Jamura: A Conversational Smart Home Assistant Built on Telegram and Google Dialogflow

Sanket Salvi*, Geetha V†, Sowmya Kamath S‡
Department of Information Technology, National Institute of Technology, Karnataka
Surathkal, Karnataka, India
Email: *sanketsalvi.salvi@gmail.com, †geethav@nitk.edu.in, ‡sowmyakamath@nitk.edu.in

***Abstract*—With an ever-increasing number of smart connected devices for various applications, there is a need for finding a new, smarter way of communicating with all the homogeneous and heterogeneous devices in a particular network. Conversational Bots, also known as Chatbots, are currently a popular solution in many applications, as they provide a user-friendly interface and more intuitive recommendations to user queries. In this work, the domain of home automation is considered from the area of the Internet of Things, and a Chatbot application built using technologies like Natural Language Processing, Machine Learning, and Service-Oriented Computing is designed as an intuitive user-interface for Smart home products. The aim of this paper is to build easy to implement and integrate DIY Smart Home Assistant using available technologies. The proposed Conversational Artificial Intelligence system can aid the user in smart decision making, predictive and preventive analytics, and showed promising results during experimental evaluation.**

***Index Terms*—Web of things, Conversational agents, Home Automation, Intelligent Decision-making, Chatbot**

I. Introduction

A Chatbot is an individual computerized aide who is built to support human-like interaction. It collaborates with the client through a verbal or textual chat interface, giving a virtual talking accomplice to speak. Though regularly utilized as a part of a dialog system for functional purposes similar to client administration or data procurement, a Chatbot can be customized to give some assistance for several intents. For example, tasks such as product recommendations, food recommendations, checking the weather, or planning a gathering can be easily handled by a Chatbot. From the early 60s to present day, Chatbots have undergone various improvements concerning technology, accuracy, efficiency, and application. The current generation of Chatbots [1] has evolved to be more intelligent due to advances in Machine Learning, Natural Language Processing, and Big Data Analytics. Such innovative technologies are also made available in the public domain in the form of APIs; developers can utilize that for building customized Chatbots, and thus increase its utility for a variety of end-user applications.

On the other hand, the emerging concept of Web of Things (WoT) as a platform for simple interaction and integration of smart devices in the Internet of Things (IoT) realm, can be seen as a significant development towards improving the ease-of-access and usability of IoT infrastructures, specifically in customer-oriented product markets like smart homes and smart cars. Guinard et al. [2] described the Web of Things paradigm emphasizing its features like support for modeling functionality as Linked Resources, representing resources, service management through a uniform interface, Thing syndication, Callback feature for Things through WebHooks and Web-enabling Constrained Devices, etc. Milson et al. [3] demonstrated the application of WoT concepts in AI-enabled product developments, like those in popular products such as Amazon Alexa, Apple Siri, and Google Home.

Over the past few years, developers have attempted to build interactive applications in the smart home domain to enhance ease-of-use. Obaid et al. [4] developed a ZigBee based voice-controlled smart home system that integrates device level interaction with user speech recognition and personalization. Guinard et al. [5] designed Sun SPOT, as a use-case of WoT in Smart Power Grids. Their work provided an insight into the need and development of web services and REST APIs, their expected formats and functionalities. Their work uses the ReST paradigm [6] for building a searchable network of things powered by RESTful services [7] [8].

Guinard et al. [9] described a Social Web of Things, wherein the Smart Devices can be part of the Social Network and can communicate with each other in a more natural and human way. Their work also discussed the development of OpenSocial, an Open Social Network API Stack to build Social Web of Things. This work provides an insight into understanding the interactions between user-level interfaces and Devices. Embodied Conversational Agents (ECAs) were proposed by Santos-Prez et al. [10] that are developed to be a natural interface between humans and Ambient Intelligence. ECAs can help people in their general daily routine and can be built on an architecture based on open-source tools and libraries. Their prototype virtual agent called AVATAR [10] acts as a natural control interface of the home automation system.

Richardson and Ruby [11] developed an integrated Telegram based messaging platform with camera-mounted Raspberry Pi that provides a visual of an intruder, for the house owner. An advanced version of the same concept was proposed by Ngu et al. [12], where the entity in front of the door could be identified, and an appropriate message corresponding to the findings can be sent to house owner for further instructions. Guinard et al. [13] also proposed a Telegram based application for water consumption monitoring and alerting, that uses Ubidots IoT platform for data storage and visualization while

978-1-7281-1895-6/19/$31.00

Water level sensors are used to monitor water consumption.

In this paper, we consider the Smart Home space as a use-case for the development of an Intelligent Conversational Agent-based user interface. The Chatbot is named as 'Jamura,' as the application of Chatbot is analogous to the entertainer who plays a particular sort of sidekick part in the conventional folk theater of India, who is called as 'Jamura.'

The proposed Chatbot aims at providing the ability to monitor and control household Internet-connected devices and respond to queries about overall house and garden health. The typical house health parameters like temperature, humidity, light intensity, the operational status of lights and fans in each room, soil moisture of garden and operational status of the water pump can be made easily accessible and controllable via natural language commands using proposed Chatbot. In addition to this, integration of Smart Door System is also proposed, which can identify the presence of human being at the door and assist the home-owner for Intruder Detection and/or automate decision support for further actions. The proposed home automation system uses the concepts of Web of Things to make services, devices, and related data more transparent and accessible across the network while maintaining data integrity.

The rest of this paper is organized as follows. In Section II, we describe the proposed framework, its architecture, and the various functionalities designed for Jamura. Section III presents the experimental setup and implementation specifications of the proposed framework. In Section IV, we discuss the experimental results and outcomes of our implementation, followed by a conclusion and future work.

II. Proposed Framework

Fig. 1 shows the functional architecture and logical arrangement of components that support the proposed functionalities of Jamura. Here, the bottommost layer depicts Sensors, Actuators, and Micro-Controllers, deployed in respective locations in the Smart home environment. The data collected from the devices in the lowest layer is aggregated and stored on Jamura's Local Server, which also works as a standalone Intruder Detection/Alert System. The data can be accessed in the local network, and it is updated over ThingSpeak Cloud for remote monitoring. Two Webhooks are used to process and redirect requests and responses between ThingSpeak and DialogFlow depending on the type of information. ThingSpeak Webhook will handle Local Client related queries while Telegram Webhook will handle Intruder Detection/Alert System-related queries. Out of available Chatbot platforms like IBM Watson Conversation Service, AgentBot, Twyla, Api.ai, Microsoft Bot Framework, Microsoft Language Understanding Intelligent Service (LUIS), etc., Dialogflow was selected for the implementation of Jamura, due to its ease of implementation, wide range of features and integration options. The final layer acts as an endpoint for capturing user interactions. With respect to our work, this layer could be well-served by both Telegram and Google Assistant. Telegram is selected due to its popularity and ease of integration using APIs, while Google Assistant is selected due to its support for speech recognition.

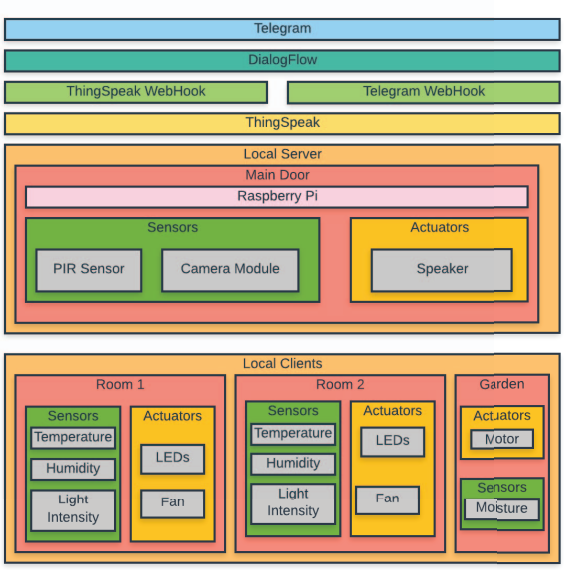

Fig. 1. System Architecture for Jamura

For the sake of simplicity, 'Room 1', 'Room 2', 'Garden' shown in Fig. 1, will be addressed as Local Clients and 'Main Door' will be addressed as Local Server. Based on functionality in a few situations, the Local Server could also behave as a Client. The main door has two tasks to handle - capture the image when the PIR sensor detects a visitor at the door, and act as a local server collecting data from Local Clients.

A. Jamura's Web of Things Architecture

This section describes the components of the Web of Things Architecture designed for Jamura. It consists of four layers - Access, Discovery, Share, and Compose, the details of which are discussed in detail below:

1) Access: This layer is responsible for turning any *Thing* into a programmable *WebThing* that other devices and applications can easily talk to. *Things* can be smoothly integrated to the web by exposing their services through a RESTful API using HTTP, built on top of TCP/IP, as well as the JSON data format. The data captured by each of the Local Clients must be well-formed in light-weight JSON format to enable interaction with the Local Server, by consuming provisioned RESTful APIs.

2) Discovery: In this layer, we propose an HTTP-based protocol with a set of resources, data models, payload syntax, and semantic extensions that *WebThings* and applications should follow. In the case of non-IP based clients, the JSON data sent by the Local Client should contain a Unique Identifier. For designing a scalable and protocol-agnostic infrastructure, we propose to incorporate non-IP based identification in Jamura.

3) Share: At this level, we look into applying fine-grained sharing mechanisms on top of RESTful APIs. This is used to establish a few complicated tasks like collective data gathering, collaborative functioning, or device to device communication. The proposed system will use Google Vision API, Google Dialowflow API, and Telegram API to achieve data sharing over the Internet and provide easy to use access for Users.

4) Compose: At the compose layer, the challenge is to build large-scale, meaningful applications over the Web of Things infrastructure of Jamura. The highest level, i.e., the design of the User Interface (UI) and the seamless integration of the different layers has to be addressed. However, in the proposed system, existing interfaces like Telegram and Google Assistant will be used.

B. Webhooks

A Webhook which is also widely referred to as an HTTP push API or web callback [2] [5] [9] is a method for enabling an application to exchange data with other application for processing. Unlike regular APIs where one must request for data at short intervals to get it at near real-time, a Webhook provides data to other applications proactively, as its generated. This makes Webhooks much more efficient for both provider and consumer. For Jamura, we proposed a PHP based ThingSpeak Webhook, hosted on Heruko Cloud via Dropbox. Heroku is a free web hosting platform which provides a wide range of language support for deploying web applications whereas Dropbox is a Cloud-based file-sharing application. Another Webhook This Webhook will be requested by Google Dialogflow when a specific event occurs.

C. The Jamura Chatbot Interface

In order to build the Chatbot interface, use of Google's Dialogflow is proposed. Under which each fully functional Chatbot is called as an Agent and such Agents are best described as NLU (Natural Language Understanding) modules in Dialogflow developer documentation [14]. An Agent uses following components:

- ***Intent*** is question-answer pair, where several ways for asking same question and its corresponding expected reply is specified.
- ***Entity*** is a key-value pair, where key is a word and values are its synonyms.
- ***Actions and Parameters*** are used for filtering intents and generating a suitable query format using variables from the query.
- ***Fulfillment*** is used if a query has to be redirected to a Webhook for processing and response.

In addition to Dialogflow, we also propose to use Telegram [15], which is a cloud-hosted desktop and mobile messaging application, with a high emphasis on speed and security. Telegram provides a simple way to create a Chatbot by using a *Botfather*, which in itself is a Chatbot to create Chatbots. However, to provide intelligence for the proposed Chatbot, we would require integration of Dialogflow with Telegram and ThingSpeak. This can be achieved by using ThingSpeak and Telegram Webhooks. Google's Image Processing and Analytics API is integrated for identification of features from the images captured by the camera at the main door to support Jamura's Intruder Detection/Alert System.

III. Experimental Setup & Implementation

A. Experimental Setup

The hardware and software used in the development of Jamura is tabulated in Tables I through IV. In Table I-III, the various hardware components like sensors, actuators and micro-controllers used are presented, along with their respective usage. In Table IV, the various software, technology frameworks and API stacks used for integrating the Web of Things infrastructure of Jamura, along with the cloud services used to integrate and process data from various functional modules is presented.

TABLE I
HARDWARE REQUIREMENTS - TYPES OF SENSORS

S.No.	Component	Model	Details
1	Humidity Sensor	SEN-10167	Monitors room humidity
2	Temperature Sensor	SEN-10167	Monitors room temperature
3	Light Sensor	SEN-09088	Monitors ambient light in the room
4	Soil Moisture Sensor	SEN-13322	Monitors moisture content of garden soil
5	Motion Sensor	SEN-13285	Detects presence of obstacle in front of main door
6	Camera	DEV-14028	Capture photo and videos for analysis

TABLE II
HARDWARE REQUIREMENTS - ACTUATORS

Sl. No.	Component	Model	Justification
1	SMD Lights	COM-11821	Improve room illumination
2	Submersible Water Pump	ROB-10455	Garden watering
3	DC Motors	ROB-11696	Used for simulating AC or Fan in actual smart home scenario
4	Speaker	COM-14023	Used for voice output

B. Jamura's IoT Infrastructure

The implementation of Jamura is divided into two phases - the setup of the IoT ecosystem and implementation of Web of Things and Services. As shown in Fig. 1, there are four *Local Clients*, Room1, Room2, a garden and a main door. For the implementation of Room1 and Room2, we used Light-Dependent Resistor (LDR), Digital Humidity Temperature (DHT11), Surface-Mount Device Light Emitting Diodes (SMD-LED) and WeMos D1 Mini. The connections

TABLE III
HARDWARE REQUIREMENTS - MICROCONTROLLERS

Sl.No.	Component	Model	Justification
1	WeMos D1 Mini	ESP8266	Transmits the sensed data to Local Server and receives input from Local Server
2	Raspberry Pi	3B	Monitors the main door and performs operation as Local Server and Gateway

TABLE IV
SOFTWARE REQUIREMENTS

Sl. No.	Software/ Technology	Usage
1	Python3	Used for implementing the Local Server, REST APIs and for sending image to Telegram
2	Flask SQLAlchemy	For maintaining received data at Local Server in a SQL format
3	Flask Marshmallow	For serializing and deserializing JSON data
5	Arduino IDE	For writing and uploading code over WeMos, for reading and writing data over sensors and actuators respectively
6	Google Cloud Platform	Used to create Chat Bot using Dialogflow and to invoke Vision API for Image Analysis
7	Heroku	Used for deploying the Jamuara Webhook over a global domain
8	PHP	Used to code the Jamura Webhook

are shown in Fig. 2. An Arduino IDE with Adafruit DHT library is used for programming WeMos. An object of DHT class is created based on supplied DHT Type and pin number as initialization parameters. Once initialized, the object can is used to retrieve values of temperature and humidity. The output pin of LDR is connected to A0 pin of WeMos, and the value sensed at A0 is directly read using AnalogRead function of WeMos. The LED is connected to pin D1 of WeMos and which is used for emulating real-world light control. Fig. 3 shows the implementation of Local Client for Room1 and Room2. The sensed data is frequently updated over Local Server and ThingSpeak Cloud.

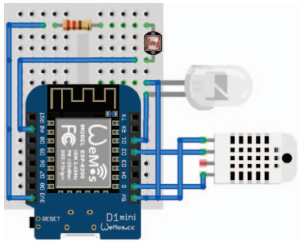

Fig. 2. Circuit Diagram for Sensor-Actuator Module for Room1 and Room2

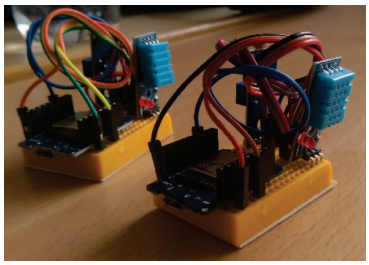

Fig. 3. Room1 and Room2 Local client and Hardware configuration

Similar to Room1 and Room2, the Garden local client is also connected to Local Server. The main objective of this local client is to sense moisture content of the soil, send it to Local Server and turn on Water pump when it drops below a preset threshold. To implement this, we used a Moisture Sensor and Relay Switch, which is connected to WeMos. Since the power required to run the water pump is more than the output power of WeMos, the relay switch is used, as shown in Fig. 4 for providing additional power from an external source. The setup for Local Client Garden is as shown in Fig. 5.

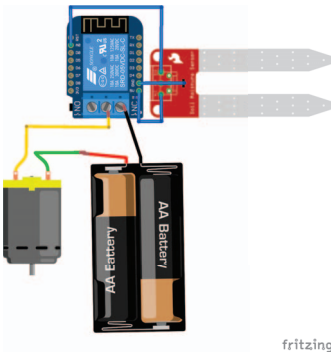

Fig. 4. Circuit Diagram for Garden Monitoring System

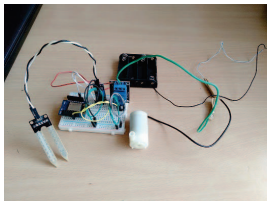

Fig. 5. Garden Monitoring System

As WeMos comes with onboard WiFi ESP8266 module, we used it to connect it to our Local Server (i.e., Raspberry Pi). Each device has a Unique Identifier which is sent during URL request. Each of these Local Clients can send data in JSON format as shown Code 1.

```
{  "channel": {
    "id": 342778,"name": "Room1",
    "description": "Monitors Room1 sensors",
    "field1": "Temperature",
    "field2": "Ambient-Light-Intensity",
```

```
    "field3": "Humidity", "field4": "Lights",
    "created_at": "2018-10-08T09:19:42Z",
    "updated_at": "2018-11-18T09:56:51Z"
  },
  "feeds": [
    {
      "created_at": "2018-11-18T09:56:51Z",
      "field1": "22", "field2": "35",
      "field3": "62", "field4": "0"
    } ] }
```


Code 1. Sensor data in JSON Format

As a final step in the setup, as shown in Fig. 6, a PIR sensor, and Camera is connected to Raspberry Pi (shown in Fig. 7). The PIR sensor continuously checks for any motion near it, and when detected, an event script for clicking a photograph is triggered. Google Vision API is used to extract features from the image and assign labels to the image for identifying the occurrence of features like Nose, Eyes, Arms, Face, Human, Girl, etc. Once this is done, the image is sent to the User's smartphone via Telegram Webhook, for next action/command. Apart from being a Local Client for Main Door, the Raspberry Pi also works as a Local Server. It provides access to custom-defined REST APIs to serve the functions defined for Jamura (described in Section III-C).

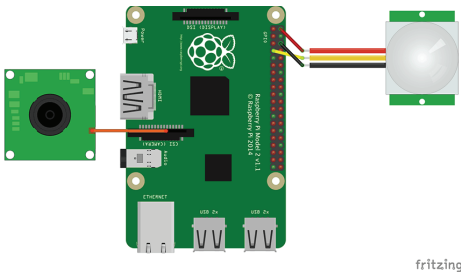

Fig. 6. Circuit diagram for Raspberry Pi Door Module

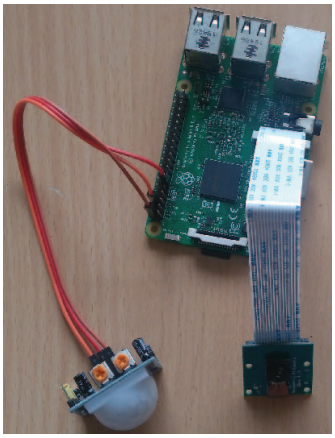

Fig. 7. Local Client (Main Door) / Local Server

C. Jamura's WoT Ecosystem and Services

1) Implementation of REST API: As discussed in Section II (C), the four layers of WoT i.e. Access, Discovery, Share and Compose, are to be provisioned. One of the main requirements for this provisioning is the need for well-formatted data. The defined JSON data format (shown in Code 1) fits well with this requirement, however, to send and receive data in the prescribed format, a lightweight framework is essential. Hence, we used the concept of RESTful Web services on Local Server for supporting a event-driven, asynchronous architecture for Jamura.

The RESTful services were implemented using Flask, Python's SQLAlchemy and Marshmallow deserialization library, that are highly efficient for REST API development. SQLAlchemy is a Python SQL toolkit that provides developers with the complete features and flexibility of SQL, where as, Flask-sqlalchemy is Flask library which extends integration with SQLAlchemy in a Flask application. Flask-marshmallow is a Flask extension that is used to integrate the object serialization / deserialization library with Flask. For Jamura, we used flask-marshmallow to correctly render JSON responses from the various things in the IoT infrastructure. The process of designing the RESTful Service Ecosystem is described below.

- Import Flask to create an instance of the Web application, request for data, jsonify it to generate a JSON output. Convert this into a Response object with a mimetype "application/json", then use SQAlchemy to access the database, and Marshmallow to generate the serialized object.
- Set the path to the defined SQLite URI, and bind SQLAlchemy and Marshmallow to the Flask application.
- Once SQLAlchemy is bound to the Flask app, the models to be used are declared. In the proposed work, a model called 'Thing' is used and its field and properties were defined.
- Define the structure of the response of the created endpoint. All endpoints must support JSON responses. The designed JSON response contains key-value pairs for *time, deviceId, roomName, lightintensity, humidity, temperature* and *lightstatus*.
- We set the route to '/thing' and set the HTTP methods to POST. Next, for the route and methods, we define a function that will be executed when the endpoint is accessed. A new 'thing' object is created using the data present in request and same is added to the database in the form of JSON Object.
- Define an endpoint to get list of all things (i.e. connected devices) and show the result as JSON response.
- Also, define an endpoint to get device data based on device Id.
- Define another endpoint to update a Thing object. The thing object associated with a given id on parameter can be accessed at this endpoint.
- Define an endpoint to delete a thing. The thing associated with that id, can be deleted by sending a JSON request to this endpoint.

2) Webhook Implementation: To redirect the request from Google's Dialogflow to Jamura's Local Server so that data could be fetched, a dedicated Webhook is implemented which is named as ThingSpeak Webhook. Based on the query, Dialogflow can resolve it by using defined *Intents* and *Entities*. If a device accesses *Fulfillment* endpoint with the defined intent then, the request will be forwarded to ThingSpeak Webhook for resolution. From this request, ThingSpeak Webhook will extract value for corresponding action. Depending on the action, a POST request will be sent to Jamura's Local Server in JSON format. Following is a sample JSON request generated by Dialogflow:

```
{ "result": {
    "source": "agent",
    "resolvedQuery": "What is temperature
        in Room1",
    "action": "sensorbyroom",
    "parameters": {"room": "Room1","
       sensor": "temperature"},
    "fulfillment": {
      "speech": "The temperature in Room1
          is 22",
      "source": "Webhook"},},}
```

Code 2. Dialogflow request message

Based on variable values, respective data from corresponding device environment will be fetched. Further, by assigning the fetched values to desired variables, a JSON response is generated and sent back to Dialogflow. For example, as shown in Code 2, the query is *"What is temperature in Room1?"*. Based on type of query the resolved *action* and *parameters* is identified. As shown in Code 2, *action* is "sensorbyroom" and *parameters* are "room" and "sensor". Value for "room" and "sensor" is "Room1" and "temperature" respectively. Based on room name, specific sensor/actuator will be selected and sensed value will be returned. Fig. 8 shows implementation the process of query resolution and processing of ThingSpeak Webhook.

3) Using Google Dialogflow and Telegram: Dialogflow provides an interactive platform for creating a set of valid user queries and their expected response. It uses its native NLP and Machine Learning capabilities to understand the query if the requested query does not exactly match. This provides flexibility for users to query with minimum inputs. From the given query, Dialogflow recognizes the entities that are referred to in the query, and it identifies the corresponding action that has to be triggered. Based on the action, respective entities will be sent to the Webhook for serving the request.

Dialogflow provides an option for deploying the Chatbot over Telegram. For this, we need to create a bot on Telegram using Telegram *"Bot Father"*. It will create a simple Chatbot based on a predefined template and provide unique TokenID. The provided TokenID is passed to Dialogflow, which then obtains a Webhook to a Telegram bot. This will enable the

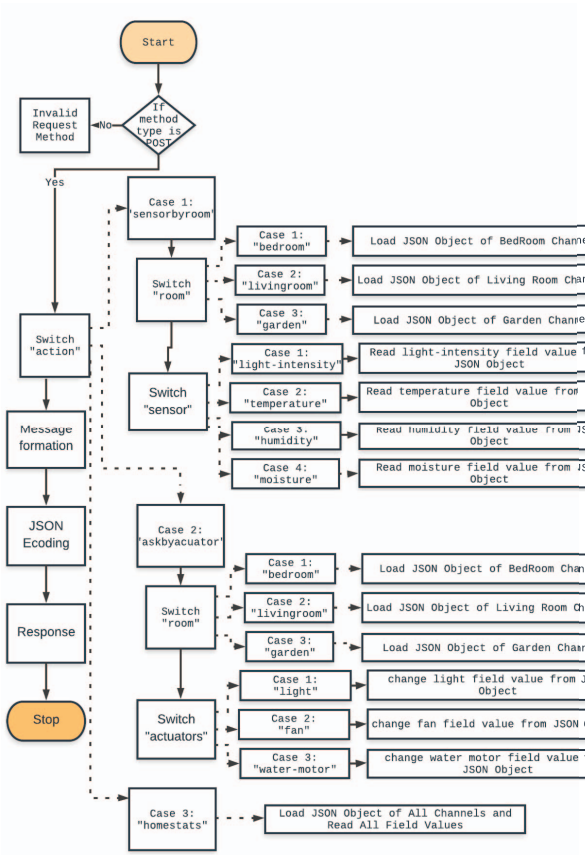

Fig. 8. Webhook Flowchart

user to use Telegram for requesting any required details with respect to the deployed sensor in the house. However, once the Telegram Webhook is obtained by Dialogflow, it cannot be simultaneously used for any other purpose. For example, if a bot initiated message has to be sent to the user. Thus, for our proposed system, we have used 2 Chatbots, one for interacting with sensors and other exclusively to send bot initiated messages based on an assigned trigger (i.e., Intruder Detection/Alert System).

IV. Jamura in Action - Results & Observations

In this section, we showcase the functionalities and capabilities of the designed Smart Home Conversational Agent, Jamura. Various User Interfaces of Jamura are shown in Fig. 9. In Fig. 9(a), Jamura has been integrated with Google Assistant. This provides user to use set of voice or text commands to interact with Jamura. Fig. 9(b) shows Jamura as an Iframe web component which can be easily integrated into any custom made websites. Moreover, Fig. 9(c) shows user interaction with Jamura over Telegram Messaging Platform.

Fig. 10 illustrates the process of interaction between the user and a visitor at the main door. The Smart Door System is initiated when the presence of any object/human is sensed by the PIR motion sensor associated with the main door. This triggers the camera module and captures the object/human's

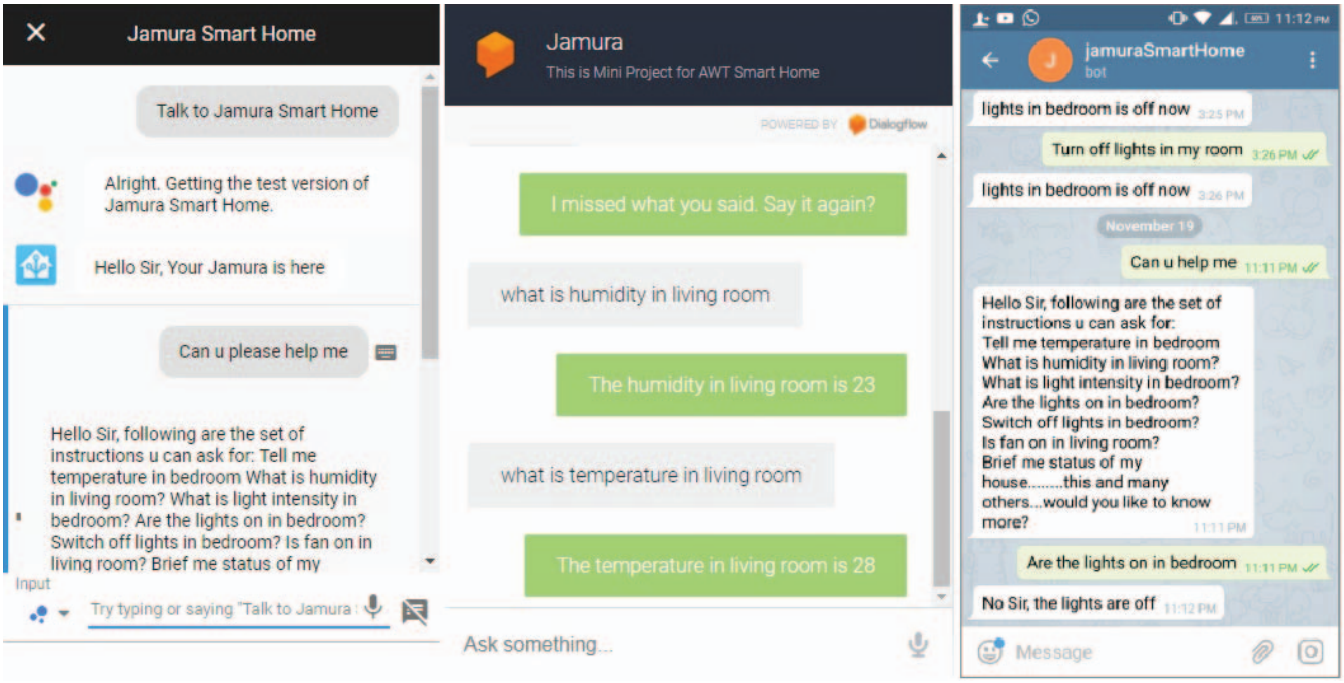

Fig. 9. a) Jamura Google Assistant Interface, b) Jamura Iframe Interface, c) Jamura Telegram Interface

photo, which is then transferred to Google Vision Service for identification of the presence of a human in the photo captured. If a human is present in the captured photo, then the photo is sent to the homeowner, requesting further action. If the homeowner replies within a timeframe of 30secs, then Jamura provides the message as an audio output to the visitor waiting outside the door. Otherwise, a default message saying "*Sorry, Owner did not respond*" is returned to the visitor.

To evaluate the performance of Jamura in real deployment scenarios, few tests for observing the response time were conducted. Variety of *intents* which represent the possible set of queries that a human can send to the agent were defined. Table V shows the observed response time of agent with respect to different intents. Here, *Session* represents the number of distinct sessions for which corresponding *intent* was called and *count* represents total occurrences of particular *intent* during testing. It is observed that queries which required Webhook for generating response took more time when compared to queries which were served directly through Dialogflow. This is the effect of the additional latency incurred during communication from Dialogflow to Webhook, Webhook to Cloud, processing of received data from cloud and finally, generated response to Dialogflow. Fig. 11 is obtained from Dialogflow's Analytics section which shows overall performance with respect to requested queries and accuracy of response. It is seen that the majority of requested queries where handled by *Default Welcome Intent*. This intent also had about 50% of the total *exits*, which implies that 50% of total users terminated their session under this intent. However, 12.5% of total queries went ahead with further queries which were resolved by *Who are you* intent. Also, it can be seen that 25% of the queries were handeled by *Brief Home Stats* intent and remaining 12.5% and 12.5% were handled by *What can I do for you?* intent and *AskHome* intent.

Fig. 10. Conversation with Jamura Smart Door

TABLE V
JAMURA'S RESPONSE TIME EVALUATION

Sl.no	Intent	Sessions	Count	Median Response time	90% response time
1	Welcome Intent	21	31	0.04s	3.08s
2	AskHome2	14	19	0.37s	10.12s
3	Quit	12	15	0.06s	0.12s
4	What can I do	11	13	0.38s	0.39s
5	My name is	14	15	0.08s	0.12s
6	Turn On/Off	9	11	0.34s	0.34s
7	Who are you	14	16	0.04s	0.04s
8	Which rooms	13	15	7.42s	7.42s
9	Tell me sensor	11	13	0.08s	0.08s
10	Brief me about Home Stats	9	10	0.28s	0.28s

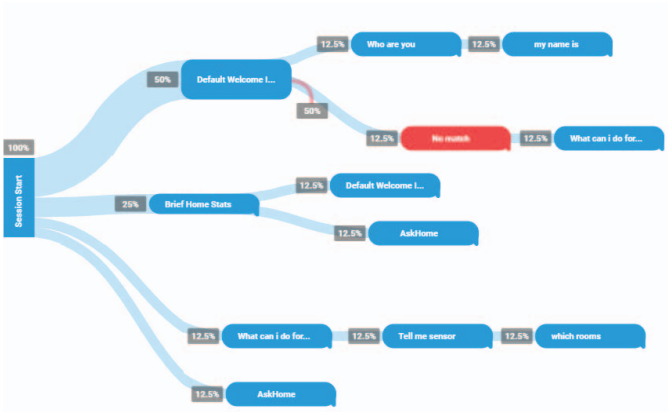

Fig. 11. Percentage Requested Intent and Accuracy

V. CONCLUSION AND FUTURE SCOPE

In this paper, an intelligent conversational agent that is built based on a robust framework using Telegram and Google DialogFlow was presented. These customized Chatbots for IoT applications are built on the basis of open source frameworks, this enables the provisioning of addressable IoT resources using Web of Things concepts. It was observed that implemented system works well under restricted set of queries. It is mainly due to limited training queries and unexpected queries. It was also observed that due to hard-coded user-id the smart door Chatbot responds only to authorized user. Jamura provides combined benefits of similar systems proposed under [16] [17] like Speech Recognition, Context Recognition, Remote Monitoring and Control, Easy-to-use Interface. In addition, our system also provides support for integration with various other platform as shown in Fig.9

As part of future work, we intend to incorporate data analytics and optimization techniques for enhancing the learning skills of Jamura to better suit the needs of the Smart-Home owner.

ACKNOWLEDGEMENT

This publication is an outcome of the R&D work undertaken in the project under the Visvesvaraya Ph.D. Scheme of Ministry of Electronics & Information Technology, Government of India, being implemented by Digital India Corporation (formerly Media Lab Asia).

REFERENCES

[1] N. M. Radziwill and M. C. Benton, "Evaluating Quality of Chatbots and Intelligent Conversational Agents," *arXiv e-prints*, p. arXiv:1704.04579, Apr. 2017.

[2] D. Guinard, V. Trifa, F. Mattern, and E. Wilde, *From the Internet of Things to the Web of Things: Resource-oriented Architecture and Best Practices*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 97–129.

[3] R. Milton, D. Hay, S. Gray, B. Buyuklieva, and A. Hudson-Smith, "Smart iot and soft ai," in *Living in the Internet of Things: Cybersecurity of the IoT - 2018*, March 2018, pp. 1–6.

[4] T. Obaid, H. Rashed, A. Abu El Nour, M. Rehan, M. Muhammad Saleh, and M. Tarique, "Zigbee based voice controlled wireless smart home system," *International Journal of Wireless and Mobile Networks*, vol. 6, 02 2014.

[5] D. Guinard, V. Trifa, and E. Wilde, "Architecting a mashable open world wide web of things," Tech. Rep., 2010-02, .

[6] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[7] L. Richardson and S. Ruby, *Restful Web Services*, 1st ed. O'Reilly, 2007.

[8] A. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and Q. Z. Sheng, "Iot middleware: A survey on issues and enabling technologies," *IEEE Internet of Things Journal*, vol. 4, no. 1, pp. 1–20, Feb 2017.

[9] D. Guinard, M. Fischer, and V. Trifa, "Sharing using social networks in a composable web of things," in *2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, March 2010, pp. 702–707.

[10] M. Santos-Pérez, E. González-Parada, and J. M. Cano-García, "Avatar: An open source architecture for embodied conversational agents in smart environments," in *Ambient Assisted Living*, J. Bravo, R. Hervás, and V. Villarreal, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 109–115.

[11] R. G. Anvekar, R. M. Banakar, and R. R. Bhat, "Design alternatives for end user communication in iot based system model," in *2017 IEEE Technological Innovations in ICT for Agriculture and Rural Development (TIAR)*, April 2017, pp. 121–125.

[12] A. Patel and A. Verma, "Iot based facial recognition door access control home security system," *International Journal of Computer Applications*, vol. 172, pp. 11–17, 08 2017.

[13] Z. Che Soh, M. S. Shafie, M. Affandi Shafie, S. Sulaiman, M. Nizam Ibrahim, and S. A. Che Abdullah, "Iot water consumption monitoring and alert system," Banda Aceh, Indonesia, 09 2018, pp. 168–172.

[14] "Agents overview for chatbot creation using dialogflow [online]," https://dialogflow.com/docs/agents, accessed: 23-Jan-2019.

[15] "bots: An introduction for developers, telegram apis. [online]," https://core.telegram.org/bots, accessed: 23-Jan-2019.

[16] T. Parthornratt, D. Kitsawat, P. Putthapipat, and P. Koronjaruwat, "A smart home automation via facebook chatbot and raspberry pi," in *2018 2nd International Conference on Engineering Innovation (ICEI)*, July 2018, pp. 52–56.

[17] C. J. Baby, F. A. Khan, and J. N. Swathi, "Home automation using iot and a chatbot using natural language processing," in *2017 Innovations in Power and Advanced Computing Technologies (i-PACT)*, April 2017, pp. 1–6.