Software Design Specification

AI Inbound Calling Agent

Internal Advisor:

Dr. Muhammad Saad Razzaq

Project Manager:

Dr. Muhammad Ilyas

Project Team:

Muhammad Asmaad Saeed	BSCS51F22S037	(TL)
Laiba Siraj	BSCS51F22S046	
Fatima Zahid	BSCS51F22S010	
Areeba Rehman	BSCS51F22S039	

Submission Date:

December 15, 2025

Project Manager's Signature

Document Information

Category	Information
Customer	UOS – Department of Computer Science
Project	AI Inbound Calling Agent
Document	Software Design Specification
Document Version	1.0
Identifier	
Status	Draft
Author(s)	Muhammad Asmaad Saeed, Laiba Siraj, Fatima Zahid, Areeba Rehman
Approver(s)	PM
Issue Date	Sept. 15, 2025
Document Location	
Distribution	1. Advisor 2. PM

Definition of Terms, Acronyms and Abbreviations.

Term	Description
ASR	Automatic Speech Recognition
NLU	Natural Language Understanding
TTS	Text-to-Speech
KG	Knowledge Graph
GraphRAG	Graph-based Retrieval-Augmented Generation
OTP	One-Time Password
VoIP	Voice over Internet Protocol.
PTSN	Public Switched Telephone Network
TLS	Transport Layer Security
SRTP	Secure Real-Time Transport Protocol
GDPR	General Data Protection Regulation
PDPL	Pakistan Personal Data Protection Law
HIPAA	Health Insurance Portability and Accountability Act

Table of Contents

1. Introduction 4

1.1 Purpose of Document 4
1.2 Project Overview 4
1.3 Scope 4

2. Design Considerations 4

2.1 Assumptions and Dependencies 5
2.2 Risks and Volatile Areas 6

3. System Architecture 8

3.1 System Level Architecture 8
3.2 Sub-System / Component / Module Level Architecture 11
3.3 Sub-Component / Sub-Module Level Architecture 12

4. Design Strategies 116

5. Detailed System Design 118

6. References 19

1. Introduction

1.1 Purpose of Document

This Software Design Specification (SDD) defines the architectural, subsystem, and component-level design of the AI Inbound Calling Agent developed for the University of Sargodha (UOS). It serves as a blueprint for developers, system architects, testers, and evaluators to understand the system's internal structure, module interactions, data flows, and design strategies.
The project follows a modular, component-based, object-oriented design methodology

1.2 Project Overview

The AI Inbound Calling Agent is an intelligent, Urdu-speaking virtual assistant designed to automate inbound customer support for the University of Sargodha (UOS). It integrates Automatic Speech Recognition (ASR), Natural Language Understanding (NLU), and Text-to-Speech (TTS) technologies to manage real-time calls, understand natural language queries, and deliver accurate, context-aware responses in both Urdu, English and hybrid of both languages.

1.3 Scope

The system's included functionalities will be:

i. Automatically answers inbound calls and interacts with callers in real time.
ii. Uses speech recognition and Natural Language Understanding (NLU) to identify intent and provide accurate responses.
iii. Supports bilingual communication in both English and Urdu and hybrid of two.
iv. Retrieves essential information from the university's database, including admission schedules, departmental contacts, office timings, and event details and others.
v. Allows seamless call transfer to a human operator for complex queries.
vi. Provides an analytics dashboard to monitor call volume, response accuracy, and performance based on system logs and user feedback.
vii. Includes basic urgency detection to improve response quality and user experience.

The system's excluded functionalities will be:

I. Outbound or promotional calling.
II. Video or chat-based communication.
III. Integration with social media or non-telephony communication platforms.
IV. External Support for non-verbal communication modes (e.g., emails, or text chat).

2. Design Considerations

The design of the AI Inbound Calling Agent for university admissions must account for diverse caller backgrounds, heavy call traffic during admission cycles, and Pakistan's inconsistent telecommunication conditions. Since users may speak in Urdu, English, or a code-switched mix, the system requires accurate, low-latency ASR and reliable intent detection under noise and variable speech patterns. It must operate stably despite network fluctuations, ensure secure real-time integration with university databases, and comply with relevant data protection laws. Clear fallback mechanisms and escalation to human operators are necessary for complex or sensitive queries. These considerations directly shape the system’s architecture, model selection, telephony pipeline, and error-handling strategies to deliver a robust and accessible admission support solution.

2.1 Assumptions and Dependencies

2.1.1 Assumptions

1. **Stable Telephony and Network Connectivity**
   VoIP/PSTN gateways are expected to provide stable audio streams with minimal jitter, and continuous internet connectivity is assumed for real-time ASR, NLU, and TTS.

2. **Cloud ASR and TTS Services Remain Available and Reliable**
   External speech services (e.g., OpenAI/Google) are assumed to remain available and perform consistently for Urdu, English, and mixed-language queries.

3. **Callers Frequently Use Bilingual Code-Switching**
   Callers are expected to frequently switch between Urdu and English, influencing ASR, NLU, and TTS configuration.

4. **Structured & Updated University Data**
   The Neo4j knowledge graph is assumed to receive regularly updated, well-structured datasets for admissions, departments, and schedules.

5. **System Will Operate in a Cloud or Hybrid Deployment**
   The system is assumed to run in a cloud or hybrid setup capable of handling ASR/TTS workloads and concurrent call sessions.

6. **Session-Based Conversation Handling**
   Each call is assumed to maintain a separate session with temporary context for multi-turn dialogue without long-term memory.

2.1.2 Dependencies

1. **ASR, NLU, and TTS Engines**
   System performance depends on cloud-based AI components. An API failure or degradation directly impacts the call flow and user experience.

2. **Telephony Provider (VoIP/PSTN)**
   The system depends on SIP trunks or telephony gateways for receiving calls, routing sessions, and streaming audio.

3. **Backend Services and Flask API**
   All modules communicate through the central orchestrator. The entire pipeline relies on Flask for routing, session management, and middleware operations.

4. **Message Queue (RabbitMQ/Kafka)**
   Reliable streaming of audio chunks and non-blocking ASR/NLU/TTS pipelines depends on a stable message queuing service.

5. **Neo4j/GraphRAG Knowledge Base**
   Contextual and accurate answers depend on the proper functioning, indexing, and updating of the knowledge graph.

6. **System Logs and Storage Services**
   Logging, analytics, and call summaries rely on secure storage services with suitable retention policies.

2.2 Risks and Volatile Areas

This subsection identifies the major sources of uncertainty which may affect the system's design or long-term maintainability.

2.2.1 Technology Risks

1. **Variations in Urdu/English Speech Patterns**
   ASR accuracy may drop due to regional accents, background noise, or rapid code-switching. This may introduce misinterpretations or incomplete responses.

2. **Cloud API Changes (Pricing or Rate Limits)**
   AI service providers may change pricing, quotas, or model availability, which could influence cost and architectural choices.

3. **Telephony Integration Issues**
   Differences in SIP protocols, codecs, or network configurations may require design adjustments in the call-handling layer.

2.2.2 Requirements and Data Volatility

1. **Rapidly Changing University Information**
   Admission dates, fee structures, and department updates frequently change. Designing for regular updates is necessary to avoid outdated responses.

2. **Expanding Query Categories**
   Over time, more departments may need to be added, increasing the complexity of intent classification and knowledge graph growth.

3. **Operational Policies May Evolve**
   Changes such as new call escalation rules or call-duration limits can affect the design of call flow and routing logic.

2.2.3 Security and Privacy Risks

1. **Handling Caller Information**
   Even though the system avoids storing sensitive data, logs still pose risks if not encrypted or anonymized.

2. **Compliance Requirements**
   Regulations (e.g., GDPR/PDPL) may evolve, requiring changes to data handling, retention, and anonymization strategies.

2.2.4 Performance Risks

1. **High Call Volume Peaks**
   During peak periods (admissions), concurrent call demand may exceed model or server capacity.

2. **Pipeline Latency**
   ASR → NLU → TTS may introduce noticeable delays if models or network conditions degrade.

3. **Knowledge Graph Query Delays**
   Slow or inefficient graph queries can cause response lag, especially under load.

2.3 Constraints Influencing the Design

1. **Real-Time Responsiveness**
   The system must deliver responses within approximately 2 seconds, which limits the use of heavy models or complex inference pipelines.

2. **Inbound Voice-Only Constraint**
   The system is restricted to voice-based communication. No chat, email, AI chat bot, or outreach module is included in the design.

3. **Compliance and Legal Constraints**
   The system must follow GDPR/PDPL guidelines, avoiding storage of personal identifiers or raw audio where unnecessary.

4. **Hardware and Bandwidth Limitations**
   Real-time ASR/TTS performance is sensitive to CPU, memory, and especially network bandwidth.

5. **Language Constraints**
   The design must handle Urdu-English mixed speech without requiring separate pipelines.

6. **No Proprietary Telephony Hardware**
   The solution must be compatible with generic VoIP systems, limiting the use of specialized hardware optimizations.

3. System Architecture

3.1 System Level Architecture

At the highest level, the AI Inbound Calling Agent is divided into three major layers:

A. Telephony and Audio Interface Layer

Handles inbound call reception, audio streaming, SIP/VoIP communication, session setup, and initial routing.

B. AI Processing Layer (Core Intelligence)

Processes spoken input and generated responses. This includes ASR, NLU, intent classification, Neo4j/GraphRAG retrieval, response generation, and TTS.

C. Backend Services & Management Layer

Provides APIs, logging, analytics, error handling, configuration, and admin dashboard functions

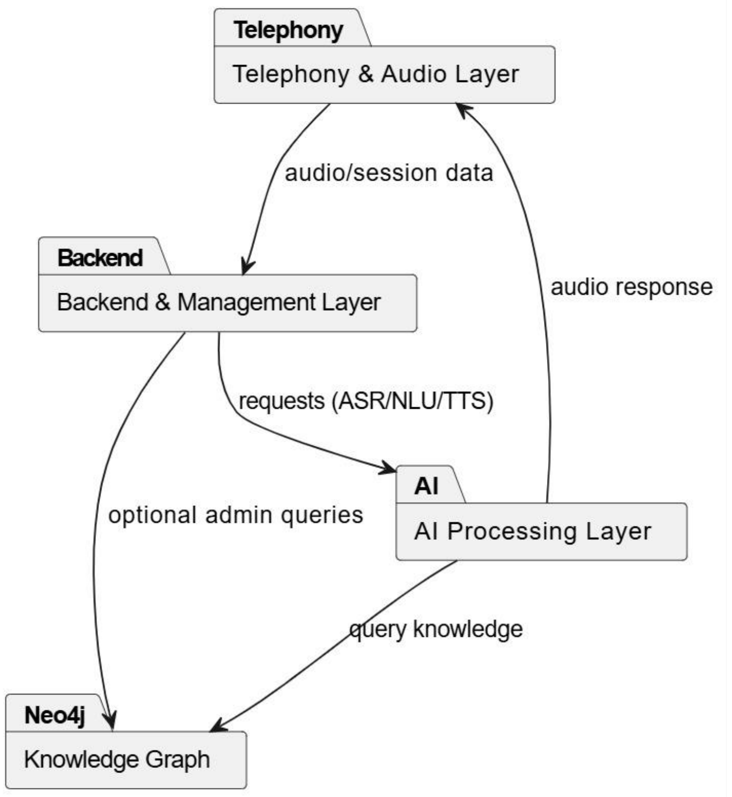


Figure 3.1: System Architecture Diagram

.

3.1.1 System Decomposition into Elements

System Element	Role
Telephony Gateway (VoIP/PSTN)	Receives inbound calls; streams audio to backend
Audio Session Manager	Manages per-caller interactions, buffering, session state
ASR Module	Converts caller speech (Urdu/English/code-switching) to text
NLU & Intent Engine	Identifies user intent, entities, and query type
Knowledge Retrieval (GraphRAG + Neo4j)	Fetches contextual, structured UOS information
Response Generator	Formulates natural language reply
TTS Module	Converts textual response into natural Urdu/English speech
Flask Backend / API Gateway	Orchestrates all modules and handles communication
Logs & Monitoring	Stores call summaries, timestamps, error reports
Admin Dashboard	Provides analytics to university staff

3.1.2 Relationships Between the Elements

1. Telephony receives the call → sends audio stream to Backend
2. Backend forwards audio chunks → ASR
3. ASR text → NLU
4. NLU → Intent classification
5. Intent → GraphRAG/Neo4j (only if data retrieval required)
6. Retrieved info → Response generator
7. Response → TTS → Audio output
8. Audio → Sent back to caller through Telephony Gateway
9. The flow is strictly session-based to maintain conversation continuity.

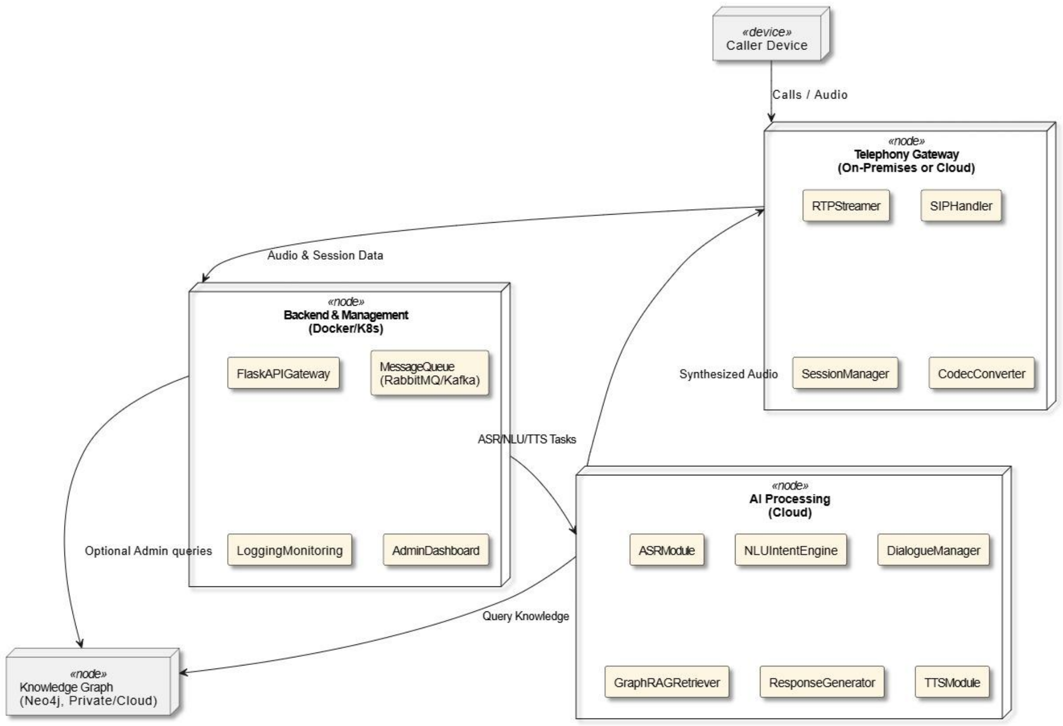


Figure 3.1.2: Deployment diagram

3.1.3 Interfaces to External Systems

External System	Interaction
VoIP/PSTN Telephony Gateway	Audio in/out, SIP signaling
OpenAI/Cloud ASR & TTS	Speech recognition & synthesis APIs
University Database / Neo4j	Graph-based information retrieval
Admin Dashboard (Web)	Call analytics, logs, monitoring

3.1.4 Major Physical Design / Deployment Considerations

1. Telephony gateway may run on-premises or cloud.
2. AI processing will run in cloud for lower latency and stable compute.
3. Knowledge graph (Neo4j) may be hosted on private server (UOS) or cloud.
4. Backend (Flask) runs in a containerized environment (Docker/K8s) for scalability.
5. Message queue (RabbitMQ/Kafka) facilitates parallel ASR/NLU/TTS pipelines.

3.1.5 Global Design Strategies

1. **Error Handling:**
- Retry logic for dropped ASR/TTS API calls.
- Fail gracefully (e.g., “Sorry, please repeat that.”).
- Default fallback responses for unknown intents.
2. **Security:**
- All communication over TLS/SRTP.
- Logs anonymized; no sensitive caller data stored.
3. **Maintainability:**
- Highly modular architecture allows replacing ASR/TTS engines.
- GraphRAG layer allows incremental knowledge updates without retraining models.

3.2 Sub-System / Component / Module Level Architecture.

3.3 Sub-Component / Sub-Module Level Architecture (1…n)

3.3.1 ASR Sub-Module Architecture

Sub-components:

- Audio Chunk Handler
- Pre-processing (noise reduction, silence trim)
- ASR API Connector
- ASR Result Normalizer
- Error & Retry Handler

Interactions:

- Streams chunks → sends to ASR API
- Receives partial/transcript → normalizes → sends to NLU

3.3.2 NLU Sub-Module Architecture

Sub-components:

- Tokenizer
- Intent Classifier
- Entity Extractor

- Context Manager
- Fallback Detector

Interactions:

- Receives text → identifies intent → routes to Dialogue Manager
- Handles ambiguous utterances
- Maintains per-call context (short-term memory)

3.3.3 Knowledge Retrieval (GraphRAG) Sub-Modules

Sub-components:

- Query Builder
- Graph Traversal Engine
- Relevance Ranking
- Data Formatter

Interactions:

- Builds query based on intent/entities
- Searches Neo4j
- Returns structured information to Response Generator

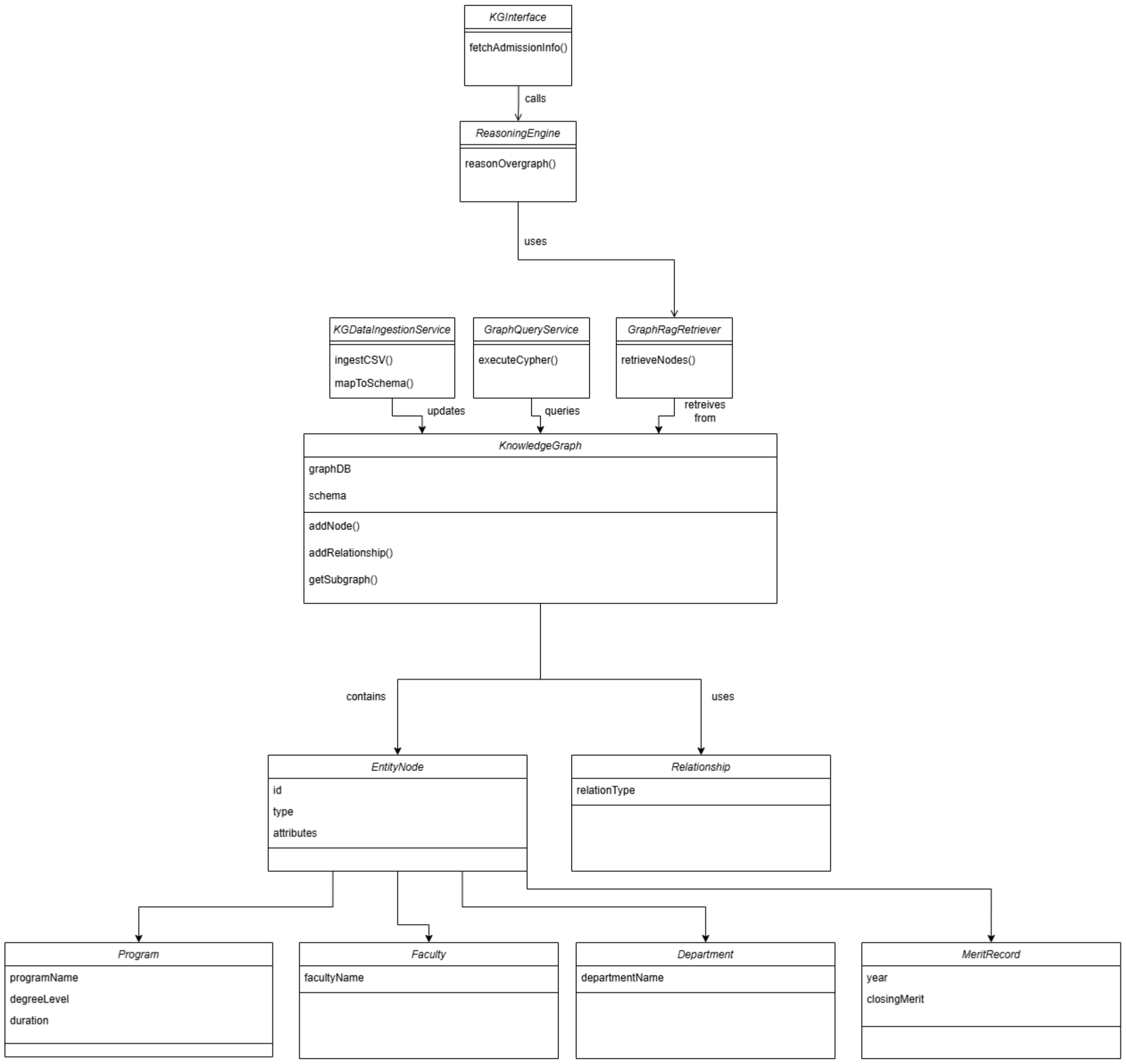


Figure 3.3.3: KG Class Diagram

Figure 3.3.3 illustrates the class diagram of the Knowledge Graph and Reasoning subsystem. The diagram highlights the internal structure of the subsystem, including entity representations, graph management, reasoning mechanisms, and the external interface used by other system components.

3.3.4 TTS Sub-Module Architecture

Sub-components:

- Text Normalization
- Language Decision (Urdu/English mix)
- TTS API Connector

- Audio Chunk Generator
- Audio Stream Router

3.3.5 Sequence Diagram

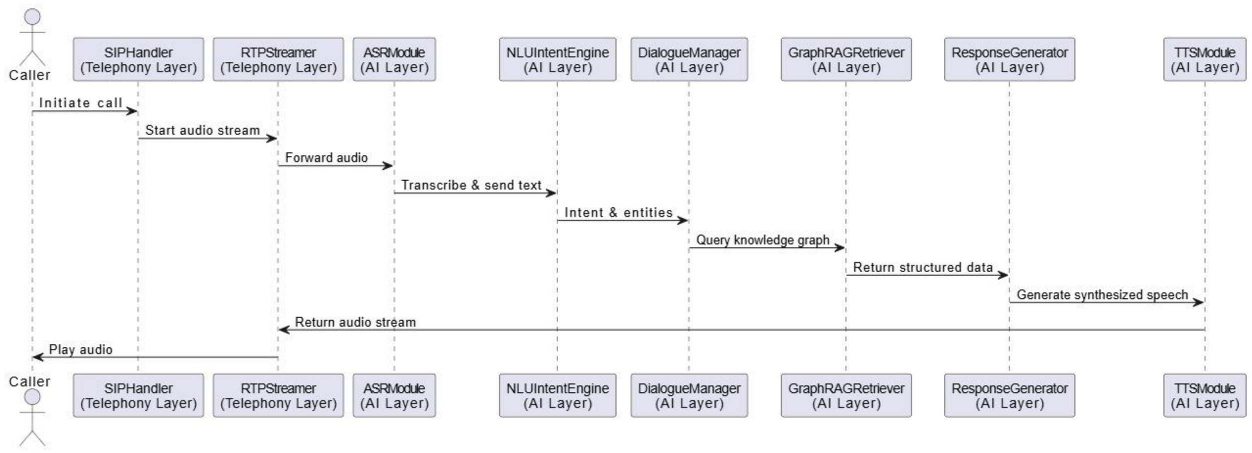


Figure 3.3.5: Sequence diagram

3.3.6 Activity Diagram

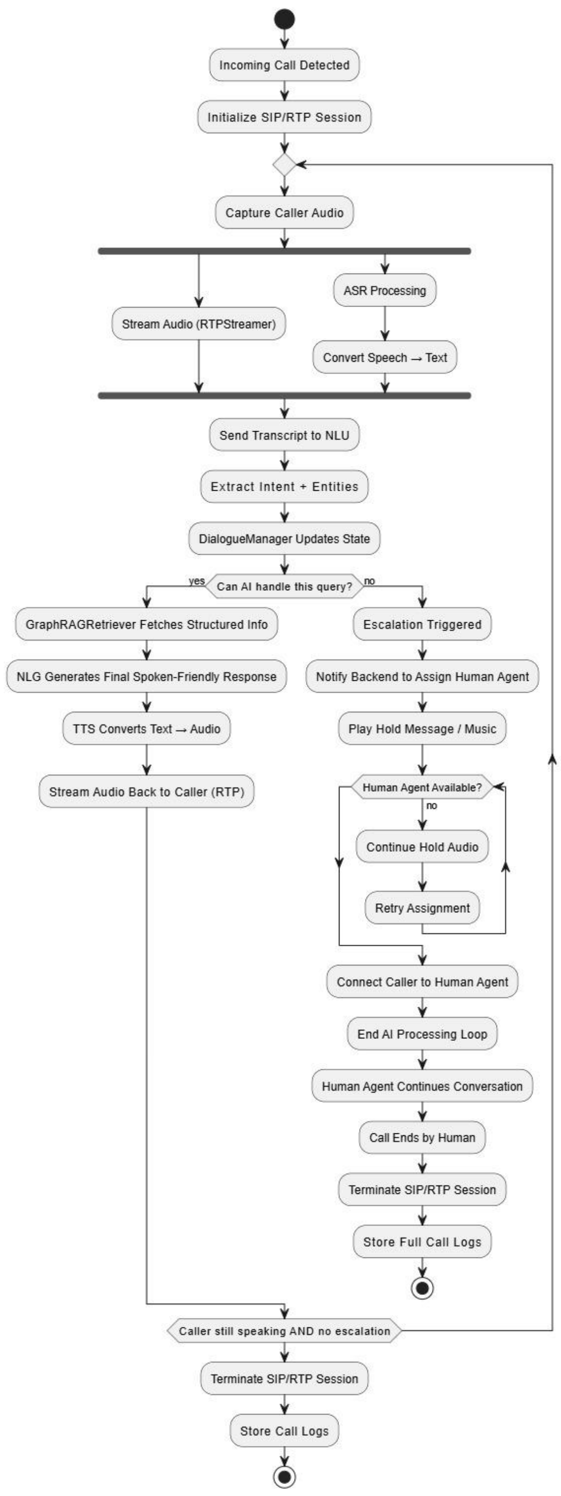

Incoming Call Detected
Initialize SIP/RTP Session
Capture Caller Audio
Stream Audio (RTPStreamer)
ASR Processing
Convert Speech → Text
Send Transcript to NLU
Extract Intent + Entities
DialogueManager Updates State
Can AI handle this query?
yes
no
GraphRAGRetriever Fetches Structured Info
NLG Generates Final Spoken-Friendly Response
TTS Converts Text → Audio
Stream Audio Back to Caller (RTP)
Escalation Triggered
Notify Backend to Assign Human Agent
Play Hold Message / Music
Human Agent Available?
no
Continue Hold Audio
Retry Assignment
Connect Caller to Human Agent
End AI Processing Loop
Human Agent Continues Conversation
Call Ends by Human
Terminate SIP/RTP Session
Store Full Call Logs
Caller still speaking AND no escalation
Terminate SIP/RTP Session
Store Call Logs

Figure 3.3.6: Activity diagram

4. Design Strategies

Strategy 1: Modular and Component-Based Architecture

Description

The system is organized into a set of loosely connected, independently deployable components, including ASR, NLU, Dialogue Manager, Knowledge Graph, TTS, Logging, and the Telephony Gateway. Each module has a clear function and communicates with others through well-structured APIs or asynchronous message queues.

Reasoning

- This setup ensures a clear separation of tasks, making each component easy to maintain and test.
- It supports independent development and replacement of AI/ML modules without disrupting other parts of the system.
- It allows parallel development, letting multiple team members work on different modules at the same time.
- It improves system resilience by isolating failures to individual services.

Trade-offs

- It increases architectural complexity because of inter-service communication and orchestration.
- It requires careful API versioning, monitoring, and deployment processes.
- It introduces minor overhead due to network communication between components.

Design Considerations

- **Future Extension:** New features, like hostel queries and exam services, can be added by including additional modules without redesigning the architecture.
- **System Reuse:** Components like ASR, NLU, and the Dialogue Manager can be reused for campus chatbots or web-based assistants.
- **User Interface Paradigm:** This design allows for a consistent conversational flow, even when backend components are upgraded or modified.
- **Data Management:** Each module minimally manages its own state; long-term data is stored in dedicated databases to maintain persistence and consistency.
- **Concurrency & Synchronization:** Independent services allow for handling multiple calls at once. Synchronized session states are maintained using session IDs and lightweight caching layers, like Redis.

Strategy 2 - Knowledge-Centric Reasoning (Knowledge Graph + GraphRAG)

Description

The system uses a combination of knowledge-centric reasoning. It employs a Knowledge Graph (KG) to store organized academic and administrative data. It also uses GraphRAG to find and provide context for information relevant to the language model. This method ensures factual accuracy, understanding of relationships, and contextual reasoning concerning admission-related questions.

Reasoning

- UOS data, such as programs, departments, merit lists, and admissions rules, has a relational nature, making a graph representation suitable.
- The KG provides precise, verifiable, and reliable information, which is essential in academic settings.
- GraphRAG allows the AI agent to navigate relevant nodes and gather important context before generating responses. This process improves accuracy for complex or interconnected queries.
- It enhances explainability by showing a clear path of which nodes, facts, or relationships influenced the final answer.

Trade-offs

- Initially building and maintaining the KG requires manual effort and regular updates.
- Combining different retrieval methods (KG, vector search, LLM) adds complexity to the architecture.
- It requires specific ETL processes to keep university data current.

Design Considerations

- **Future Extension:** The KG can be gradually expanded to include details about examinations, fee structures, scholarships, hostel information, and faculty data.
- **System Reuse:** The KG and retrieval methods can be reused in other UOS systems like web portals, student chatbots, and administrative dashboards.

- **User Interface Paradigm:** It supports natural language interactions through highly accurate backend reasoning, ensuring consistent and reliable voice responses.
- **Data Management:**

1. The KG provides organized and stable storage for relational data.
2. GraphRAG uses embeddings and semantic retrieval to handle flexible natural language queries.
3. Both methods ensure an auditable and maintainable data path.

- **Concurrency & Synchronization:**

1. KG queries run quickly and can handle multiple calls at the same time.
2. Retrieval methods function asynchronously with ASR and NLU modules, providing quick responses without interfering with other processes.

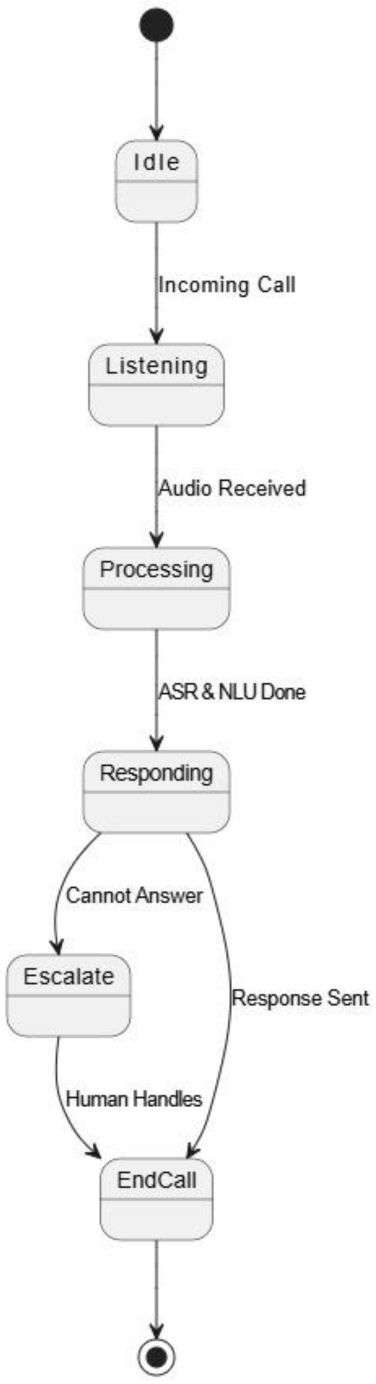
5. Detailed System Design

5.1 Class diagram

5.2 ERD

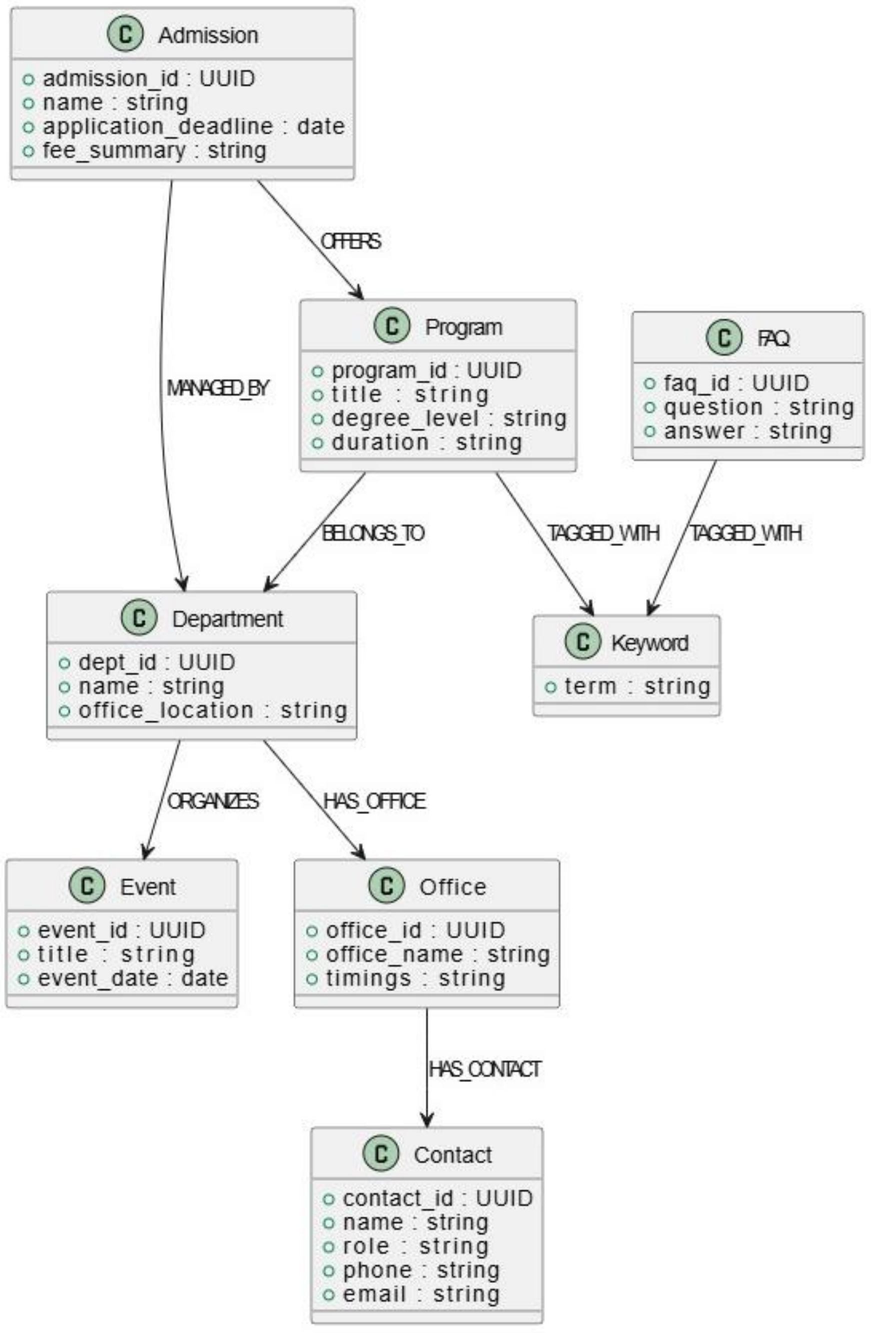
5.3 State Transition Diagram

Figure 5.3: State Transition Diagram



5.4 Logical Data Model

Figure 5.4: Logical data model



5.5 Physical Data Model

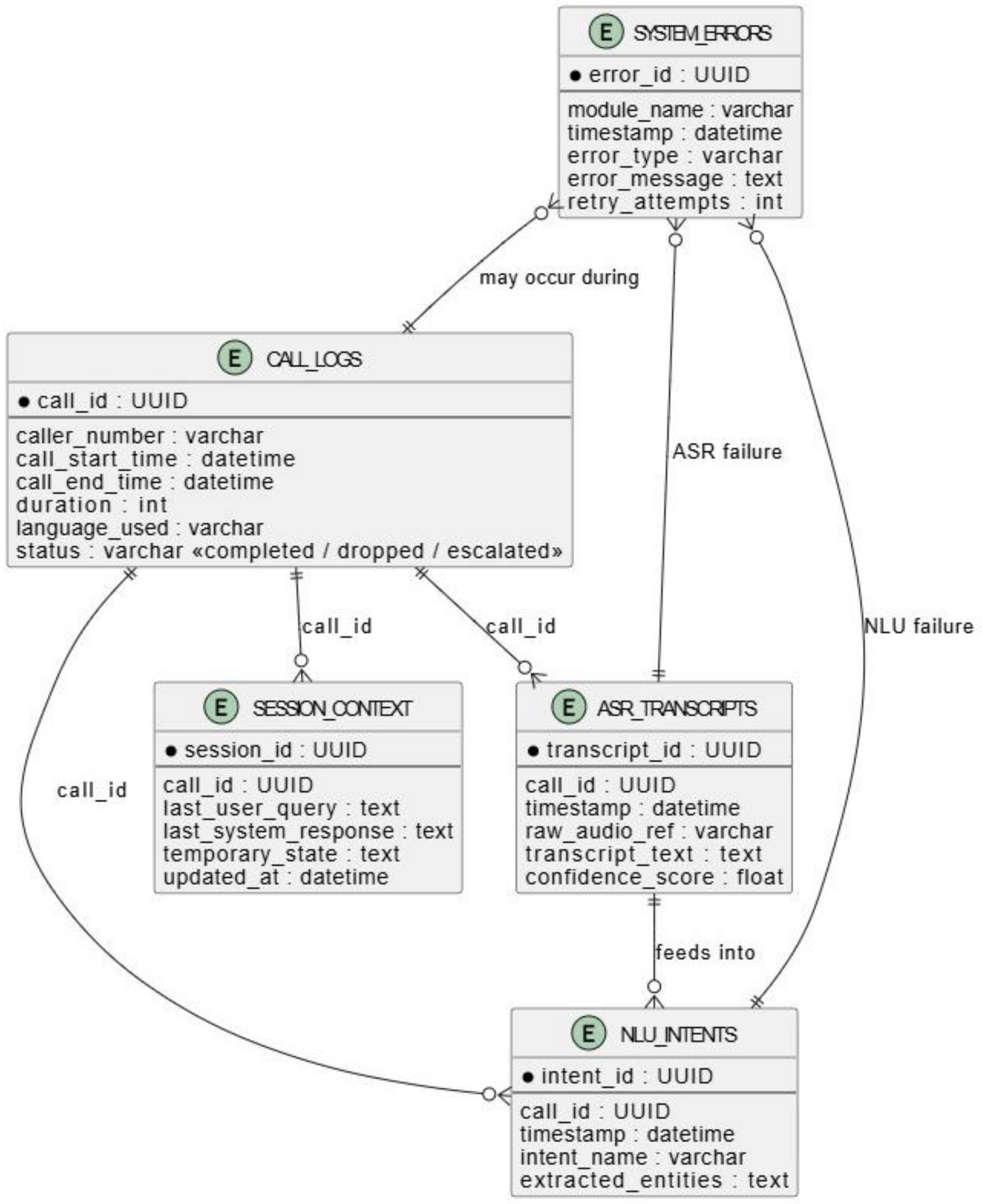


Figure 5.5: Physical data model

6. References

Ref. No.	Document Title	Date of Release/ Publication	Document Source
PGBH01-2025-Proposal	Project Proposal	Sept 23, 2025	https://github.com/AI-INBOUND-CALLING-AGENT/Proposal.pdf
PGBH01-2025-FS	Functional Specification	Oct 11 , 2025	https://github.com/AI-Inbound Calling Agent/SRS.pdf