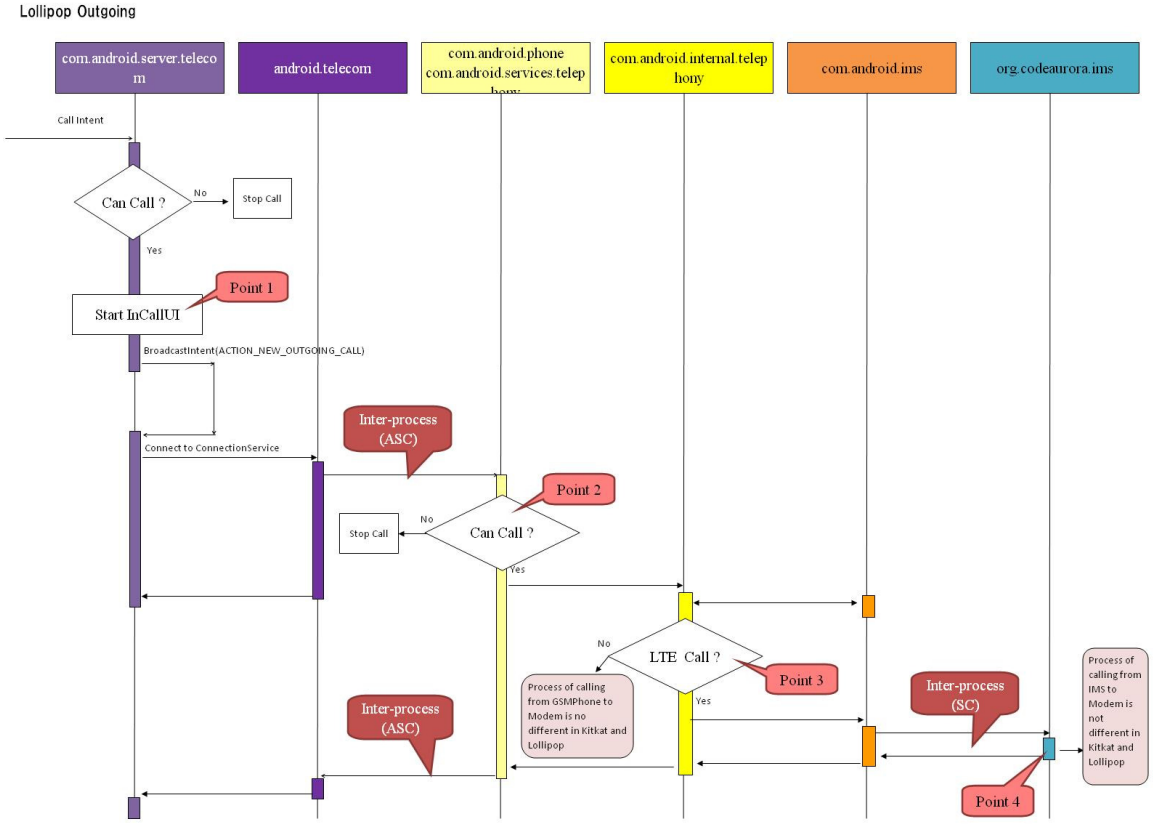

Figure 4: Lollipop Outgoing Call Sequence

This is the designers' choice to engage the user with phone. Note that ACTION_CALL is broadcasted after the call screen is initiated before com.android.phone process. While the flow depends heavily on asynchronous inter-process communication, the notification is taken forward to check the network variants, say LTE. One good practice is adopted here to use synchronous communication between android.ims and carrier's ims processes. Sending notification to modem is same as that of Kitkat after these verifications.

3.3 Kitkat Call Receive or Incoming Sequence

Incoming call sequence in Kitkat is simple and straightforward, however, different OEMs use different end modules for Call / Incall Screens, we used the standard com.android.incallui for examination in this paper. The key factor in incoming sequence is Call Reject timing. In Kitkat, call reject is verified in com.android.phone.

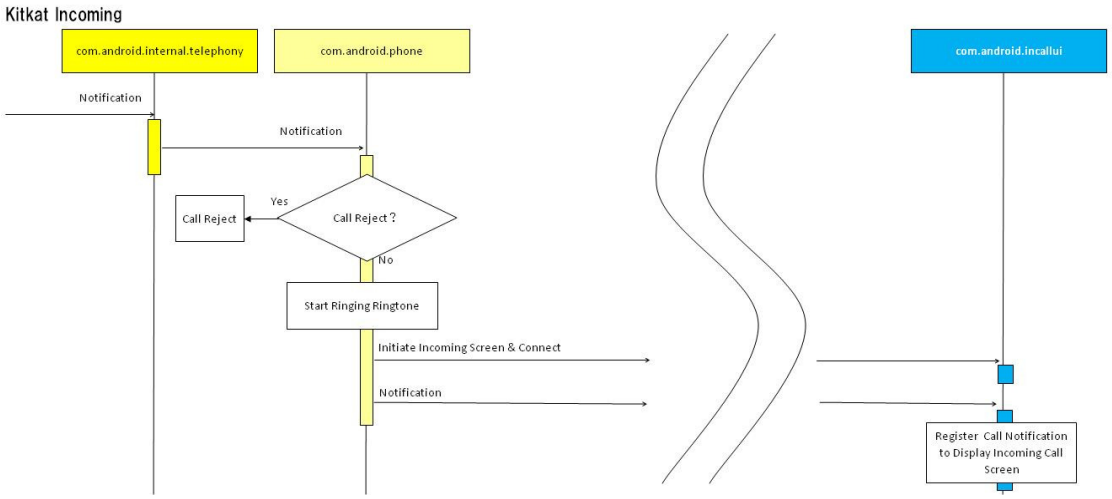

Figure 5: Kitkat Incoming Call Sequence

After Call Reject verification a notification is sent to InCallUI to register it for displaying Incall Screen.

3.4 Lollipop Call Receive or Incoming Sequences

Lollipop incoming call sequence is also an interesting case study. The notification is sent from internal.telephony to services.telephony. It relies on android.telecom asynchronously to initiate connection service. Even the connection service fails to send back its response, Call Reject is verified. Here is a pitfall resulted due to asynchronous communication.

Call generated notification is sent to services.telecom to initiate ringing a ringtone. And finally Incoming Call screen is displayed after receiving call generated notification asynchronously from andnroid.telecom. Intelligently, back-end connection channel and front-end screen channels are separated, so asynchronous is essential, but it is prone to operational bugs, which damages satisfied user experience.

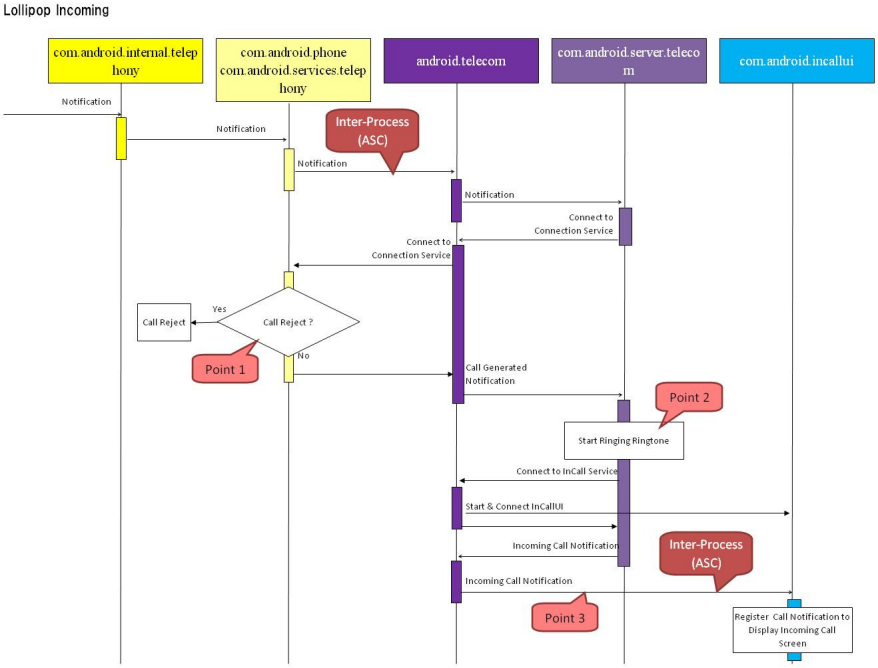

Figure 6: Lollipop Incoming Call Sequence

While observing this sequence, check the main points, Call Reject location is different in Kitkat and Lollipop, this doesn't affect much in the flow, but just a pitfall in designing, which has both advantage and disadvantages. Process of playing ringtone is shifted to services.telecom. From the front-end view, ringtone check is not given important in this sequence. services.telecom is made responsible for an interface for android.incallui.

4. Discussions

For outgoing call sequence in Lollipop, Calling screen initiated immediately after receiving call intent, this is bit early. Calling screen can be initiated after connecting to service android.telecom. This can avoid one asynchronous communication and hence can increase CSSR.

Possibility of call placement is verified here again. In Kitkat case also comes to here. Since the call does not take place in com.android.phone and since Call (Call decision) cannot be done in com.android.phone it is processed after sending the notification. This is again a pitfall and requires good logic to properly handle these processes.

Call type (3G / 4G / Variants) is verified in com.android.internal.telephony. In Kitkat it is done in com.android.phone itself and customized thereof. Android non-standard design logics used for adding carrier-specific ims app here. Up to Kitkat it was offered in the form of IMS Phone. From Lollipop, it has been implemented in-line with an interface of /system/framework/ims-common.jar which is prepared by the Android standard. This is prone to heavy crash bugs [7]. Inter-process asynchronous communication among android.telecom, com.android.phone, com.android.ims is prone to failures.

For incoming call sequences in Lollipop compared to Kitkat, Call Reject location is different in Kitkat and Lollipop. Process of playing ringtone is shifted to com.android.server.telecom, and for ACTION_CALL intent there are many problems. And android.telecom is made responsible for an interface to initiate Incall Screen but inter-dependent on its predecessor processes asynchronously. This traverse among processes, leads to various bugs and quality reduction.

5. Conclusions

This study of outgoing and incoming call sequences resulted to a better way of designing the call sequences as the outside factors, such as modem, poses a big threat and prone to many bugs and call dropouts, hence reduction in CSSR. Since the inter-process communication is mostly asynchronous, probability of non-dependency on resulted values is high. Minimizing the inter-process dependency is inevitable, however, it is not that much easy when working with structured Android modules, dealing many network types and carrier-specific connection apps. Overall, in Lollipop, it is clear that Android provided an opportunity for designers to engage the user with phone, by asynchronous kickbacks, not let him wait until the notification is sent successfully to the modem. More dilution of the processes gears up for failures and reduction of CSSR. Hence, a better way to work with Android call modules to process different network variants and carrier-specific connection apps is very necessary to discuss.

6. Future Work

Inter-process dependencies cannot be avoided, however, it can be addressed with different perspective by giving a single module, which can deal (1) necessary Android internal processes, (2) all the types of networks & their variants and (3) carrier-specific connection apps need to be proposed. This proposal is given in another paper.

Acknowledgments

We acknowledge our deepest gratitude to RAMTEJ Technologies Corporation for its financial support for this project. Grant No. RTG-RD-CJ01-2015. Also for providing part of the internal code (though open source but customized) to carry our study / examination in this project.

References

[1] WikiPedia, "Call Setup Success Rate" http://en.wikipedia.org/wiki/Call_setup_success_rate
[2] Android System Architecture: http://source.android.com/source/index.html
[3] "Nexus Files for Developers", https://developers.google.com/android/nexus/drivers
[4] Kris Carlon, "Android Lollipop vs Android KitKat comparison: what's different?" http://www.androidpit.com/android-lollipop-vs-android-kitkat
[5] Def., Call Stack, https://quizlet.com/14409191/technopedia-terms-flash-cards/
[6] Android Forums, http://androidforums.com/threads/how-to-solve-error-unfortunately-the-process-android-process-acore-has-stopped.854534
[7] GitHub "android: application crashes on upload - lolipop" issue 583 https://github.com/camlistore/camlistore/issues/583 Mar 2015

Venkata Sanyasi Rao Sasipalli is a lead researcher at Center for Excellence in Computer Technology and Principal Advisor to RAMTEJ Technologies Corporation, Hiroshima, Japan. He received Doctor of Engineering from Hiroshima University, Masters

from Technical University of Kaiserslautern, Germany and Andhra University, India. His research interests include Computer Engineering, Data Approximations, Antenna Systems and RFID Technology.

Zulfadhli Bin Zainuddin is an engineer working on developing architectural designs for mobile applications at RAMTEJ Technologies Corp., He received his bachelors degree from University of Malaya, Malaysia. His research interests include programming languages, logical designing and security network.

Krishnam Raju Gottumukkala is an advisor at Center for Excellence in Computer Technology, Visakhapatnam, India. He is a professor emeritus of Engineering Mathematics, Andhra University. His research interests span to computer engineering, CFD, and Optimization techniques.