DEVELOPMENT DOCUMENTATION

for

Relevium

Version 0.7

Prepared by
Abdelrahman Solyman - Ahmed Samir
Mahmoud Khaled - Mohamed ELTorky
Mohamed Fathalla - Omar Mohamed

Supervised by
Dr. Mohamed Magdy

College of Computing & Information Technology

January 17, 2019

Contents

List of Figures

Revision History

Name	Date	Reason For Changes	Version
sample.pdf	September 23, 2018	--	v0.1
spec.pdf	November 27, 2018	Agent Specifications	v0.2
illustrations.pdf	December 23, 2018	Illustrations Draft	v0.3
appendix.pdf	January 10, 2019	Populating Appendix	v0.4
revision.pdf	January 12, 2019	Diagrams Revision	v0.5
mocks.pdf	January 13, 2019	Design Mocks	v0.6
review.pdf	January 16, 2019	Comprehensive Review	v0.7

1 Introduction

1.1 Purpose

The purpose of this document is to give a detailed description of the requirements for the "Relevium" software. It will illustrate the purpose and complete declaration for the development of system. It will also explain system constraints, interface and interactions with other external applications.

1.2 Document Conventions

- Crow notation is used to describe Entity-Relationship diagrams.

1.3 Intended Audience and Reading Suggestions

This document is primarily intended to be proposed to Dr. Mohamed Magdy for his approval and a reference for developing the first version of the system for the development team.

1.4 Project Scope

A natural disaster is a sudden event that causes widespread destruction, lots of collateral damage or loss of life, brought about by forces other than the acts of human beings.

The project aims to develop mobile application that enhance the communication between people who are in danger using chatting groups, and managing disaster by recommending an evacuation plan in addition supporting virtual assistant based on NLP.

1.5 Used Technologies

- *Android Studio* as our **IDE** for developing the application prototype.
- Design of project diagrams we used *Lucidchart* and *StarUML*.
- Documentation & Presentation are written in LaTeX document preparation system.

1.6 Prototype

1.6.1 Expected Design

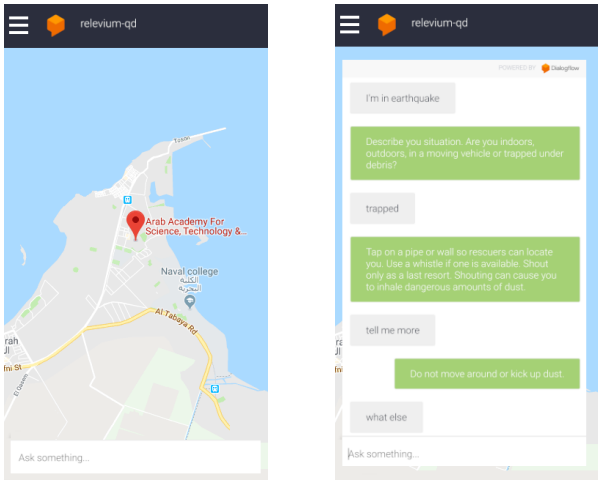

Figure 1.1: Expected Relevium Home Screen

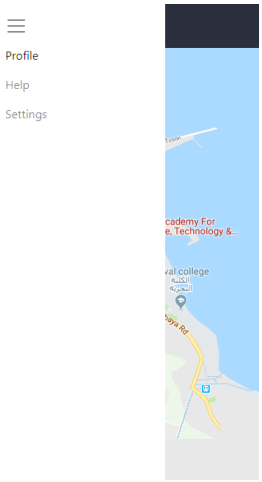

Figure 1.2: Expected Application Feature List

1.6.2 Actual Prototype

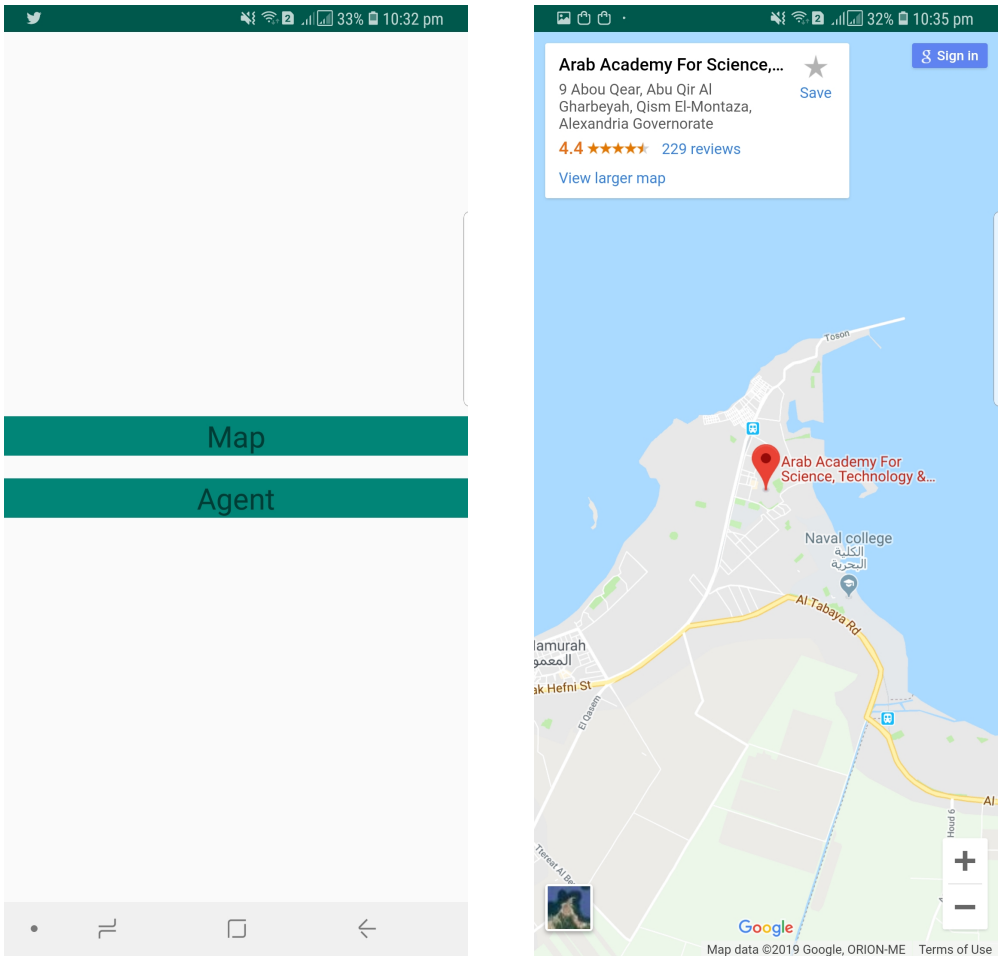

Figure 1.3: Actual Relevium Home Screen & Map

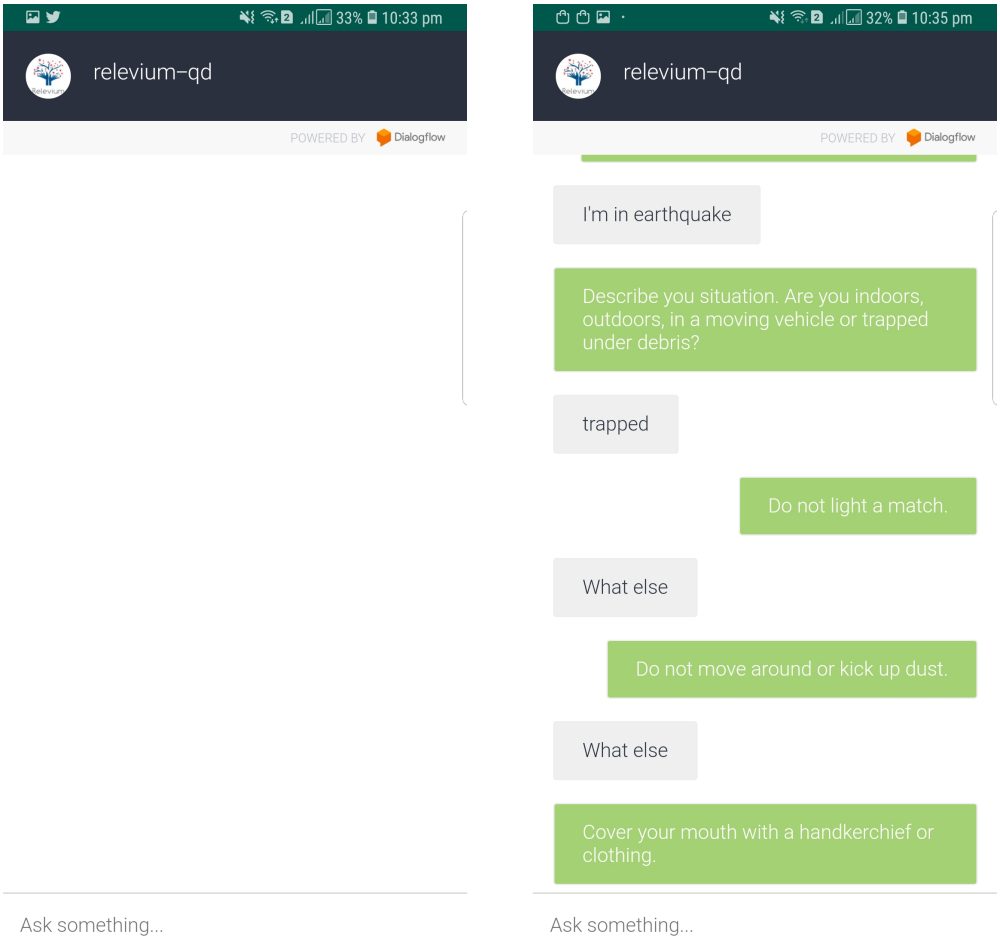

Figure 1.4: Actual Relevium Agent

2 System Architecture

2.1 Product Perspective

This application is designed to enhance the communication between people who are in danger using chatting groups, and managing disaster by recommending an evacuation plan in addition supporting virtual assistant based on NLP.

2.2 Product Function

Relevium is a mobile application with built in application assistant which leads to smoother and easier user experience. The application mainly focus on the user's safety through collecting real time data only in the disastrous situations eg: (Hurricanes, earthquakes, etc) to help the user to safely escape or ping his/her location to notify the authorities for rescue, or it can be used to access list of the nearby shelters and potential danger zones to be away from. The app also can be used as messaging application with the nearby users during the disaster to request nearby help or even a pickup if your car got jammed.

2.3 Operating Environment

This application is specifically designed for Android and iOS. There needs to be a GPS based system for the application to access. The interface will be made to have a similar look and feel that is consistent with other Android applications. Most Android applications have similar way to display and navigate through data.

2.4 Design and Implementation Constraints

2.4.1 Solution Constraints

Due to the choice of technology, some constraints have arisen mainly in the web crawling section. To start, a crawler must wait between repeated accesses to the same website. Otherwise, the crawler can be blocked. In addition, duplicate content can generate a waste of valuable resources.

2.4.2 Memory & Space Constraints

Our application run on mobile devices with very limited hardware specs which may lead to overheating or battery drain, above all that it will be limited to small amount of ram and storage space on some devices.

2.4.3 Schedule Constraints

The project has a relatively small time-frame. Therefore, the time is a major concern. The project should be functioning and completed by the mid of 2019.

2.4.4 Policy & Privacy Constrains

The application tries to use all the available mobile resource in disastrous times to save all the user in potential danger eg: using the phone while it's on standby mode to alert the user or use mobile flash to help users who are stuck under derbies. Although we do our best to save the user life, this does not mean we will prevent their death 100%, which is must be stated to clear us from any charges.

2.5 Assumptions and Dependencies

One assumption about the product is that it will always be used on mobile phones that have enough performance. If the phone does not have enough hardware resources available for the application, for example the users might have allocated them with other applications, there may be scenarios where the application does not work as intended or even at all.

Another assumption is that the GPS components in all phones work in the same way. If the phones have different interfaces to the GPS, the application need to be specifically adjusted to each interface and that would mean the integration with the GPS would have different requirements than what is stated in this specification.

2.6 Related work

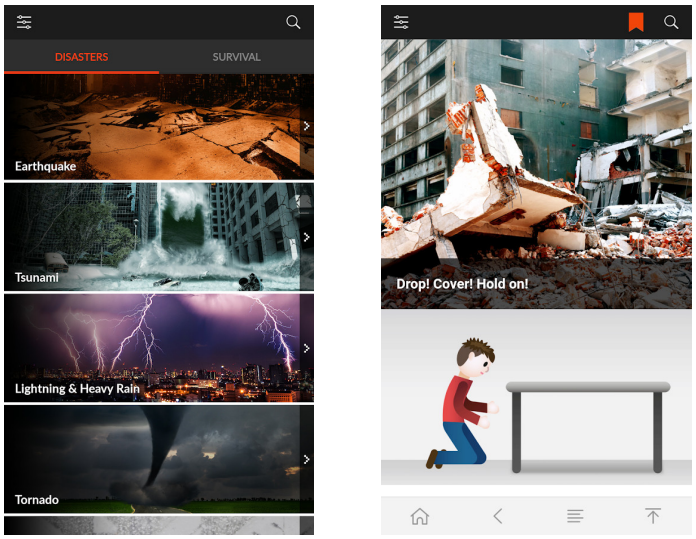

Figure 2.1: Emergency Preparedness & Disaster Survival Guide is paid mobile application that cover almost all the how to survive a disastrous situation with simple illustrations and written guides.

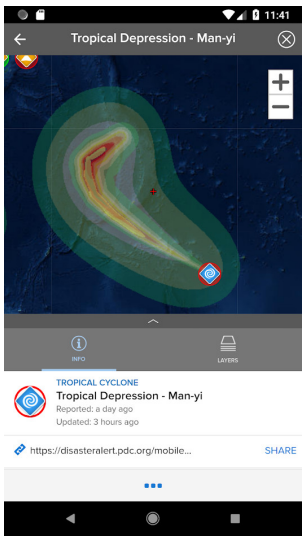

Figure 2.2: Disaster Alert is a free, mobile app that provides individuals, families, and their loved ones with the information they need to stay safe anywhere in the world. Disaster Alert offers near real-time updates about 18 different types of active hazards as they are unfolding around the globe.

3 Functional Requirements

3.1 Relevium Watchful Smart Helper

Communicate with the application through phone's built-in assistant.

3.1.1 Description and priority

We create intents in the agent that map user input to responses. In each intent, we define examples of user requests that can trigger the intent, what to extract from the request, and how to respond.

Generally, an intent represents one dialog turn within the conversation. The agent would match that input to its corresponding intent and return the response you defined within that intent. The agent's response usually prompts users for another utterance, which your agent will attempt to match another intent, and the conversation continues.

3.1.2 Usability

Relevium project uses Watchful Smart Helper as a advanced way to communicate with user. When the user log-in user will receive a welcome intent from pre-populated text responses from the responses which has already been trained.

3.2 Paramedic User

3.2.1 Description and priority

One of the most important features During the disaster. When the user register to our application user will be asked to fill a form to prove that the user a paramedic or not (optional), and if user fill this field, then user required to prove that by a picture for Work identity or any document Prove that user have the right to work with a paramedic degree.

3.2.2 Guidelines

There may be conditions under which the paramedic user will decide to get away or to be there to help others,there are no conditions for user to follow,he has full freedom to choose what he wants to do by helping others or escape in case of danger.

3.3 Voice Recognition

3.3.1 Description and priority

Alternatively referred to as **speech recognition, voice recognition** is a computer software program or hardware device with the ability to decode the human voice. Voice recognition is commonly used to operate a device, perform commands, or write without having to use a keyboard, mouse, or press any buttons.

Today, this is done on a computer with **ASR (automatic speech recognition)** software programs. Many ASR programs require the user to "train" the ASR program to recognize their voice so that it can more accurately convert the speech to text. For example, you could say "open Internet" and the computer would open the Internet browser. The first ASR device was used in 1952 and recognized single digits spoken by a user (it was not computer driven).

Today, ASR programs are used in many industries, including health care, military (e.g., F-16 fighter jets), telecommunications, and personal computing (i.e. hands-free computing).

3.3.2 Methodology

Voice recognition software on computers requires that analog audio be converted into digital signals, known as analog-to-digital conversion. For a computer to decipher a signal, it must have a digital database, or vocabulary, of words or syllables, as well as a speedy means for comparing this data to signals. The speech patterns are stored on the hard drive and loaded into memory when the program is run. A comparator checks these stored patterns against the output of the A/D converter, an action called pattern recognition.

In practice, the size of a voice recognition program's effective vocabulary is directly related to the random access memory capacity of the computer in which it is installed. A voice recognition program runs many times faster if the entire vocabulary can be loaded into RAM, as compared with searching the hard drive for some of the matches. Processing speed is critical, as well, because it affects how fast the computer can search the RAM for matches.

3.3.3 Usability

Relevium project uses voice recognition as an alternative way to carry out the communication with the user, we use it because of sometimes the user in critical situations won't be able to use the typing method to use the Relevium mobile app, it also enhance the communication and offer the user a variety of different possible ways to use them as input method or interaction ways with our application.

3.4 Evacuation plan

Find shortest safe path to evacuate and override traffic rules if necessary.

3.4.1 Description and priority

Evacuations are more common than many people realize. They are most frequently the results of fires and floods. Major storms such as hurricanes cause often result in mass-scale evacuations. In addition, hundreds of times a year, transportation and industrial accidents release harmful substances, forcing many people to leave their homes and places of work.

The amount of time you have to leave will depend on the hazard. If the event is a weather condition, such as a hurricane, you might have a day or two to get ready. However, many disasters allow no time for people to gather even the most basic necessities, which is why planning ahead is essential. Plan how you will assemble your family (or employees for workplace evacuation planning) and supplies and anticipate where you will go for different situations. Choose several destinations in different directions so you have options in an emergency and know the evacuation routes to get to those destinations

3.4.2 Usability

The country also lacks effective disaster preparedness system to confront natural disasters. Timely disaster warning and evacuation guideline can save lives of the people. In addition, a tourist or a blind people may face difficulties in finding safe area or shelter place prior to the occurrence of natural disasters.

3.4.3 Guidelines

There may be conditions under which you will decide to get away or there may be situations when you are ordered to leave. Follow these guidelines for evacuation:

- Plan places where your family will meet, both within and outside of your immediate neighborhood.
- Use the Family Emergency Plan to decide these locations before a disaster.
- If you have a car, keep the gas tank full if an evacuation seems likely. It is also good practice to maintain at least a half filled gas tank in the event of an unexpected need to evacuate. Gas stations may be closed during emergencies and unable to pump gas during power outages.
- Plan to take one car per family to reduce congestion and delay.
- Become familiar with alternate routes and other means of transportation out of your area. Choose several destinations in different directions so you have options in an emergency.

- Leave early enough to avoid being trapped by severe weather.
- Follow recommended evacuation routes.
- Do not take shortcuts, they may be blocked.
- Be alert for road hazards such as washed-out roads or bridges and downed power lines.
- Do not drive into flooded areas.
- If you do not have a car, plan how you will leave if you have to. Make arrangements with family, friends, or your local government.
- Take your emergency supply kit unless you have reason to believe it has been contaminated.
- Listen to a battery-powered radio and follow local evacuation instructions.
- Take your pets with you, but understand that only service animals may be permitted in public shelters. Plan how you will care for your pets in an emergency.
- Call or email the out-of-state contact in your family's communication plan. Tell him or her where you are going.
- Secure your home by closing and locking doors and windows.
- Unplug electrical equipment such as radios, televisions and small appliances. Leave freezers and refrigerators plugged in unless there is a risk of flooding. If there is damage to your home and you are instructed to do so, shut off water, gas, and electricity before leaving.
- Leave a note telling others when you left and where you are going.
- Wear sturdy shoes and clothing that provides some protection such as long pants, long-sleeved shirts, and a cap.
- Check with neighbors who may need aid.

3.5 Early alerting system

Predict hazardous environments and alert users.

The system leverage a third-party Disaster Management Server (DMS), mobile device with our application installed on it and Connect DMS to get updates about disaster (tsunami, cyclone or flood) to get automatic notification of upcoming disaster.

3.6 Direct users to shelters and highlight danger zones

When our application recognizes the user in probable disaster zone then application will disseminate visual and audio disaster warning and evacuation guideline including shortest path of shelter on the map of the application. Evacuation progress is also tracked using DMS.

3.7 Share location with user information

Track your location on the map and offer assistance.

3.7.1 Description and priority

Relevium tells user when a friend is nearby, it even lets you "wave" at them and gives you the option to send a message if they wave back. Also let you tell your friends where you are and give them directions to your location. It will also let you pick a special friend (like a family member, spouse or love interest, for example) to share your location with long-term.

3.7.2 Usability

To do so, tap the blue dot indicating where you are in the map or go to the side menu in the app and tap "Get Started" After that, just follow the same instructions but instead tap "Share Location," then choose any number of contacts with whom you want to share your location for a few minutes, hours, months or on an on-going basis.

Those who don't have Relevium can share through a short link via SMS. (Of course, only share that short link with someone you trust enough not to post it to the internet or share with others you don't know or want finding you.)

3.8 Network of mobiles during the disaster (MANET)

Stands for "Mobile Ad-hoc Network".

Connect users in the same geographical area using a wireless mesh network (WMN).

A MANET is a type of ad-hoc network that can change locations and configure itself on the fly. Because MANETS are mobile, they use wireless connections to connect to various networks. This can be a standard Wi-Fi connection, or another medium, such as a cellular or satellite transmission.

Some MANETs are restricted to a local area of wireless devices (such as a group of laptop computers), while others may be connected to the Internet. For example, A VANET (Vehicular Ad-Hoc Network), is a type of MANET that allows vehicles to communicate with roadside equipment. While the vehicles may not have a direct Internet connection, the wireless roadside equipment may be connected to the Internet, allowing data from the vehicles to be sent over the Internet. The vehicle data may be used to measure traffic conditions or keep track of trucking fleets. Because of the dynamic nature of MANETs, they are typically not very secure, so it is important to be cautious what data is sent over a MANET.

3.8.1 Description and priority

In the infrastructure based wireless networks a node can only send a packet to a destination node only via access point (in cellular network like GSM, it is called base station). The access point establishes a network area and only the nodes in this area can use access point's services. There are some unknown events, which cause access point's malfunction. The nodes lose their network and they are quasi not working.

It is the biggest infrastructure's disadvantage. There are also some reasons to sacrifice or not to use access point's services. These can be cost factor, impossibility to install access point in short time, etc. In this case, the nodes have to build its own network. This network is called wireless ad hoc network.

The wireless ad hoc networks only consist of nodes equipped with transceiver. The network are created to be independent from an infrastructure. Therefore, the nodes must be able to arrange their own networks. Keep in mind, that a node can now communicate only with other nodes in its transmission range. In the infrastructure based wireless network, the nodes can communicate with a node, which is located in another network area, by transmitting data to destination access point and this access point relay the data to the desired node.

It seems like, that the ad hoc networks are not powerful enough. Each node has its own transmission range, if these small transmission areas are combined, they will form a much bigger transmission area. The nodes transmit their data with single or multiple hopping technique. Now a suitable routing algorithm must be implemented, so the process of transmitting data will be more effective.

3.8.2 Usability

The usability of this feature in Relevium project will occur when the infrastructure of the cellular network or WLAN isn't available or accessible, so we can make the users who have Relevium mobile app communicate with their mobiles using Mobile ad-hoc network (MANET) to share knowledge and current situation that's happening at each person side, and to help society collaborate and quick help each other during disasters.

3.9 Web crawler to collect data from websites

Monitor Twitter, Reddit and other news aggregators for any events.

3.9.1 Description and priority

A Web crawler is an Internet bot which helps in Web indexing. They crawl one page at a time through a website until all pages have been indexed. Web crawlers help in collecting information about a website and the links related to them, and also help in validating the HTML code and hyperlinks. A Web crawler is also known as a Web spider, automatic indexer or simply crawler.

3.9.2 Methodology

The crawl started by making requests to the URLs defined in the script attributes, then call callback method to pass the response object as an argument. In the parsing callback, we loop through the response elements using a CSS Selector, yield a Statements which has specific keywords

3.9.3 Usability

Web crawlers collect information such the URL of the website, the meta tag information, the Web page content, the links in the web page and the destinations leading from those links, the web page title and any other relevant information. They keep track of the URLs which have already been downloaded to avoid downloading the same page again.

A combination of policies such as re-visit policy, selection policy, parallelization policy and politeness policy determine the behavior of the Web crawler. There are many challenges for web crawlers, namely the large and continuously evolving World Wide Web, content selection trade-offs, social obligations and dealing with adversaries.

Web crawlers are the key components of Web search engines and systems that look into web pages. Web crawlers are also used in data mining, wherein pages are analyzed for different properties like statistics, and data analytic are then performed on them.

4 Non-Functional Requirements

Nonfunctional Requirements (NFRs) define system attributes such as security, reliability, performance, maintainability, scalability, and usability. They serve as constraints or restrictions on the design of the system across the different backlogs.

4.1 Performance Requirements

What should system response times be, as measured from any point, under what circumstances? Are there specific peak times when the load on the system will be unusually high?

4.1.1 Relevium Watchful Smart Helper

This feature should be completed in time that ranges between **(3 - 5) seconds**

4.1.2 Voice Recognition

The response time of voice recognition should be in the range between **(2.5 - 5) seconds** in case of online, and in case of offline it should be between **(2 - 4) seconds**

4.1.3 Early alerting system

This system triggered in the online mode before the disaster happen, so any short delay in response will be acceptable and the response time should be between **(7 - 10) seconds** and under load it

4.1.4 Network of mobiles during the disaster (MANET)

This feature is meant to be used in offline mode and during disasters, so it should have maximum time of **10 seconds** for critical conversations between people.

4.2 Safety Requirements

User agrees by using this system that under no circumstances or any theories of liability under international or civil, common or statutory law including but not limited to strict liability, negligence or other tort theories or contract, patent or copyright laws, will the system be liable for damages of any kind occurring from the use of this system or any information, goods or services obtained on this website including direct, indirect,

consequential, incidental, or punitive damages (even if the system has been advised of the possibility of such damages), to the fullest extent permitted by law. Some jurisdictions do not allow the exclusion or limitation of certain damages so some of these limitations may not apply to you.

4.3 Security Requirements

This application will have users credentials like (mobile numbers, addresses ..etc) So it must have encrypted personal data and the chat between users should be encrypted too. Some of the supposed algorithms to be used are : **AES-256, ARC4, RSA-2048**

- User ID: Any user who uses the system shall have a User ID.
- User Identification: The system requires the user to identify himself / herself using secure credentials. The user credentials are typically some form of "username" and a matching "password",
- Modification: Any modification (insert, delete, update) for the database shall be synchronized and done only by the administrator in the ward.
- Administrators Rights: Administrators shall be able to view and modify all information in system.

Other "external" factors which can be used alongside the user's identification to securely reset a password may be voice recognition, fingerprints, or smart-cards. If the person requesting the reset can show they have two or more specific elements – such as knowledge, a possession, or something inherent to the user and only the user – that only the account holder should have, then the password reset mechanism can be triggered.

4.4 Software Quality Attributes

This mobile app should guarantee availability as it will be available every time and it must maintain lowest MTBF (mean time between failure) also it must meet correctness because any misinformation could lead to dangerous situation for the user, it should be flexible to be used easily in hard times, also it should offer the testers an instructions about how the application should be tested and what is the correct input and output, it also should be adaptable to any environment changes, app must meet portability condition as it used in a critical situations, the last but not least is the maintainability concern and it should be maintained at least one time per week.

5 Appendix A: Analysis Models

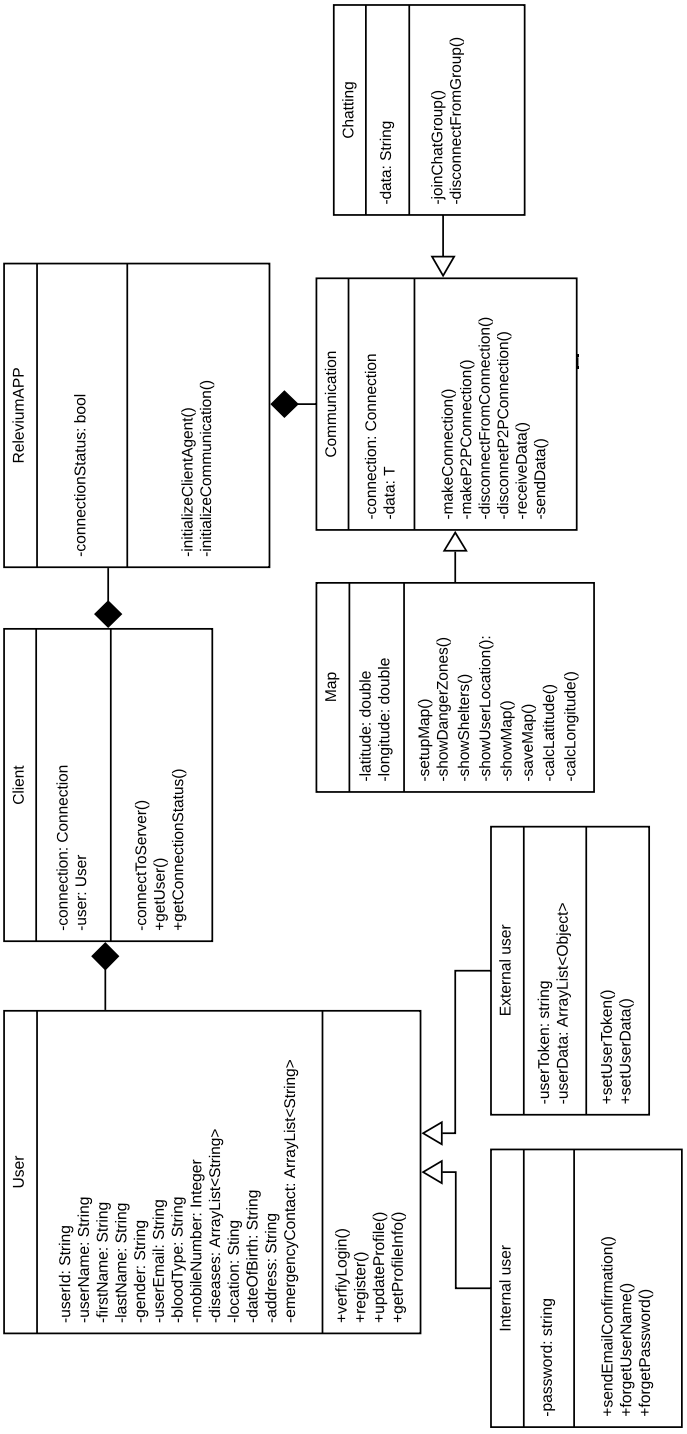

Figure 5.1: Class Diagram - Client Domain Logic

User Class: The user class contain all the information about the user, that the app uses to populate all the fields in the profile tab. All the class variables are set on the class constructor, in contrast the user can be an ***internal*** or ***external*** User, both ***inherit*** the class ***User***.

- ∇ **verfiyLogin()** This method takes the user ID and password or user token and authenticate it with our servers to verify user log-in credentials, finally it returns a Boolean which it's value depends on success or failure of the log-in process.
- ∇ **register()** This method takes the user to the registration process, which asks him/her about all the required data to fill.
- ∇ **updateProfile()** This method is used to update any user profile info and return Boolean depends on its succeed on updating or not.
- ∇ **getProfileInfo()** This method is used by the application to populate all the fields of the the profile tab and it return an ArrayList of all the different user variable objects.

Internal User Class The internal client class is the class responsible for all the clients who registered through our servers, it handles the user log-in process, register and credentials loss.

External User Class The external client class is the class responsible for all the clients who sign in through external API(s) eg: (Gmail, Facebook, etc...) and get their token and save it in our servers for future log-in. The class also get the data of the user from the external API after claiming the token, finally it redirect the new users to the register process with completed fields of their claimed data from the external API. If the external client did register before, then he/she will be sent to the Home screen of Relevium application.

Client Class: Is the root class in the client which responsible for establishing connection with the server and instantiate a user object to access user info and it populate ReleviumAPP class.

- ∇ **connectToServer()** Used to establish http connection with the server.
- ∇ **getUser()** Used to get user object by the others classes.
- ∇ **getConnectionStatus()** Used to get connection status by other classes.

ReleviumAPP Class: is the main Activity in the mobile app by which other activity will be called, First of all, the class will check the connection status as it initialed, if there is a connection available, if so it will set **connectionStatus** to be true, else it will be false. The **ReleviumApp** which will has multiple buttons, map "initialized by default on sub-Activity, chat, and the Assistant, and when a user push one of these buttons, activity will populate to a new sub-activity, that handle the requested function.

- ∇ **initializeAgent()** Linked to assistant button and used to connect to the agent and populate to new sub-activity to view the agent.

- ∇ **initializeChat()** Linked to Chat button and used to populate to the Chatting activity and it will use communication protocols defined in communication class.
- ∇ **initializeMap()** Linked to Map button and used to populate to the Map activity and it will use communication protocols defined in communication class.

Communication: is the class used to handle the communication between the server and the client or provides an easier way to create and manage sessions between users without internet connection, only via Wi-Fi or the Bluetooth. The class is inherited by map and chatting class.

- ∇ **makeP2PConnection()** is used to establish peer to peer connection with other mobile device.
- ∇ **makeConnection()** is used to handle requests between the client and the server or an API server.
- ∇ **disconnectP2PConnection()** is used to disconnect from established connection.
- ∇ **disconnetConnection()** is used to disconnect from established connection.
- ∇ **receiveData()** used to receive data from other device that connected with.
- ∇ **sendData()** used to send data to other device that connected with.

Map class: is the class that handle the connection between Geo-Location API and the Mobile APP, and perform all functions related to Geo-Location API. The class will be called by controller by default after the user log-in as a sub-activity, and it inherit P2P connectivity.

- ∇ **setupMap()** used to setup connection between the Geo-location API and set all required configuration.
- ∇ **showDangerZones()** connect to the API to set Danger zones Layers on the Map.
- ∇ **showShelters()** set shelters location on the map the recommend shortest path from user location to shelters.
- ∇ **showUserLocation()** set user location on the map
- ∇ **showMap()** used to populate map on the screen
- ∇ **calcLatitude()** used to set the Latitude Attribute using the GPS.
- ∇ **calcLongitude()** used to set the Longitude Attribute using the GPS.

Chatting Class: is the class used to handle chatting between users online and offline, so it inherit P2PConnectivity to work in offline state, the user join a group based on his location to chat with other users in the same area to ask for help or help others users. it will operate as sup-activity when the user push the chat button on the main view.

- ∇ **joinChatGroup()** used to connect user to a chat group based on his location.
- ∇ **disconnectFromGroup()** used to disconnect form chat group.

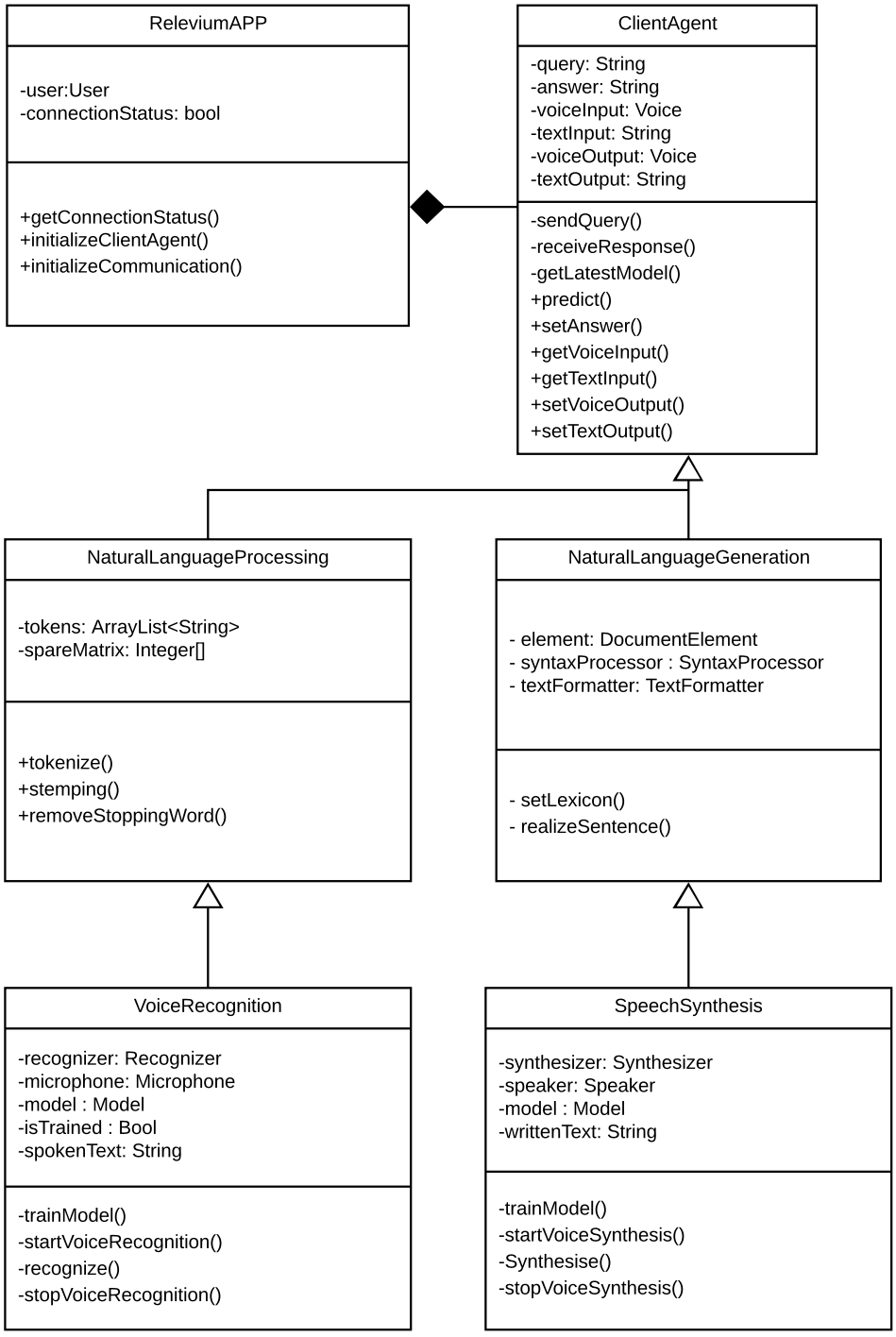

Figure 5.2: Class Diagram - Server Domain Logic

Client Agent Class: the class the handle connection between the online agent if the online connection is available, but if the online connection is not available the agent will perform the prediction on the latest downloaded model.

- ∇ **sendQuery()** used to send Query entered from user to server Agent.
- ∇ **receiveResponse()** used to receive Response from server Agent.
- ∇ **getLatestModel()** used to download current working version model from the server agent.
- ∇ **predict()** used to perform classification on the locally downloaded model.
- ∇ **setAnswer()** used to set the answer to the value resulted from predict method.
- ∇ **getVoiceInput()** used to access voice input and it used by inherited classes.
- ∇ **getTextInput()** used to access text input and it used by inherited classes
- ∇ **setVoiceOutput()** used to set voice output and mainly used by speech synthesis.
- ∇ **setTextOutput()** used to set text output and mainly used by natural language generation.

NaturalLanguageProcessing Class: Is the class that process input text or, voice that converted to text and generate a sparse matrix to apply it a classification model. it also handle text pre-processing.

- ∇ **tokenize():** Used to convert the corpus to token.
- ∇ **stemming():** Used to convert words to its origin example "Visiting" will converted to be "Visit".
- ∇ **removeStoppingWord():** used to remove unwanted words from text.

VoiceRecognition Class: voice recognition class: is the class the handle and process the input voice and converting it to readable text by using Neural network based model.

- ∇ **trainModel():** Used to train model on training set.
- ∇ **startVoiceRecognition():** Used to set the voice recognition to listening mode.
- ∇ **recognize():** Used to recognize and convert voice to text.
- ∇ **stopVoiceRecognition():** Used to release Voice Recognition resources.

NaturalLanguageGeneration Class: Is the class that responsible for creating and generating human language text from a machinery text.

- ∇ **setLexicon():** Generate tokens based on specific context.

- ∇ **realizeSentence():** Used to realize and generate accurate pronunciation to the user.

Speech Synthesis Class: Is the class responsible for generate human voice dialog for specific context from components of words to form voice signal.

- ∇ **trainModel():** Used to train model on training sets of texts.
- ∇ **startVoiceSynthesis():** Used to start Voice Synthesis operation and start receiving text to convert it to human voice.
- ∇ **Synthesise():** Used to apply Synthesis algorithms and convert text to human voice.
- ∇ **stopVoiceSynthesis():** Used to release Voice Synthesis resources.

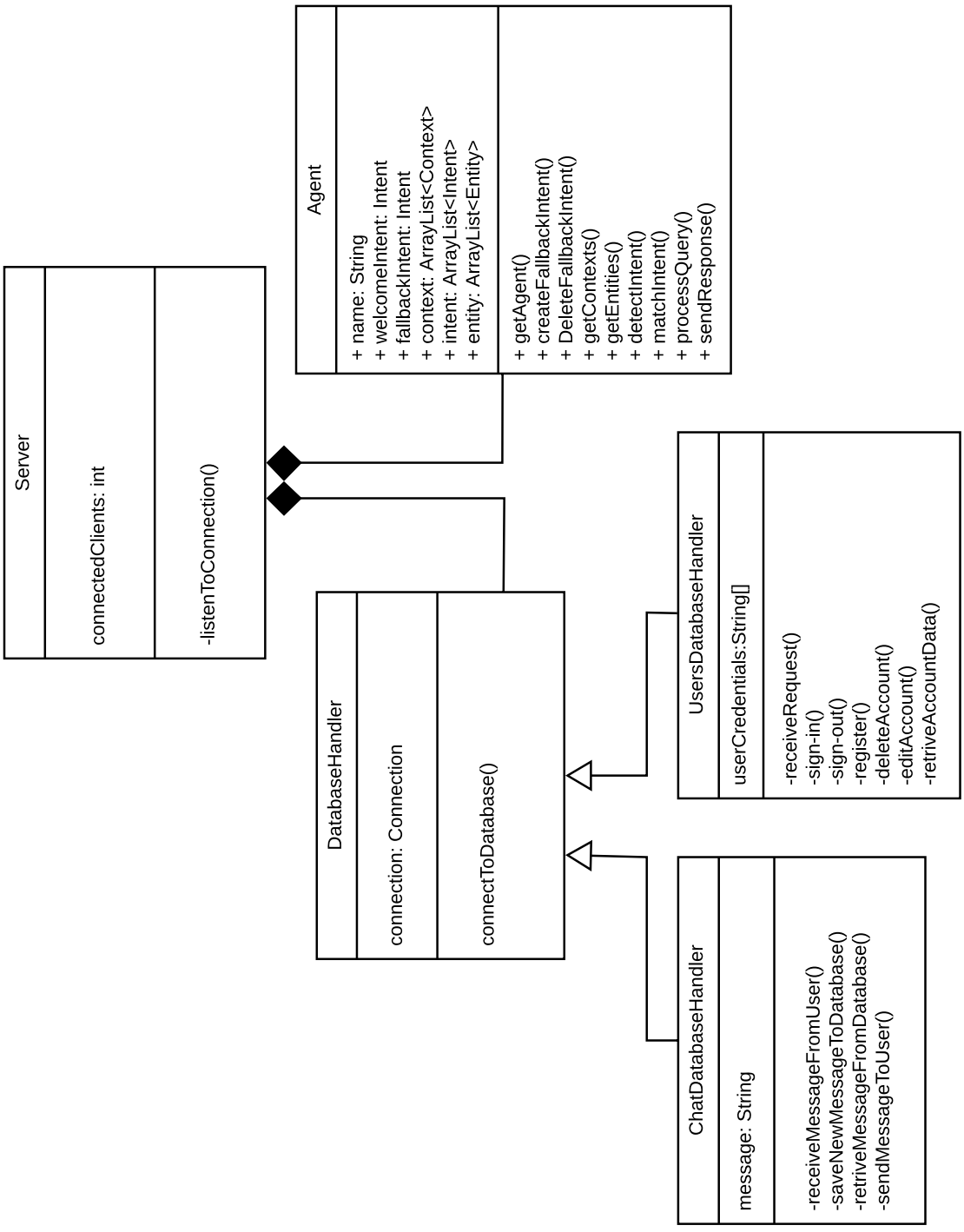

Figure 5.3: Class Diagram - Server Domain Logic (cont.)

Client Class: Is the root class in the client which responsible for establishing connection with the server.

- ∇ **ConnectToServer():** Used to establish http connection with the server.

Server Class: Count connected clients to the server.Also contains Database Class and Agent Class which has a composition relation with the server.

- ∇ **ListenToConnection():** Used to get the requested connection from client.

Database Class: The database class, is the class responsible for connect the server to the database.Also Database Class contains two sub-classes which is ChatDatabase and UserDatabase.

- ∇ **reciveMessageFromUser():** Used to get message from the user.
- ∇ **saveNewMessageToDatabase():** Used to save new message from the user to database.
- ∇ **retriveMessageFromDatabase():** Used to get message from the database.
- ∇ **saveNewMessageToDatabase():** Used to send message saved on database to the user.

UsersDatabase Class: Used to identify that information entered are correct (user credentials).

- ∇ **receiveRequest():** Used to receive connection request from user.
- ∇ **sign-in():** Used to sign in to the app using information has entered before by user.
- ∇ **sign-out():** Used to sign out from the app.
- ∇ **register():** Used to register a new account to the app by Fill all text fields showed to user.
- ∇ **deleteAccount():** Used to delete account that user created.
- ∇ **editAccount():** Used to edit account details or data that entered when user register.
- ∇ **retriveAccountData():** Used to show user data.

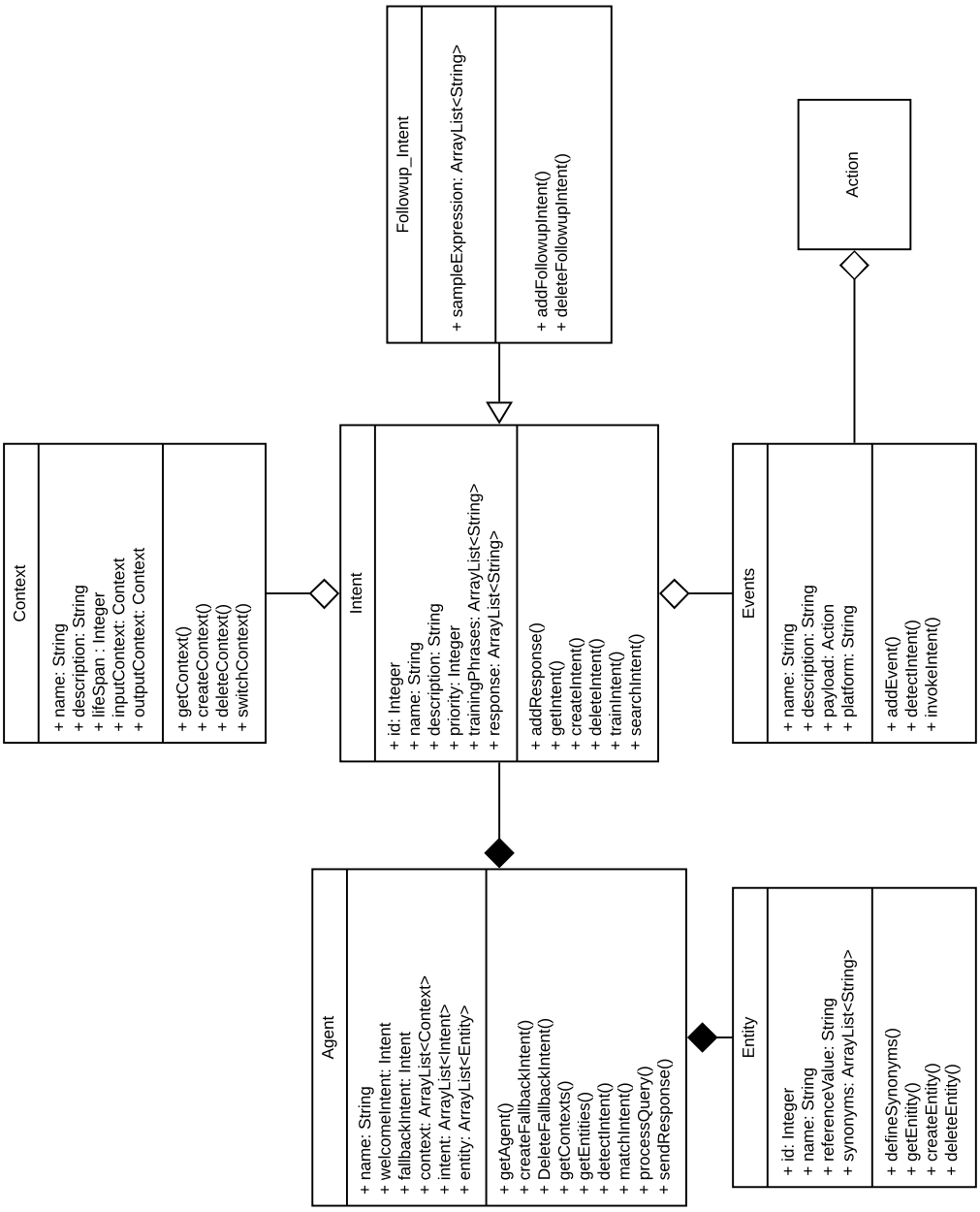

Figure 5.4: Class Diagram - Agent Implementation

Intent: To define how conversations work, we create intents in the agent that map user input to responses. In each intent, we define examples of user requests that can trigger the intent, what to extract from the request, and how to respond.

Generally, an intent represents one dialog turn within the conversation. The agent would match that input to its corresponding intent and return the response you defined within that intent. The agent's response usually prompts users for another utterance, which your agent will attempt to match to another intent, and the conversation continues.

Intent Components:

- Intent name: The name of the intent. The intent name is passed to your fulfillment and identifies the matched intent.
- Training phrases: Examples of what users can say to match a particular intent. Agent automatically expands these phrases to match similar user utterances.
- Action and parameters: Defines how relevant information (parameters) are extracted from user utterances. Examples of this kind of information include dates, times, names, places, and more. You can use parameters as input into other logic, such as looking up information, carrying out a task, or returning a response.
- Response: An utterance that's spoken or displayed back to the user.

Primary Intents:

1. Welcome Intent: When the Welcome Intent is matched, agent will respond with one of the pre-populated text responses from the **Responses**. Each time this intent is matched, agent returns one of the responses. For each subsequent matching, agent selects a different response until all responses have been used. then, it begins again choosing responses randomly.
2. Fallback Intent: Triggers if a user's input is not matched by any of the regular intents or if it matches the training phrases.
3. Follow-up Intent: provide a simple way to shape a conversation without having to create and manage contexts manually. These special intents are nested under their parent intent and are designed to handle preset replies from the user.
   a) **Yes**: "*yes*", "*do it*", "*sure*, "*exactly*", "*confirm*", "of *course*", "*sounds good*", "*that's correct*", "*I don't mind*", "*I agree*".
   b) **No**: "*no*", "*don't do it*", "*definitely not*", "*not really*", "*thanks but no*", "*not interested*", "*I don't think so*", "*I disagree*", "*I don't want that*".
   c) **Later**: "*later*", "*not yet*", "*ask me later*", "*not*", "*next time*", "*I said later*", "*some other time*", "*do it later*", "*not at the moment*", "*not at this time*".
   d) **Cancel**: "*cancel*", "*stop*", "*dismiss*", "*skip*", "*forget that*", "*stop it*", "*never mind*", "*do nothing*", "*just forget about it*", "*cancel that*".

e) **More**: "*more*", "*more results*", "*anything else*", "*other results*", "*what else*", "*are there more*", "*tell me more*", "*show me more*", "*more information*".

f) **Next**: "*next*", "*next page*", "*go forward*", "*show me next*", "*following result*", "*switch to the next*", "*go to the next page*", "*read the next results*", "*show me the next page*", "*show me the following results*".

g) **Previous**: "*back*", "*previous*", "*go back*", "*previous page*", "*previous results*", "*return to the previous*", "*return to previous page*", "*go to the previous page*", "*go back to previous results*", "*read out the previous results*".

h) **Repeat**: "*repeat*", "*repeat it*", "*come again*", "*do it again*", "*say it again*", "*say the same again*", "*please repeat that*", "*repeat these results*", "*could you repeat that*", "*repeat what you've just said*".

Context: Contexts represent the current state of a user's request and allow your agent to carry information from one intent to another. You can use combinations of input and output contexts to control the conversational path the user takes through your dialog.

Contexts let agent control conversation flows by letting agent define specific states that a conversation must be in for an intent to match. Normally, Agent matches an intent if its training phrases closely resemble the user utterance. However, when you apply contexts to an intent, Agent will only consider that intent for matching if the context is active.

Contexts are very helpful in controlling order of intent matching & creating different outcomes for intents with the same training phrases.

There are two types of context that let you activate and deactivate contexts and can control the flow of your conversation:

- Input contexts: When applied to an intent, an input context tells agent to match the intent only if the user utterance is a close match and if the context is active.
- Output contexts: When applied to an intent, an output context tells agent to activate a context if it's not already active or to maintain the context after the intent is matched.

Entity: Entities are the agent's mechanism for identifying and extracting useful data from natural language inputs.

While intents allow the agent to understand the motivation behind a particular user input, entities are used to pick out specific pieces of information that the users mention. Any important data we want to get from a user's request will have a corresponding entity.

Event: Events allow agent to invoke intents based on something that has happened instead of what a user communicates. Agent supports events from several platforms (like Google Assistant, Slack, and more) based on actions users take on those platforms.

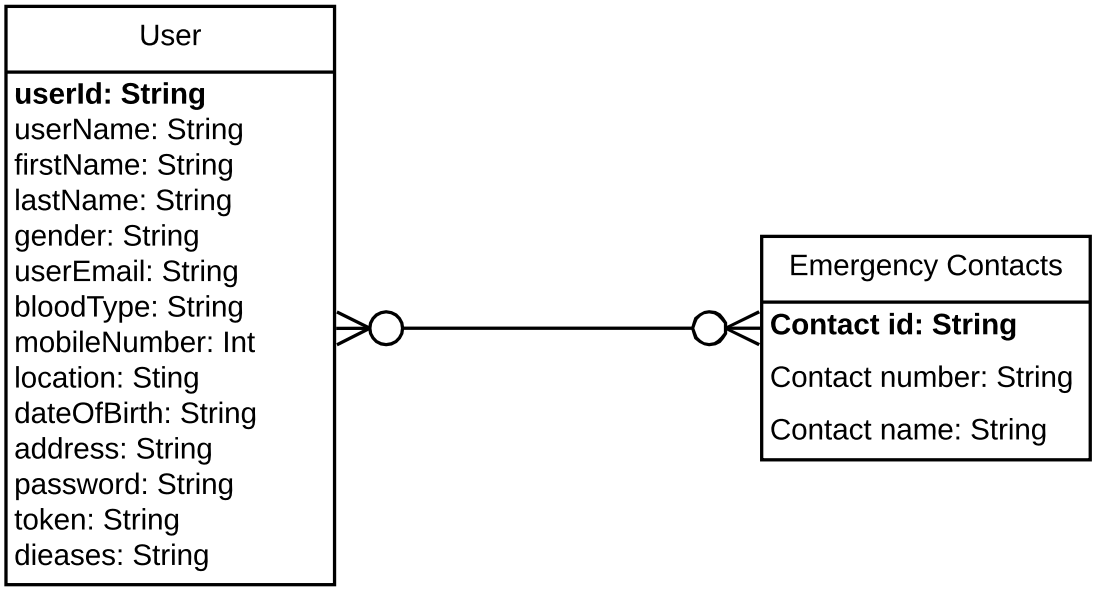

Figure 5.5: Entity Relationship Diagram - User

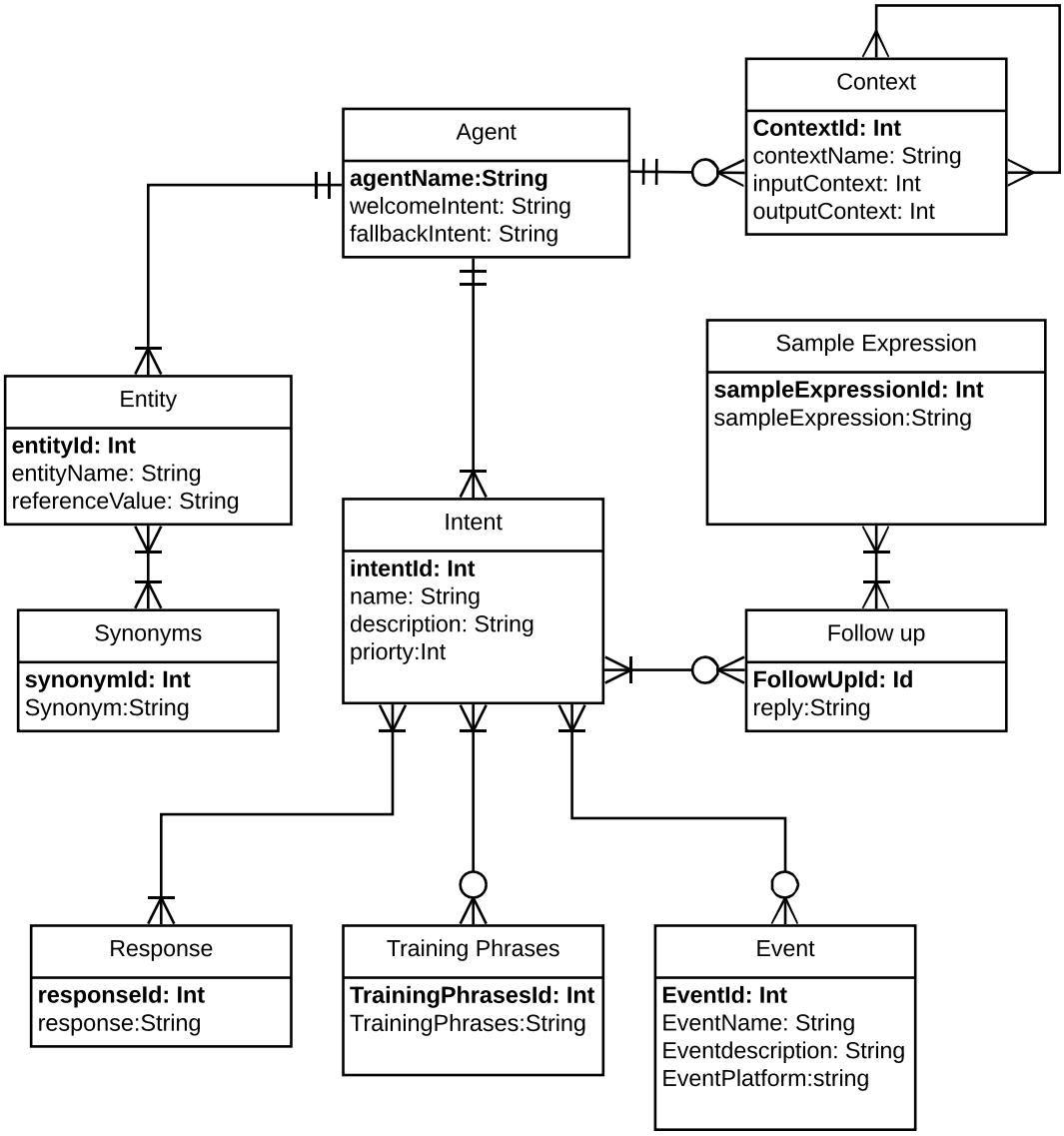

Figure 5.6: Entity Relationship Diagram - Agent

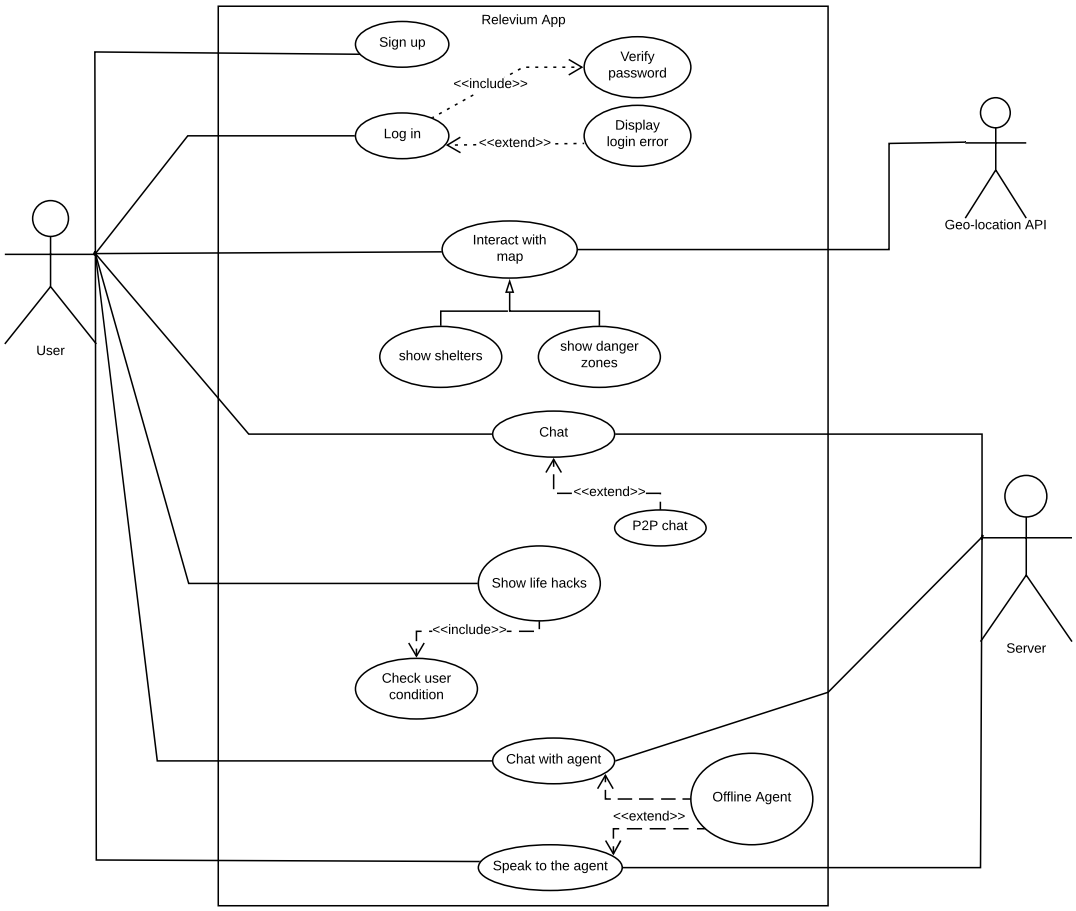

Figure 5.7: Use Case Diagram - Application Interaction

There are three actors User, Geo-Location API,Server. User: will have to sign up if he doesn't have an account and it will check for the credentials if they are correct or wrong, he also can log-in but if credentials are wrong it will display a log-in error, he can also interact with the map to navigate danger zones and safe zones(shelters) using the information provided by Geo-Location API, during disasters each user can chat to nearby users while there are no cellular infrastructure, in addition to the user can show life hacks or some evacuation plans dependant on the current situation, user can also speak to the app.

- Geo-Location API: provide the user with geo-location information.
- Server: Contains database and control the online chat flow with other users.

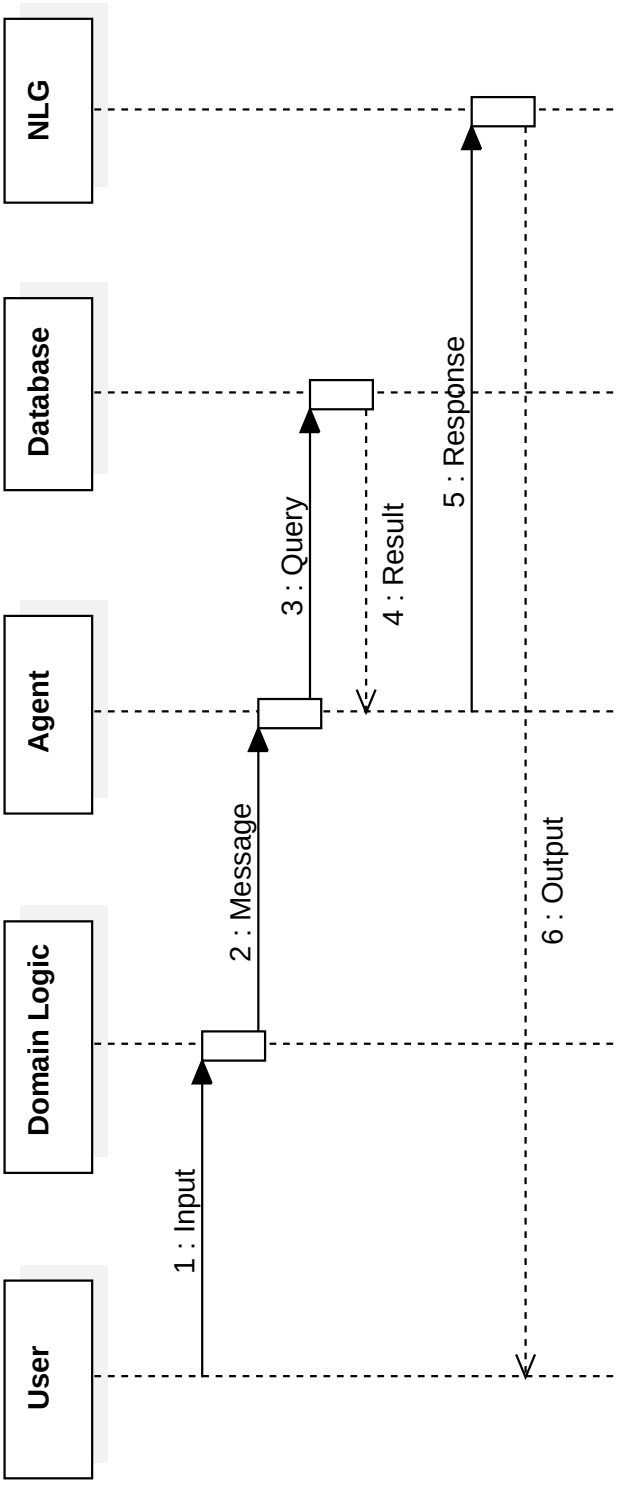

Figure 5.8: Sequence Diagram - Utterance Process

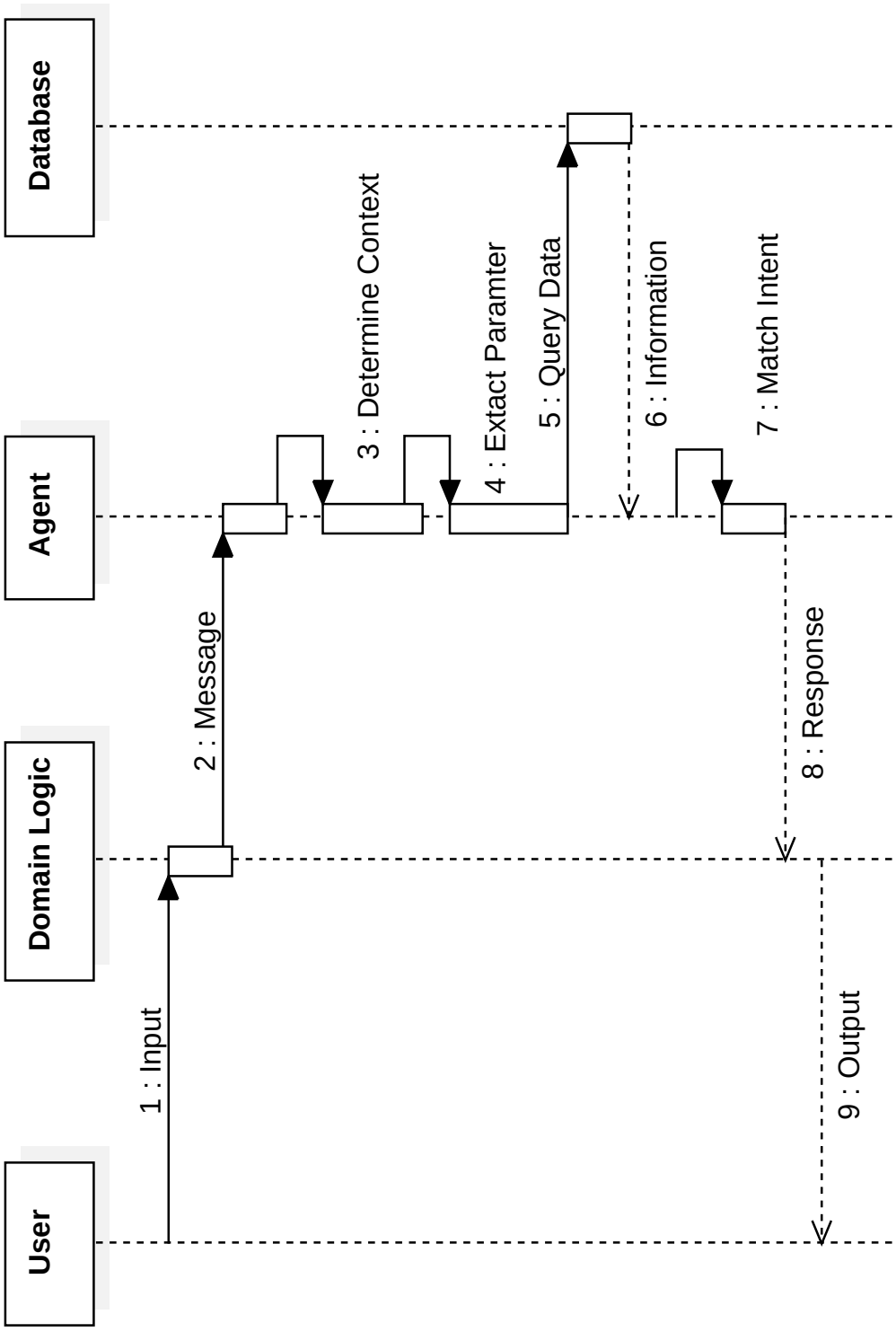

Figure 5.9: Sequence Diagram - Communication Route

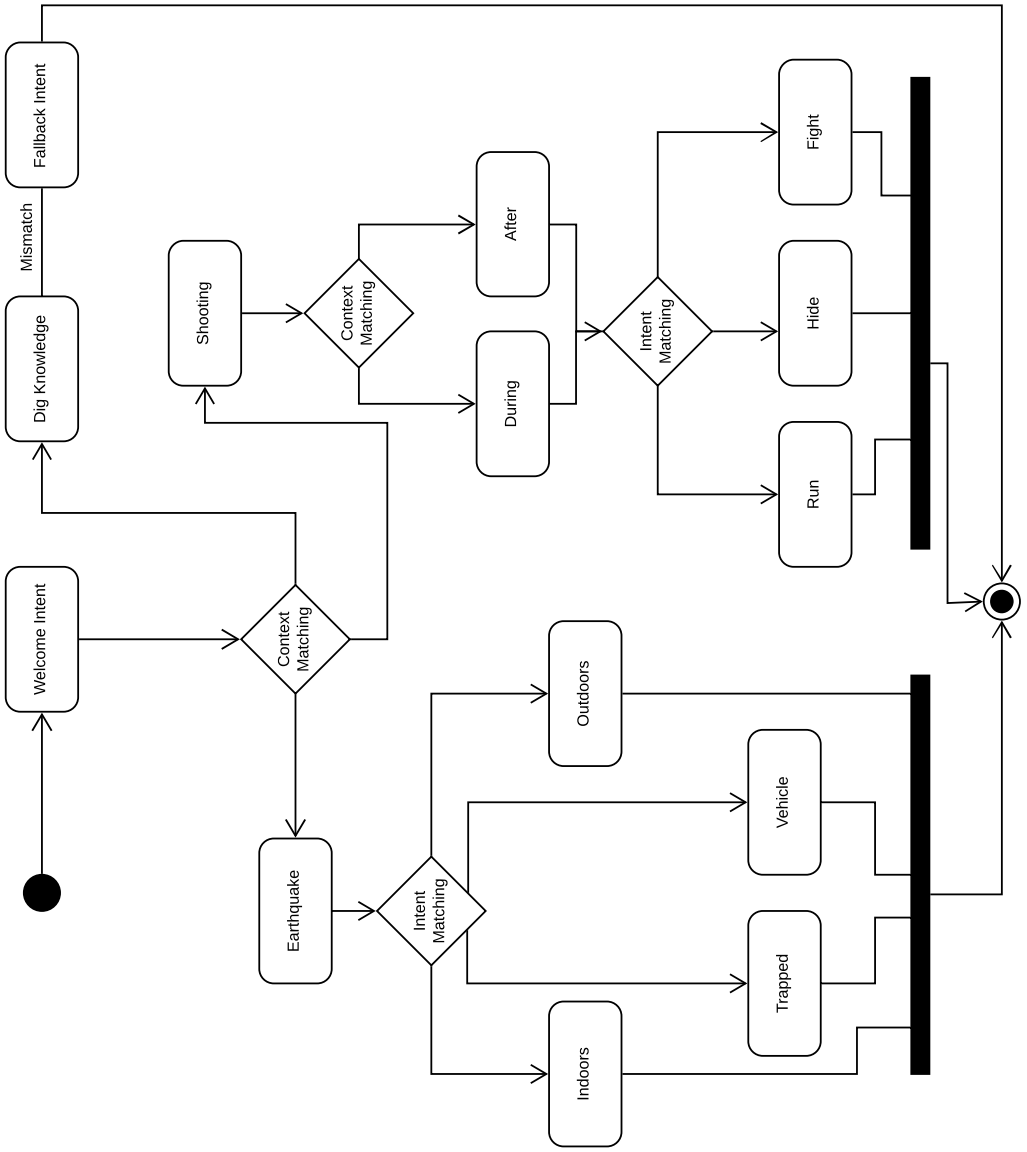

Figure 5.10: State Chart Diagram - Context Matching

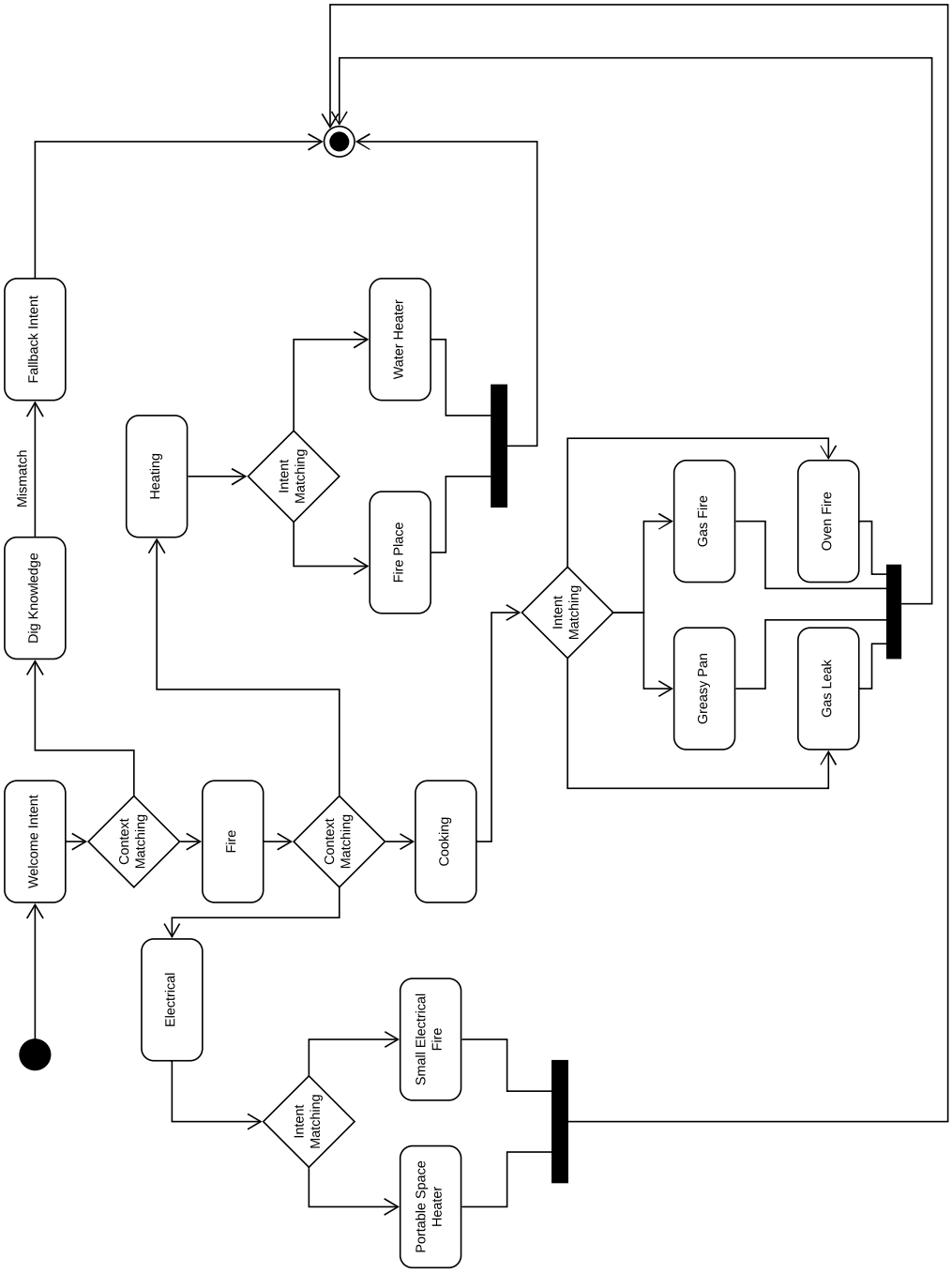

Figure 5.11: State Chart Diagram - Context Matching (cont.)

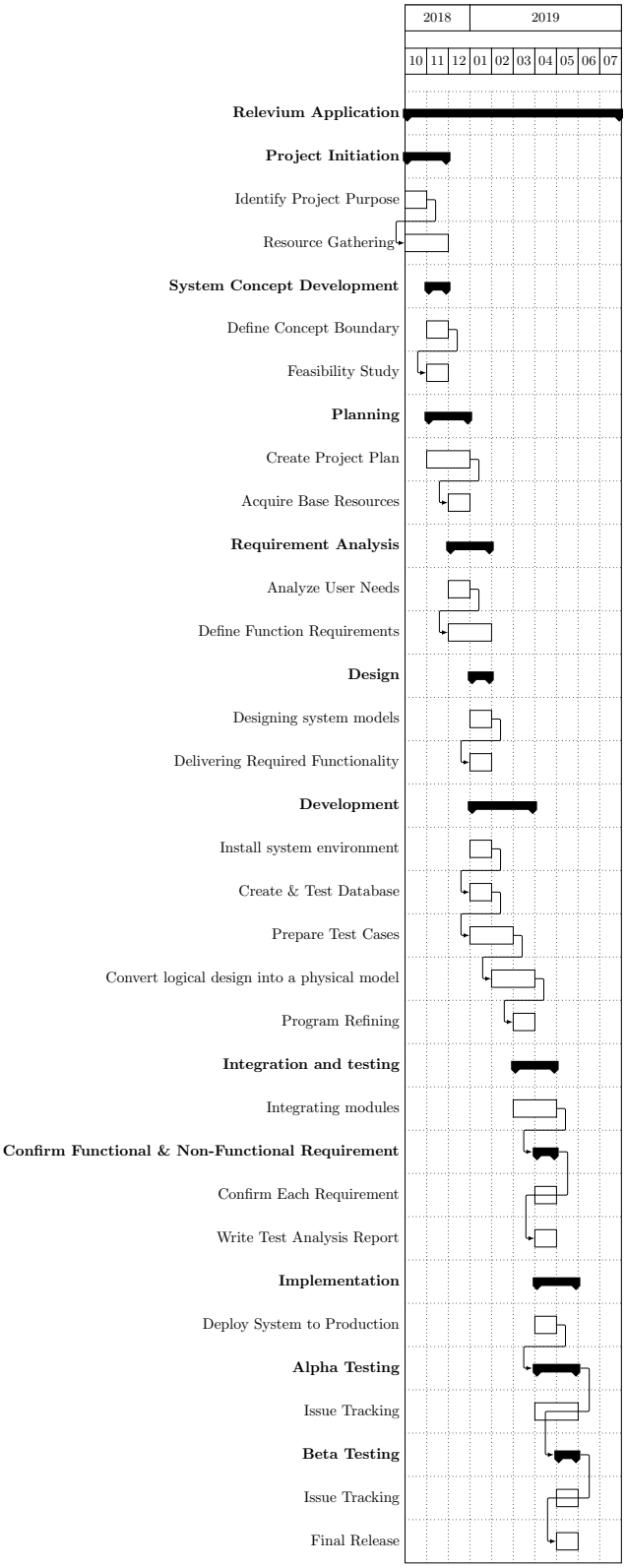

Figure 5.12: Gantt Chart - Compact View

Figure 5.13: Gantt Chart - Detailed View