

Event-Driven Android Framework for Context-Aware Automatic SMS and WhatsApp Replies Featuring Dual-SIM Routing, Call-Waiting Handling, and On-Device Privacy

Diptesh Dilip Dange¹, Karan Chandarakant Jadhav²,
Pratiksha Shivaji Patil³, Sahil Dipak Lonkar⁴

^{1,4}Department of Information Technology, Government College of Engineering Karad

^{2,3}Computer Science & Engineering, DY Patil college of Engineering and Technology, Kolhapur

Abstract

Missed calls are a direct source of lost business for small enterprises and independent professionals, yet a dependable automatic reply tool is difficult to build on present-day Android. Telephony broadcasts carry limited detail, handsets frequently contain two SIM cards, WhatsApp and WhatsApp Business may be installed side by side, WhatsApp drops the caption text of shared documents, and the security modules of banking applications may object to accessibility services that have a wide scope. In this paper, the design of a pair of Android applications is described: a User App that sends the replies and an Admin App that controls accounts and licences. Kotlin, Jetpack Compose, a layered clean architecture, and Firebase are used for the implementation.

Five design elements are presented. These are a concurrent call-waiting finite state machine that follows a main call and an interrupting call separately; a binding of the registered business number to incoming-call filtering, SIM-specific SMS sending, and WhatsApp client selection through direct and elimination matching; a two-phase accessibility routine that delivers a document first and its text afterwards; a strictly ordered decision cascade in which licensing and privacy checks take precedence over user preferences; and an edge-resident data policy under which call logs, contacts, and message text remain on the handset while only identity, licence, and session information is stored in the cloud. The design is examined through scenario walk-throughs and a qualitative capability comparison, and field measurements are left for future work.

Keywords: Android Telephony, Automated Call Response, Dual-SIM Routing, Finite State Machine, Accessibility Service, Edge Privacy, Jetpack Compose, Clean Architecture, Firebase, Subscription Licensing

1 Introduction

1.1 Motivation

Clinics, tutors, delivery agents, repair shops, and sales staff all rely on the telephone as the first point of contact with customers. A person who is driving, attending a meeting, or already speaking to another client

cannot pick up every call, and a caller who gets no reaction frequently moves on to a competitor. A prompt written message, sent as an ordinary Short Message Service (SMS) text or through an over-the-top service such as WhatsApp or WhatsApp Business, can hold the caller's interest until the owner becomes available. An automatic responder that performs this task dependably has to serve two parties. The owner needs control over who is contacted and when, while the vendor needs the ability to provision, suspend, and renew access from a distance. The system described in this paper addresses both needs through a User App that reacts to calls on the handset and an Admin App that handles customers and their subscriptions.

1.2 Engineering Challenges

Five practical difficulties are encountered when such a system is built for Android 7.0 to Android 15 (Application Programming Interface (API) levels 24 to 35).

1. **Dual-SIM ambiguity.** Many handsets carry a personal Subscriber Identity Module (SIM) and a business SIM. A listener that reacts to every call, together with the default system SMS route, may answer a personal call with a business message or charge the reply to the wrong number.
2. **Two WhatsApp clients.** If WhatsApp and WhatsApp Business are installed together, a generic send intent either displays a chooser or goes out through the unintended account.
3. **Call-waiting overwrite.** The telephony manager exposes only the idle, ringing, and off-hook states. When a second caller rings during an ongoing conversation, a simple tracker replaces the stored number of the first caller. As a result, the interrupting caller never receives a reply and the first caller may receive an incorrect follow-up.
4. **Captions on documents.** When a Portable Document Format (PDF) file is shared with WhatsApp through a standard send intent, the accompanying text is not displayed on the preview screen, in contrast to image and video shares.
5. **Security software conflicts.** Protection libraries bundled with some banking and payment applications may react to enabled accessibility services, particularly those that request broad inspection of windows or are not restricted to named packages.

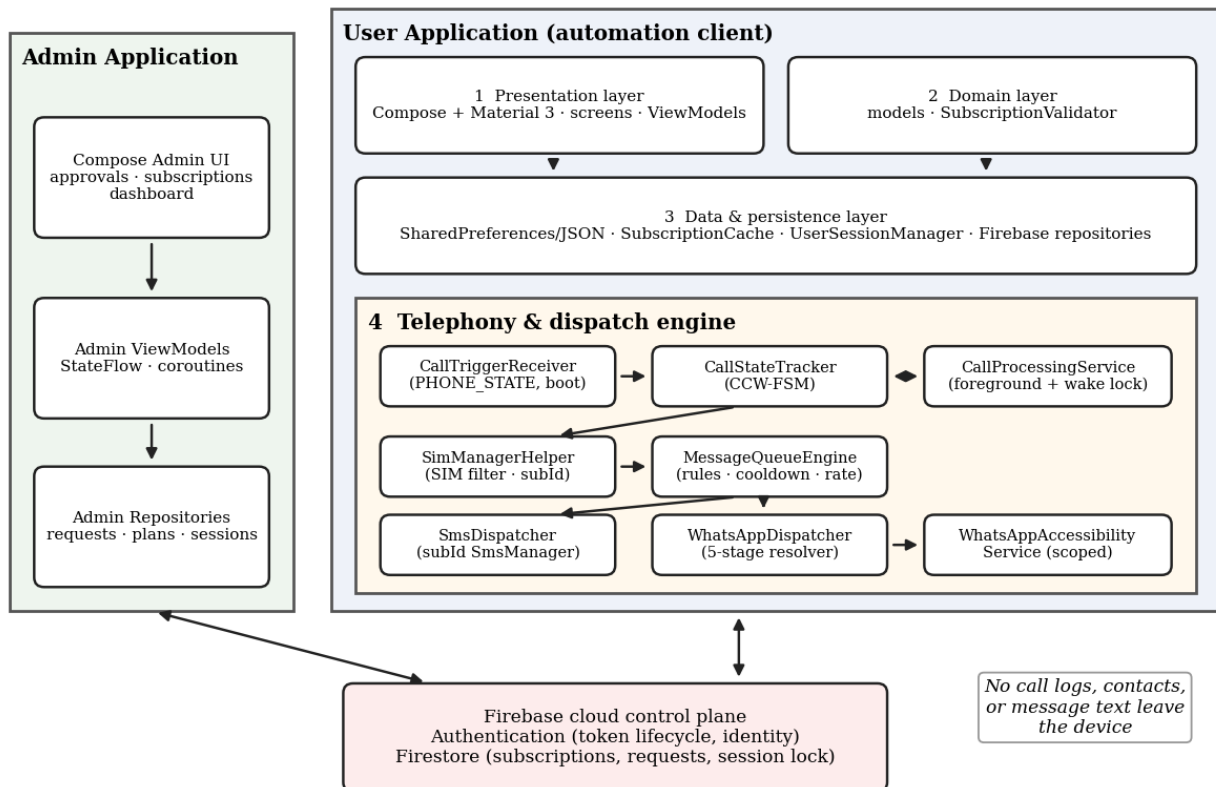
1.3 Contributions

The main contributions of the work are listed below.

- A two-application architecture with four internal layers and a shared cloud control plane.
- A concurrent call-waiting finite state machine (CCW-FSM) that keeps independent primary and secondary contexts.
- Registered-number binding that is applied consistently to incoming-call filtering, SMS routing, and WhatsApp client resolution.
- A two-phase accessibility routine restricted to WhatsApp package names and free of interactive-window inspection.
- An ordered decision cascade, supported by a message history that records only what was actually delivered.
- An edge-resident data policy combined with cloud-enforced licensing and single-device sessions.

The remainder of the paper is organised as follows. Section 2 presents background concepts, Section 3 lists the requirements, and Section 4 describes the architecture. The methods are detailed in Section 5, the administrative plane in Section 6, the technology stack in Section 7, and the end-to-end pipeline in Section 8. The design is analysed in Section 9, and Section 10 concludes the paper. An overview of the architecture is given in Figure 1.

Figure 1: Architecture of the Proposed System with Admin Application, Layered User Application, and Firebase Cloud Control Plane



2 Background and Related Concepts

2.1 Telephony and Subscription APIs

Call progress is reported by the Android telephony manager through three states, namely idle, ringing, and off-hook [1]. When a device contains several SIM cards, subscription records describe every active line, and an SMS manager can be created for one particular subscription so that the message leaves through the chosen number [2, 3]. The SIM-aware behaviour of the proposed system rests on these interfaces.

2.2 Accessibility Services

An accessibility service can observe events in the user interface (UI) and perform actions on behalf of the user. Since this capability is powerful, platform policy requires clear disclosure and user consent, and a service can be limited to specific packages through its configuration [4, 5]. In the present design, this restriction is regarded as a security feature and not as an afterthought.

2.3 Layered Architecture and Reactive State

Clean architecture keeps business rules independent of UI and storage frameworks by separating policy from mechanism [6]. Official Android guidance recommends unidirectional data flow, in which the interface observes immutable state and passes events upward [7]. Both ideas are adopted in the system.

2.4 Event-Driven Integration and Role Separation

Message-oriented integration patterns explain how events are filtered, routed, and transformed by a chain of small components [8], and the call pipeline follows the same model. Keeping privileged administration

apart from ordinary use is an established access-control principle [9], which is reflected in the division into two applications.

2.5 Managed Cloud Services

Hosted authentication and document database services provide token refresh, offline persistence, and real-time listeners [10, 11]. These features allow licences to be enforced remotely while private call data is never stored in the cloud.

3 Requirements

Functional requirements. The system is required to satisfy the following.

- **FR1.** Authenticate users with credentials issued by an administrator.
- **FR2.** Permit only one active device per account.
- **FR3.** Detect missed calls, completed calls, and call-waiting events in the background.
- **FR4.** Act only on calls that are received on the registered SIM.
- **FR5.** Send SMS messages through the registered subscription.
- **FR6.** Select the correct WhatsApp client and send text, images, and documents with captions.
- **FR7.** Support templates for each call outcome, with placeholders.
- **FR8.** Honour a blacklist with the highest priority, an unknown-contacts option, business hours, cooldown periods, and rate limits.
- **FR9.** Maintain histories of recent calls and sent messages.
- **FR10.** Enforce yearly and lifetime subscriptions remotely.
- **FR11.** Provide the administrator with handling of registrations, plan switching, suspension, manual user creation, and live statistics.

Non-functional requirements. The system is required to survive reboots and removal of the task, to stay responsive to remote state changes, to report delivery status truthfully, to keep displayed statistics consistent with stored data, and to avoid collecting private communications in the cloud.

4 System Architecture

The system comprises two Android applications and a shared cloud control plane. The automation engine runs inside the User App on the subscriber's handset, whereas the Admin App, operated by the vendor, manages the life cycle of customers. Authentication and Firestore collections for subscriptions, registration requests, and single-device session locks are supplied by the cloud plane. The overall structure is shown in Figure 1, and the four layers of the User App are explained below.

4.1 Layers of the User App

Presentation layer. The interface is written entirely in Jetpack Compose with Material Design 3. Immutable state exposed by view models is observed by the screens, and user intents are passed upward. Whenever a subscription becomes expired or suspended, a lock overlay takes over navigation so that a lapsed account cannot reach any automation screen.

Domain layer. Plain Kotlin models (application settings, dispatch mode, queued message, subscription, and subscription state) and deterministic validation policies are placed in this layer, which is independent of any framework.

Data layer. Repositories connect the on-device storage with the cloud store. Settings, templates, queues, cooldown timestamps, histories, and a cached subscription record serialised as JavaScript Object Notation

(JSON) are held on the device, whereas the cloud store is used only for authentication and licence data.

Telephony and automation layer. An autonomous, event-driven subsystem is formed by a call-trigger receiver, a foreground call-processing service, the call-state tracker, a SIM helper, a message-queue engine, an SMS dispatcher, a WhatsApp dispatcher, and an accessibility service limited to WhatsApp. A boot receiver together with an alarm-based mechanism restarts the service after a device reboot or after the task is removed from the recent-apps list.

4.2 Admin App

A smaller version of the same pattern is followed by the Admin App, with a Compose interface, view models that expose reactive state, and repositories for approvals, subscriptions, and sessions. Administrators are authenticated separately with an email address and a password.

5. Methods

5.1 Concurrent Call-Waiting State Machine

A naive tracker keeps a single number and replaces it whenever a new ring arrives. In contrast, two contexts are maintained by the CCW-FSM. The primary context stores the active number, its SIM slot and subscription identifier, the time at which the call went off-hook, and a direction flag. The secondary context stores the waiting number, its slot and subscription identifier, the time at which ringing started, and a flag that records the occurrence of call waiting. The behaviour is illustrated in Figure 2 and listed in Table 1.

Figure 2: Concurrent Call-Waiting State Machine

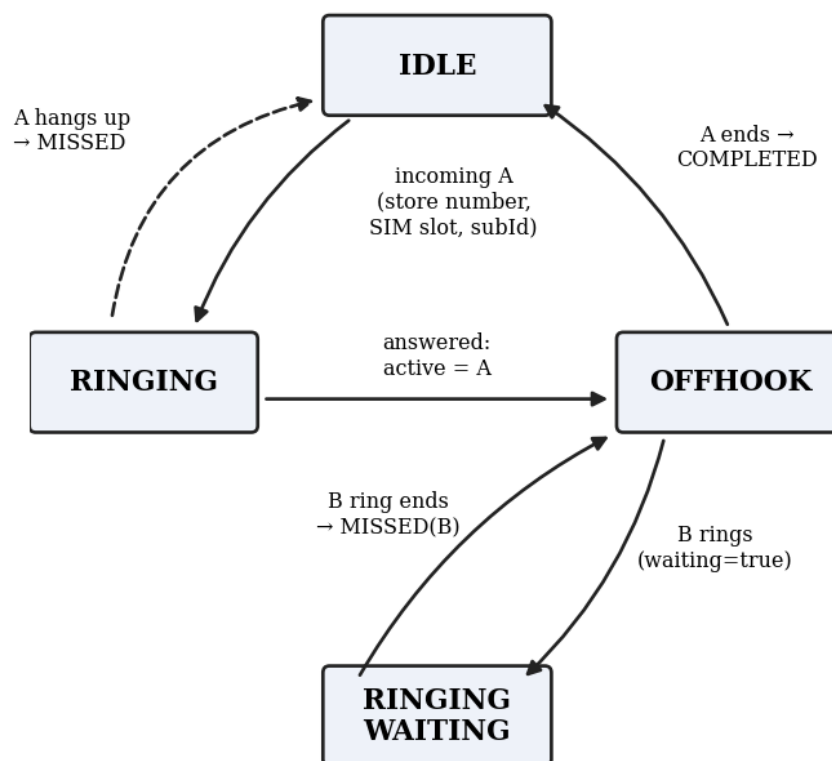


Table 1: CCW-FSM Transitions and Emitted Events

Transition	Effect
Idle to Ringing	The number, SIM slot, and subscription identifier of caller A are captured.
Ringing to Off-hook	Caller A is promoted to the active call.
Ringing to Idle	A ended the call before it was answered, so a missed or rejected event is emitted for A.
Off-hook to Ringing	A stays active, caller B is recorded in the secondary context, and the waiting flag is set.
Ringing to Off-hook (B stops)	The ring of B ended while A continues, so a missed event is emitted for B immediately.
Off-hook to Idle	An incoming-completed or outgoing-completed event is emitted for A, and both contexts are reset.

The key difference from simple trackers lies in the fourth and fifth rows of Table 1. A missed-call reply is sent to the interrupting caller as soon as the ringing stops, without waiting for the first conversation to finish, and a completed-call reply is still sent to the first caller afterwards.

5.2 Registered-Number Binding

Let P_{reg} denote the registered business number, and let $D10(\cdot)$ denote the function that retains only the last ten digits of a number, which makes the comparison independent of country-code formatting.

5.2.1 Incoming-Call Filtering

On receipt of a phone-state broadcast, the receiver reads the subscription and slot identifiers and compares them with the active subscriptions reported by the platform. For a call c , the decision is given by Eq. (1).

$$\text{AllowCall}(c) = \begin{cases} 1, & \text{if } |\text{ActiveSIMs}| \leq 1 \vee D10(\text{SIM}_c.\text{number}) = D10(P_{\text{reg}}) \\ 0, & \text{otherwise} \end{cases} \quad \text{----- (1)}$$

Hence a call received on any other SIM is discarded before it reaches the queue, and personal calls never trigger business replies.

5.2.2 Outbound SMS Binding

The default SMS manager is not used. Instead, the helper looks up the subscription whose number matches $D10(P_{\text{reg}})$ and builds a manager bound to that subscription, using the instance factory on Android 12 and later and the static subscription-based accessor on earlier releases. As a result, multipart messages leave through the registered line and are billed to it.

5.2.3 WhatsApp Client Resolution

When only one client is installed, it is selected immediately. Otherwise, a four-stage search is carried out in order: (a) the accounts held by the system account manager for each client, (b) the profile card of the device owner, (c) the synced raw-contact records, and (d) a previously verified package stored in the session. Direct matching is applied in each of the first three stages, accepting an account whose $D10$ value

equals $D_{10}(P_{reg})$, together with elimination matching. Let S_{other} denote the set of D10 numbers of every SIM except the registered one, as defined in Eq. (2).

$$S_{other} = \{D_{10}(SIM_i.number) : SIM_i \neq SIM_{reg}\} \quad \text{----- (2)}$$

An account whose number lies in S_{other} belongs to the secondary SIM. It is therefore eliminated, and the other client is chosen for the registered number.

5.3 Two-Phase Accessibility Routine

To attach text to a PDF file, a small state machine controlled by a flag, which records whether the document has already been sent, is run by the service. In the first phase, the node tree of the active window is traversed, the send control on the document preview is located and clicked, and the flag is set. In the second phase, after the app has returned to the conversation, the message input is located, the pending text is entered, and the send control is clicked a second time.

Security is handled in three ways. First, the interactive-window inspection capability is left out of both the static configuration and the runtime settings. Second, the service is restricted to the two WhatsApp package names in both places, so that no events from other applications, such as banking or password-manager apps, are delivered by the operating-system framework. Third, an affirmative disclosure that cannot be skipped is displayed before the user is taken to the system accessibility settings, in accordance with platform policy [5].

5.4 Decision Cascade

After SIM filtering, every event is passed through the queue engine, whose stages are executed in a fixed order, as listed in Table 2. The stages are conjunctive, so no looser preference can override a stricter one.

Table 2: Ordered Decision Stages

No.	Stage	If Not Met
0	Call received on the registered SIM	Drop
1	Master automation switched on	Stop
2	Subscription valid	Stop
3	Number not blacklisted	Stop
4	Unknown-only condition satisfied	Stop
5	Within business hours	Stop
6	Cooldown period elapsed	Stop
7	Rate limit available	Stop
8	Choose SMS, WhatsApp, or both	Stop if none
9	Dispatch and log the actual result	Not applicable

Let M denote the master switch, S the validity of the subscription, B membership in the blacklist, U the unknown-only option, K membership among the saved contacts, H the business-hours condition, C the

expiry of the cooldown, and R the availability of the rate limit. A message is permitted if and only if the condition in Eq. (3) holds, where Allow is the result of Eq. (1).

$$Permit = Allow \wedge M \wedge S \wedge \neg B \wedge (\neg U \vee \neg K) \wedge H \wedge C \wedge R \quad \text{----- (3)}$$

The channel selection then decides which dispatchers are run, namely SMS only, WhatsApp only, both, or none. A blacklisted number is always blocked irrespective of whether it is a saved contact, and the master switch overrides every channel choice.

5.5 Templates and Placeholders

Separate templates are available for missed and completed calls, with independent text for each channel and optional media. The placeholders for caller name, number, and call time are replaced at sending time, so a single template can produce a personalised message, for example a greeting that names the caller and states the time of the call. Document attachments are shared through temporary read-permission Uniform Resource Identifiers (URIs) granted by a file provider.

5.6 Sessions and Licensing

Single-device sessions. A session identifier is stored together with the account. If the same credentials are used on a second handset, the first handset notices that its identifier is no longer current, stops the service and all dispatchers, clears the local session, and returns to the login screen. Duplicate replies and credential sharing are prevented in this way.

Subscription engine. Two plans are offered: a yearly plan, with a start date, an expiry date, and the number of remaining days, and a lifetime plan without an expiry. For yearly plans, the effective expiry is bounded as in Eq. (4), so that a wrongly set stored date cannot stretch a term beyond 364 days. Here T_{store} is the expiry held in the cloud record and T_{start} is the start time, both in milliseconds.

$$T_{exp} = \min(T_{store}, T_{start} + 364 \times 86400000) \quad \text{----- (4)}$$

When the current time reaches T_{exp} , the state changes to expired, the lock overlay is displayed, and the queue engine halts. A cached subscription record and automatic token refresh keep the app usable during short connectivity gaps, and suspensions, reactivations, and plan changes are propagated promptly by the cloud listener.

6 Administrative Control Plane

A prospective customer registers in the User App by providing a name, a phone number, and a plan, which creates a request that is shown to the administrator as pending. The status then moves through contacted and payment received to approved, or ends as rejected. A subscription is assigned and credentials are created upon approval. The administrator is able to review the request details, mark a customer as contacted, suspend or reactivate a subscription, and switch a customer between the yearly and lifetime plans; the change is stored and then picked up by the User App.

For customers who pay offline or do not self-register, a direct add-user form allows the administrator to create an account. Subscriptions belonging to deleted users are filtered out of the list so that stale records are never shown as active. Every figure on the dashboard is derived from stored data, namely pending requests, approved users, total users, yearly and lifetime counts, contacted customers, payments received, and revenue. The values follow the data exactly, including a fall to zero when a collection is emptied, instead of showing cached totals.

7 Technology Stack

The languages, libraries, and platform interfaces that are used are summarised in Tables 3 to 5. The abbreviations appearing in the tables are expanded here: Java Virtual Machine (JVM), Java Development Kit (JDK), Android Gradle Plugin (AGP), Software Development Kit (SDK), and Bill of Materials (BOM).

Table 3: Languages and Build Tooling

Item	Choice	Role
Language	Kotlin 2.1.0	Logic, UI, coroutines
JVM target	JDK 17	Bytecode toolchain
Build	Gradle 8.9 and AGP 8.7.2	Packaging, dependencies
Compiler plugin	Compose 2.1.0	Declarative UI
SDK range	Minimum 24, target 35	Android 7.0 to 15

Table 4: Libraries and Cloud Services

Component	Version	Purpose
Compose BOM and Material 3	2024.11.00	UI toolkit, theming
Navigation Compose	2.8.4	Routing, lock overlay
Lifecycle and ViewModel	2.8.7	State holders
Activity Compose	1.9.3	Permission launchers
Coroutines and Flow	1.9.0	Asynchronous work, state streams
Gson	2.11.0	Local JSON storage
Firebase BOM	33.7.0	Authentication, Firestore, Storage, Analytics

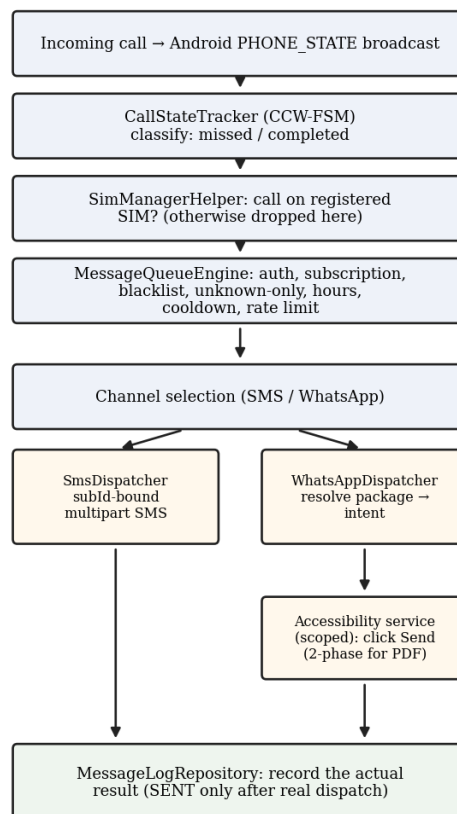
Table 5: Platform Interfaces

Interface	Function
TelephonyManager	Call-state events
SubscriptionManager	SIM slots, numbers, subscription identifiers
SmsManager	Subscription-bound multipart SMS
AccessibilityService	Scoped UI automation for WhatsApp
AccountManager	Numbers tied to WhatsApp clients
CallLog and ContactsContract	Caller lookup, saved-contact check
PowerManager and KeyguardManager	Waking the device to run intents
Foreground service	Persistent background execution
AlarmManager and boot receiver	Self-healing restarts
FileProvider	Temporary media URI grants

8 End-to-End Pipeline

The processing of a single event is traced in Figure 3. A phone-state change is broadcast by the platform, the call is classified by the tracker, calls on unregistered lines are dropped by the SIM helper, and the cascade is applied by the queue engine. Events that survive are routed to the SMS dispatcher, the WhatsApp dispatcher, or both. On the WhatsApp path, the package is resolved, the share intent is started, and the scoped accessibility service is relied upon to press the send control, after which a result is reported by the service. Finally, the actual outcome is recorded by the repository, and a message is marked as sent only after real dispatch, and not merely when an app has been opened or a draft has been prepared.

Figure 3: End-to-End Processing Pipeline



9 Analysis and Discussion

9.1 Data Placement and Privacy

Table 6 separates the data kept on the handset from the data held in the cloud. Call-log reads, contact lookups, templates, queues, cooldowns, and message histories are all processed locally, while only the information required for licensing is held in the cloud. This reflects the principle that the content of communications should not leave the device when a feature does not need it.

Table 6: Data Placement

Data Item	Location
Call log and caller numbers	Device only
Contact lookups and blacklist	Device only
Templates, queue, and cooldowns	Device only
Message and call history	Device only
Cached subscription	Device (mirror)
Identity and tokens	Cloud authentication
Subscription record	Cloud store
Registration requests	Cloud store
Active session identifier	Cloud store

9.2 Scenario Walk-Throughs

As the contribution is architectural in nature, the behaviour is verified by walking scenarios through Eqs. (1) to (4) and the state machine, as presented in Table 7.

Table 7: Scenarios and Prescribed Outcomes

Scenario	Prescribed Outcome
Call on the personal SIM	Dropped, no reply
A is active, B rings and then stops	MISSED reply for B is sent at once, COMPLETED reply for A is sent later
Blacklisted number, unknown-only off	No message
Saved contact, unknown-only on	No message
Outside business hours	No message
Both WhatsApp clients installed	Client of the registered account is chosen
PDF with a caption	Two-phase send: document first, then text
Subscription suspended	Lock overlay shown, queue halted
Login on a second device	First device stops and signs out
Collection emptied by the administrator	Dashboard shows zero

9.3 Capability Comparison

Table 8 compares the capabilities with two generic categories of existing tools. The comparison is based on an analysis of the design of those tools by the authors and not on a benchmark of named products. The word Partial marks behaviour that exists but with limitations.

Table 8: Qualitative Capability Comparison

Capability	Generic Apps	Auto-Reply	Cloud Telephony Apps	Proposed System
Incoming SIM filtering	Typically no		Not applicable	Yes
SIM-bound SMS route	Partial		No	Yes
Dual WhatsApp resolution	Typically no		Needs a paid API	Yes
Call-waiting handling	Typically no		Partial	Yes
PDF with a caption	Typically no		Through a paid API	Yes
Scoped accessibility	Varies		Not used	Yes
Call-data privacy	Varies		Cloud upload	On-device

9.4 Limitations and Threats to Validity

Permitted call-state access, background execution, and messaging behaviour differ between Android releases, and manufacturer battery policies may suspend services even when they run in the foreground. The WhatsApp path depends on interface identifiers of a third-party app, which may change and break the accessibility routine. The matching of numbers by their last ten digits assumes that the registered number and the SIM records are reported consistently, and since some carriers do not expose the number of a SIM, elimination matching is weakened. Missed and rejected calls cannot always be told apart. In addition, automated messaging must comply with the terms of messaging platforms and with local communication and data-protection rules. Finally, no field measurements are reported in this paper, so latency, delivery rate, battery cost, and behaviour under large user populations remain unquantified.

9.5 Future Work

A controlled field study across device models and Android versions is planned, along with lightweight on-device language understanding for intent-based follow-ups, dynamic handling of embedded SIM (eSIM) profiles, integration of Voice over Internet Protocol (VoIP) call states, and an official business-messaging integration that would reduce the dependence on interface automation.

10 Conclusion

An Android system that turns call events into context-aware SMS and WhatsApp replies has been presented. The design suggests that dependable auto-response on modern Android depends less on the act of sending a message than on correct decisions about which line, which client, and which caller a message belongs to. By tying every decision to the registered number, evaluating the rules in a fixed order, and recording only real outcomes, the design avoids replies from the wrong SIM, through the wrong WhatsApp account, or to the wrong caller. Keeping private communication data on the handset while licences and sessions are controlled from the cloud indicates that user privacy and commercial control need not conflict. Empirical evaluation on real devices is the natural next step, as outlined in the limitations and future work.

References

1. Google, “TelephonyManager”, Android Developers API Reference. <https://developer.android.com/reference/android/telephony/TelephonyManager>

2. Google, “SubscriptionManager”, Android Developers API Reference. <https://developer.android.com/reference/android/telephony/SubscriptionManager>
3. Google, “SmsManager”, Android Developers API Reference. <https://developer.android.com/reference/android/telephony/SmsManager>
4. Google, “AccessibilityService”, Android Developers API Reference. <https://developer.android.com/reference/android/accessibilityservice/AccessibilityService>
5. Google, “Use of the AccessibilityService API”, Google Play Console Help. <https://support.google.com/googleplay/android-developer/answer/10964491>
6. Robert C.M., “Clean Architecture: A Craftsman’s Guide to Software Structure and Design”, Prentice Hall, 2017.
7. Google, “Guide to App Architecture”, Android Developers. <https://developer.android.com/topic/architecture>
8. Gregor H., Bobby W., “Enterprise Integration Patterns”, Addison-Wesley, 2003.
9. Ravi S.S., Edward J.C., Hal L.F., Charles E.Y., “Role-Based Access Control Models”, Computer, February 1996, 29 (2), 38–47.
10. Google, “Firebase Authentication”, Firebase Documentation. <https://firebase.google.com/docs/auth>
11. Google, “Cloud Firestore”, Firebase Documentation. <https://firebase.google.com/docs/firestore>