State Machine based Human-Bot Conversation Model and Services

Shayan Zamanirad, Boualem Benatallah, Carlos Rodriguez, Mohammadali Yaghoubzadehfard, Sara Bouguelia, Hayet Brabra

State Machine based Human-Bot Conversation Model and Services

Shayan Zamanirad[1], Boualem Benatallah[1,2], Carlos Rodriguez[1], Mohammadali Yaghoubzadehfard[1], Sara Bouguelia[2], and Hayet Brabra[2]

[1] University of New South Wales (UNSW), Sydney Australia
{shayanz, boualem, crodriguez, m.yaghoubzadehfard}@cse.unsw.edu.au
[2] LIRIS – University of Claude Bernard Lyon 1, Villeurbanne, France
{sara.bouguelia,hayet.brabra}@univ-lyon1.fr

Abstract. Task-oriented virtual assistants (or simply *chatbots*) are in very high demand these days. They employ third-party APIs to serve end-users via natural language interactions. Chatbots are famed for their easy-to-use interface and gentle learning curve (it only requires one of humans' most innate ability, the use of *natural language*). Studies on human conversation patterns show, however, that day-to-day dialogues are of *multi-turn* and *multi-intent* nature, which pushes the need for chatbots that are more resilient and flexible to this style of conversations. In this paper, we propose the idea of leveraging Conversational State Machine to make it a core part of chatbots' *conversation engine* by formulating conversations as a sequence of *states*. Here, each state covers an *intent* and contains a nested state machine to help manage tasks associated to the conversation *intent*. Such enhanced *conversation engine*, together with a novel technique to spot implicit information from dialogues (by exploiting *Dialog Acts*), allows chatbots to manage tangled conversation situations where most existing chatbot technologies fail.

Keywords: Conversational Chatbot · State Machine · Natural Language Processing · REST API.

1 Introduction

Messaging bots, software robots, and virtual assistants (hereafter for simplicity called *chatbots*), are used by millions of people every day [9]. Applications such as Siri, Google Now, Amazon Alexa, Baidu and Cortana have a presence in our living rooms and are with us all the time. New chatbots are developed continuously, from those providing psychological counseling to task-oriented chatbots that help book flights and hotels. They use human-friendly interfaces, using natural language (e.g., text or voice), to access complex cognitive backend, which tries to understand user needs and serve them by invoking the proper services. Despite the interest and usage of chatbots, their interactions with users are still in primitive stage.

Studies on human-chatbot conversation patterns (e.g. [21]) reveal that, in practice, conversations are *multi-turn*, where there may exist missing information (e.g. "location") in users' utterances (e.g. *"what will the weather be like tomorrow?"*) that needs to be fulfilled by the chatbot before an actual API call be invoked. Other examples include an invocation of an API by the chatbot to resolve the value of a missing parameter, a question by a chatbot to a user to confirm an inferred intent value or make a choice among several options, extracting an intent parameter value from the history of user and chatbot interactions. In addition, according to studies on human-chatbot dialogue patterns (e.g. [25]), switching between different intents is a natural behaviour for users. Thus, there is a need for more dynamic and rich abstractions to represent and reason about multi-turn and multi-intent conversational patterns. The main challenge of achieving this objective arise from variations in open-end interactions and the large space of APIs that are potentially unknown to developers.

In this paper, we propose a multi-turn and multi-intent conversational model that leverages Hierarchical State Machines (HSMs) [7][27]. HSMs are a well-known model suited to describing reactive behaviours, which are very relevant for conversations but other specific users-bot-API conversation behaviours must be modelled too. More specifically, HSMs reduce complexity that may be caused by the number of states that are needed to specify interactions between users, chatbots and services.

In this approach, conversations are represented as a sequence of *states* each covering an *intent*. A state relies on a *nested state machine* to manage required tasks towards handling an *intent* to completion. Transitions between states are triggered when certain conditions are satisfied (e.g., detection of new intent, detection of missing required parameter). The proposed *conversational model and engine*, together with new techniques to identify implicit information from dialogues (by exploiting *Dialog Acts* [25]), enable chatbots to manage tangled and multi-turn conversational situations. In summary, our contribution is three-folded:

- We propose the concept of conversation state machines as an abstraction to represent and reason about dialog patterns. Conversational state machines represent multi-turn and multi-intent conversations where state represent intents, their parameters and actions to realise them. Transitions automatically trigger actions to perform desired intent fulfilment operations. The proposed model extends hierarchical state machine model, to effectively support complex user intents through conversations among users, chatbots services and API invocations.
- We propose a *dialog act* recognition technique to identify state transition conditions. We use dialog acts to specify interaction styles between users, chatbots and APIs (e.g., user submit utterance, chatbot detect missing slot value, chatbot ask user to provide missing slot value, user submit a new utterance to supply missing value).
- We develop a conversation management engine that is used to initiate, monitor and control the run-time interactions between users, chatbots and APIs.

The knowledge required at runtime by the conversation management engine is extracted from chatbot specification (i.e,, developer supplied user intents) and user utterances. In this way, the conversation manager automates the generation of run-time nested conversation state machines that are used to deploy, monitor and control conversations with respect to user intents and utterances.

2 Related Work

Conversational Bots. Bots are computer programs that provide natural language conversations between users and software systems. The input of such systems is natural language utterances (text/voice). The system also generates an appropriate response (in form of text/voice) back to the user. Bots are generally categorized into two classes [5][15]: (i) Non-task oriented, and (ii) Task oriented.

Non-task oriented bots focus on open domain conversations with users (i.e, non predefined goal of conversations). Examples for this type of bots include DBpediabot [1], Cleverbot[3] and Mitsuku[4]. This type of bots handle open-domain conversations and hardly keep track of conversation states and are therefore not designed to perform specific user tasks (e.g., task management).

Task-oriented bots, on the other hand, allow users to accomplish a goal (e.g. maintain schedules [6]) using information provided by users during conversations. Since the focus of this paper is on task-oriented bots, the word *"chatbot"* refers to this type of bots for simplicity. Task-oriented bots are classified into two categories [5]: *pipeline* and *end-to-end*. A *pipeline-based* chatbot is built with a set of components, each responsible for a specific task [5]. Research in this area mainly focuses on such tasks, including user intents classification [26], finding slot/value pairs [8] and controlling dialog states [9]. Interested readers are referred to [10] for a comprehensive discussion. On the other hand, *end-to-end* chatbots leverage the idea of generative models and apply neural-based approaches [18] to train and build a model that can decide what the next system action should be by looking at the dialog history [24]. Such chatbots take in user utterances (as sequences of input tokens [10]) and generates actions (such as API invocations, database queries) as sequences of output tokens. Research in this context includes the work by Rastogi et al. Li et al. [18] tackled the problem of building an end-to-end model by considering it as a task completion system, where its final goal is to complete a task (e.g. booking ticket), Furthermore, using memory networks [28], and query reduction networks [23] are other approaches that have been proposed to tackle the challenge of having end-to-end conversational chatbots.

Dialogue Management. Controlling the conversation flow, known as *dialogue management*, is one of the key tasks in conversational chatbots. Dialogue management includes keep tracking of information that is entered implicitly or ex-

[3] https://www.cleverbot.com/
[4] https://www.pandorabots.com/mitsuku/

1. **User** *Book a table at Time for Thai please*
2. **Chatbot** *What is the date?*
3. **User** *hmmm... never mind! Do I have any appointment on Saturday*
4. **Chatbot** *I cannot see anything on your calendar, you look free for Saturday.*
5. **User** *Ok, thanks!*

Fig. 1. User changes the intent to know about her calendar schedule

plicitly by users, managing complex interactions with users, and choosing appropriate actions based on the history of interactions [13]. Research in this context includes the work by Lopez et al. [19], who leveraged the concept of workflows by proposing a system that takes a business process model and generates a list of dialog management rules to deploy/run the chatbot.

Henderson et al. [11] formalized interactions as hidden states with random sequences and transition probabilities using Markov decision processes (MDP) trained on example conversations. More advanced techniques that build on NLP provide rule- and template-based conversations for data science tasks [14], pattern-matching over context-aware conversations for DevOps processes [2], nested and sequence conversations to accomplish complex data science tasks [9], integration of state machines and re-enforcement learning for dialog optimization [7], state and slot tracking during conversations [16]. However, these efforts do not focus on augmenting conversations with knowledge that is essential for the superimposition natural language interactions over large number of evolving APIs.

We identified two main limitations in the works above: First, they heavily rely on the availability of massive amounts of annotated data, structured knowledge base and conversation data. Collecting domain-specific dialog data is laborious [26] and hinders scalability in the context of larger and multi-domain systems. Second, existing end-to-end systems are hard to trace: They can be considered "black-boxes" that accept user utterances in input and return new system states/actions as output. Since in this paper we are addressing the issue of supporting *multi-intent* conversations in chatbots, the use of existing probabilistic approaches such as memory networks and MDPs becomes prohibitive due to the need for collecting huge training datasets for the intents that the chatbot aims to support. We therefore opt for pipeline-based chatbots, built with a set of key components designed to perform specific tasks (e.g. intent recognition). As we will discuss in the following sections, the main difference between our approach and existing pipeline-based ones is the clear separation between the *conversation logic* and *actual implementation* of such logic. At the center of our approach, we utilize the concept of HSMs [27][7] by formulating user-chatbot interaction as a sequence of states. Such abstraction helps bot developers seamlessly define/choose user intent(s) they would like their chatbot to support.

3 Human-Chatbot Conversations

A conversation between user and chatbot can be formulated as a sequence of utterances. For example, to answer user utterance *"Please remind @Sarah that we*

have meeting tomorrow by 1:30PM", after performing the task (e.g. reminder) chatbot replies with, e.g., *"Ok, reminder sent to Sarah"*. Studies on human conversation patterns [12][20][22] reveal that human-chatbot dialogue can be divided into three categories:

- ***Single Intent*[5] *- Single Turn***: Interaction between user and chatbot is in the form of <*Question, Answer*> pairs. The assumption here is that user provides all the required information (e.g. slots[6]/values) at once, in one single utterance [30]. Thus, each utterance from user (e.g. *"Please text Bob that we are in meeting room 401K"*) has a reply from chatbot (*"'We are in meeting room 401K' is sent to Bob"*). This type of conversation is *stateless*, i.e, each user utterance is treated separately without using any knowledge from past conversations or context. Thus, if any information is missing in user utterance (e.g location, date), chatbot is not able to perform the required task (e.g. *book a flight ticket*)). Finally, in this type of conversation, parties talk about only about one specific intent (e.g. *schedule meetings* [6]) during the whole conversation.
- ***Single Intent - Multi Turn***: Providing missing information (e.g. location, date) to generate a complete intent is a common behaviour that people follow in their daily conversations [12][15]. For example, while talking to a friend on the phone we may ask, "I'm going to have lunch, do you you have any suggestion?" to get some ideas for lunch. However, without specifying the place where we are at (e.g. "UNSW") or our preference for today (e.g. "Thai food", "Sandwich", "noodle"), our friend is less likely able to give us concrete suggestions. Thus, she asks questions to get more details (e.g. "Where are you?", "What do you prefer", "Do you like something soupy?"). Similar to this example, information (e.g. "departureDate", "destinationCity") that chatbot needs to perform a task (e.g. *call Expedia API to book a flight ticket*) is scattered across multiple user utterances.
- ***Multi Intent - Multi Turn***: In this type of conversation, the intent continuously changes. Figure 1 exemplifies a dialogue where user changes the intent by asking about *"her appointment on the weekend"*. Changing intent is something usual that people do in their day-to-day conversations [12]. However, participating in a multi-intent conversation where conversation information is scattered into multiple utterances, is a challenging task for chatbots [24]. Such difficulty stems mainly from the challenges of identifying user intent changes [21], and tracking slots information for each intent in a conversation. In the following sections, we explain our approach to empower chatbots in handling this type of conversations by utilising Hierarchical State Machines (HSMs).

[5] An *intent* refers to users' purposes, which a chatbot should be able to respond to (e.g. *"find restaurant"* or *"book table"*).

[6] A *slot* is an important information that is necessary for a chatbot to understand in order to be able to serve the correct answer (e.g. location, data, time)

4 Conversation State Machines

We propose to represent User-Chatbot-API conversations using an extended hierarchical state machine model. In this model, a state machine contains a set of states representing user intents. We call these states, "intent-states". An intent-state characterizes the fulfillment of specific user intent. In the following, we describe different types of states and transitions between states:

Basic Intent State: When user utterance carries all the required information to fulfill the user intent, chatbot does not need to communicate with user to get any further information. We call this state a *basic* intent state, where chatbot has everything needed to perform required action (e.g., API call). For example, given a user utterance (e.g. *"What is the weather forecast in Sydney?"*) with an intent (e.g. "GetWeather"), chatbot invokes an API (e.g. *OpenWeatherMap* to get weather condition) and returns a message to user (e.g. *"We have Scattered showers in Sydney."*). The interaction between user and chatbot is straight forward, without any further question from chatbot.

Nested State: If user utterance has missing information, then the chatbot needs to communicate with user to get missing values before it perform further actions to fullfill the intent. In this situation, the intent-state relies on other nested states [9] to complete the intent. More specifically, a *nested state* is used by chatbot to ask user for missing values of intent slots. Based on user response a nested state is divided into two categories: (i) *nested slot-value state*, and (ii) *nested slot-intent state*.

Nested slot-value state represents a situation where user explicitly provides "value" for the slot-filling question asked by chatbot. For example, in *restaurant booking* chatbot, given the chatbot question *"What is the date?"* (to fulfill *bookingDate* slot), user answers with *"This Sunday"*. This answer does not require any further processing as it provides the missing slot (e.g. *bookingDate*) value.

Nested slot-intent state represents a situation where user does not provide slot "value" directly, but provides an utterance with a new request (i.e, new intent) that the chatbot needs to process to obtain the missing value. For example, considering the *restaurant booking* chatbot, given the chatbot question *"What is the date?"*, user replies with *"Which day of the weekend am I free?"*. The answer from the user is another utterance whose processing identifies another intent which is represented by another intent-state e.g. "CheckCalendar". In order to obtain the slot value in this case, the parent intent-state ("BookRestaurant") triggers a transition to a nested slot-intent state ("CheckCalendar").

4.1 Transitions between States

Transitions between states are triggered when actions are performed (e.g., asking a clarification question to a user to resolve an intent parameter value) or upon the detecting intent switch in conversations (i.e, detecting a new intent). We identify three types of transitions: (i) *new intent*, (ii) *nested slot.value*, and (iii) *nested slot.intent* transitions.

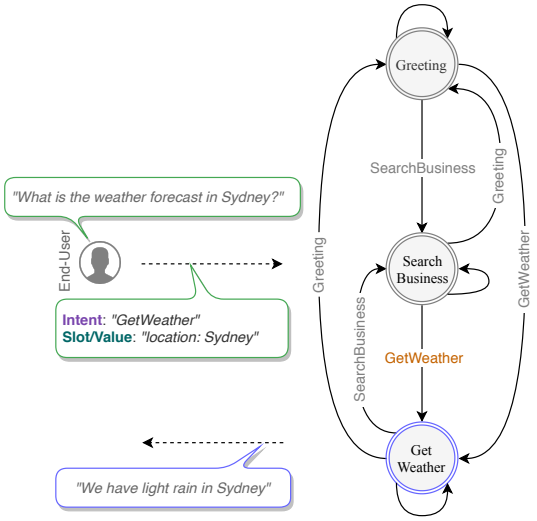

Fig. 2. Transition between intent-states based on user intent - current intent-state is denoted by *blue* color, "new intent" transition is highlighted in *orange*

New intent transitions refer to the movements between intent-states. The state machine transits to an a new intent-state if the processing of a new user utterance identifies a new intent (an intent is that is not handled by the current conversation state). Figure 2 shows an example of *"new intent"* transition between two intent-states (*SearchBusiness, GetWeather*). For example, assuming that state machine is in "Greeting" intent-state, user asks for restaurant suggestions. User utterance i.e. *"Any Italian restaurant near Kingsford"* triggers a transition to move from the state "Greeting" (current intent-state) to the state "SearchBusiness" (new intent-state). Then after, user utters another request, e.g. *"What is the weather forecast in Sydney?"*). This new user utterance has a different intent (e.g. "GetWeather"). Thus, it triggers a transition to move from "SearchBusiness" to "GetWeather" intent-state (blue colored state in Figure 2).

Nested slot.value transition represents the movement of state machine to nested slot-value state. The state machine moves to a slot-value state if user provides a "value" for the missing slot upon a bot request for such value. Figure 3 shows an example of *"nested slot.value"* transition within "GetWeather" intent-state. For example, to fill a missing slot (e.g. *location*), state machine moves from parent state "GetWeather" (current intent-state) to the nested slot-value state "location" (depicted with red colour in Figure 3) where chatbot asks user to provide information (e.g. *"Where are you?"*). User replies with a value (e.g. "I'm in Sydney"). The state machine moves back from nested slot-value state "location" to the parent state "GetWeather" to continue the conversation with user.

Nested slot.intent transitions indicate the movements of state machine to nested slot-intent states. For example, to fill a missing slot (e.g. *location*), chatbot

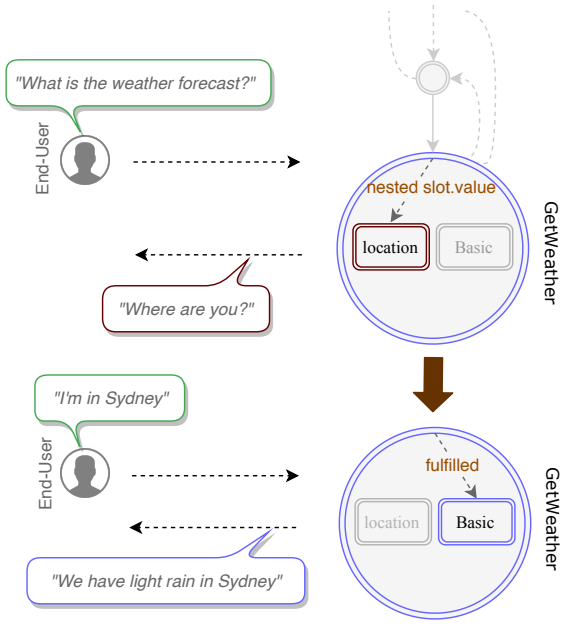

Fig. 3. Transition to nested slot-value state denoted by red color

asks user to provide information (e.g. *"Where are you?"*). User's answer (e.g. *"Where is my home town?"*) carries another intent (e.g. "GetUserDetails") to obtain the missing value. This new intent is handled by a nested state in which the value of the missing value is obtained (e.g., by invoking an API).

5 Generating State Machines

We devise *"State Machine Generater"* (SMG), a service that is used to generate a state machine that allows chatbot to manage conversations at run-time. SMG takes as input (i) user utterances, and (ii) bot specification which is a set of intents (e.g. *Greeting, SearchBusiness, GetWeather, GoodBye*). In the following, we explain the steps taken by SMG to generate a state machine.

5.1 Generating Intent States from Bot Specification

When SMG receives a bot specification (i.e, a set of user intents), it creates an intent-state per user intent. For example, an intent-state, namely *SearchBusiness* state is created to represents the user intent "SearchBusiness" (i.e, get list of restaurants and cafes).

SMG creates intent-states for two types of user intents. The first type of user intent is general communication intent (e.g. *Greeting, GoodBye*). These intents are fullfiled using *(question, answer)* pairs that do not require any API

Table 1. Examples of *Dialog Acts* in a conversation between user and chatbot.

User	**Chatbot**
Is there any Italian restaurant around? [*New Intent*]	Where are you? [*Request Information*]
I'm in Kingsford. [*Provide Information*]	I found *Mamma Teresa* in "412 Anzac Pde..." [*Provide Information*]

invocation. The second type of user intent requires the invocation of an API method to be completed. For example, in "GetWeather" user intent, the chatbot needs to invoke *OpenWeatherMap* API to retrieve weather conditions and fulfills user intent.

5.2 Generating Transitions between States

At run time, the bot generates three types of transitions namely "new intent state", "intent state to nested slot.value state", "intent state to nested slot.intent state". To generate these transitions, we leverage dialog acts [25]. In this section, we first describe dialog acts and then we explain how SMG generates transitions using dialog acts.

Dialog Acts Understanding user needs and engaging in natural language conversations requires chatbot to identify hidden actions in user utterances, called *dialog acts*. Whether the user is making a statement, or asking a question, or negotiating on suggestions, are all hidden acts in user utterances [4].

In a nutshell, dialog acts convey the meaning of utterances at the level of illocutionary force [15]. For instance, 42 dialog acts were identified in [25]. Inspired by this work and empirical studies on human-chatbot conversations [12], we adapted dialog acts to the requirements of *multi intent - multi turn* chatbots that leverage APIs. Table 1 shows examples of these dialog acts.

More specifically, we focus on the following dialog acts:

- ***U-New Intent***: this act indicates that user has a new intent. For example, when user says *"Which day of this weekend am I free?"*), her intention is to know about her availability time for the weekend (intent is e.g. *CheckCalendar*).
- ***C-Request Information***: This act indicates that chatbot asks user to provide missing slot value. For example, chatbot asks the user (e.g. *"Where are you?"*) to provide her location.
- ***U-Provide Information***: This act indicates that user provides an information (e.g. *"15 March"*) for a former question asked by the chatbot (e.g. *"What is the date?"*).
- ***U-Provide Nested Intent***: This act indicates that user provides utterance (e.g. *"When is my birthday?"*) to answer a former question asked by chatbot (e.g. *"What is the date?"*). The completion of this utterance requires transition to a nested intent state.

– ***C-Provide Information***: This act indicates that chatbot replies to user request by providing an answer. For example, chatbot answer (e.g. *"I found Mamma Teresa, it's in 412 Anzac Parade..."*) to the question asked by user (e.g. *"Is there an Italian restaurant around?"*).

Generating Transitions. We annotate user-chatbot conversation messages using dialog acts. Thus, sequences of dialog acts (e.g. *<C-request info, U-provides info, ...>*) can be inferred from conversations. We call these sequences dialog act patterns. Dialog act patterns are used by the SMG to generate state transitions.

New Intent-State - The SMG generates this type of transition upon identifying the dialog act patterns:

– *<U-new intent>*: It describes a situation where user starts a conversation by uttering a request (e.g. *What is the weather forecast for Sydney today?*) which is annotated with *U-New Intent* dialog act. This triggers a "new intent-state" transition from "Greeting" (current intent-state) to the "GetWeather" intent-state (as shown in Figure 2 with blue color).
– *<..., C-provide info, U-new intent>*: It represents a situation where user utters another request (annotated with *U-New Intent* dialog act) right after an answer from chatbot. The chatbot answer is related to a request asked by user in previous conversation turns. For example, when chatbot answers user request with e.g. *"The weather in Sydney is sunny today"*, the user asks a new utterance i.e. *"I want to drink slushy, is there any McDonald's around?"*. The new user utterance carries a new intent (e.g. *SearchRestaurant*). This triggers a "new intent-state" transition from "GetWeather" intent-state to "SearchRestaurant" intent-state.

Intent state to nested slot.value state - The SMG generates this type of transition upon identifying the following pattern: *<..., C-request info, U-provide info>*. This pattern describes a situation where user utters a request with missing information that the chatbot needs before it can fulfills the user intent. Thus, chatbot asks user to provide the missing information. In this case the user answer provides the missing value. Figure 3 shows an example of *"nested slot.value"* transition within "GetWeather" intent-state. For example, when user asks for weather condition (*"What is the weather forecast?"*), a "nested slot.value" transition from "GetWeather" intent-state (current state) to "location" nested slot-value state is created. Chatbot then asks *"Where are you?"* and user replies with *"I'm in Sydney"*. The state machine then goes back to the parent intent-state "GetWeather".

Nested slot.value state to nested slot.intent state - SMG generates this type of transition upon identifying the following pattern: *<..., C-request info, U-provide nested intent>*. This pattern describes a situation where a chatbot asks user to provide a value for a missing slot value (e.g. *location*). The user answers with another request with an intent to compute this value using another service. For example, when chatbot asks *"Where are you?"*, the user answers with the utterance *"Where is my home town?"* which is annotated with *U-Provide Nested*

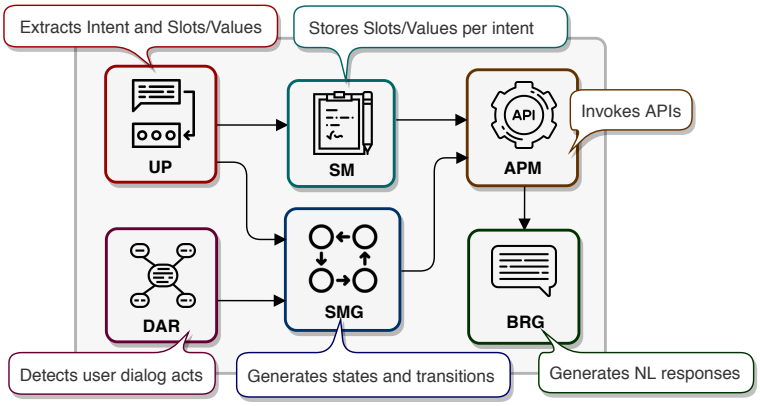

Fig. 4. Conversation Manager Architecture

Intent dialog act. This pattern triggers a "nested slot.intent" transition from "location" nested slot-value state (current state) to "GetUserDetails" nested intent-state. In this state, chatbot invokes an API to get the "user home town". The result (e.g. "Sydney") is the value for missing slot (e.g. location).

6 Conversation Manager Service

In order to support *multi-intent and multi-turn* conversations, chatbots need we devise a service that initiates, monitors and controls conversations. This service is called *conversation manager*. It utilises a set of components to communicate with users, manages the hierarchical state machine, and invoke APIs.

6.1 Conversation Manager Architecture

Figure 4 shows the architecture of conversation manager service. In terms of software architecture, the *conversation manager* relies on the following components namely *Utterance Parser (UP), Dialog Act Recogniser (DAR), State Machine Generator (SMG), Slot Memory (SM), API Manager (APM)* and *Bot Response Generator* (BRG). While *UP, APM* and *BRG* are general components that exist in every chatbot and we refer interested readers to [13][5] for details about such components. In this section, we describe *DAR, SMG, and SM* implementation details.

6.2 Dialog Act Recogniser

DAR classifies user utterance into a corresponding dialogue act class. It can use any classification model such as Naive Bayes, MaxEntropy and Support Vector Machine (SVM). In the current implementation, DAR uses a Bi-LSTM classifier [17] trained on Switchboard Dialogue Act Corpus[7] which contains 1155 human to human conversations with dialog act annotations.

[7] https://web.stanford.edu/jurafsky/swb1_dialogact_annot.tar.gz

6.3 State Machine Generator

SMG leverages pyTransitions[8], an off-the-shelf python library to generate state machines. To generate intent-states for general user intents, SMG initialises the *Machine* class from the library with intent-state (e.g. *Greeting, GoodBye*) along with an initial state (e.g. Greeting). To generate intent-states for intents with API invocations, SMG uses an extension module in the library. It imports *NestedState* class from the library with initialization arguments as "name" and "children". "name" refers to the name of intent (e.g. *SearchBusiness*) and "children" refers to required slots of the API (e.g. *type, location*). To generate transitions, SMG uses the "add_transition" operation in *Machine* class of pyTransitions library. A transition is generated by passing the "source" (e.g. *GetWeather*) and "destination" (e.g. *SearchBusiness*) states as arguments to the operation.

6.4 Slot Memory Service

"Remembering" the information that user provides in each turn is an essential feature for chatbots. This feature is indispensable when it comes to *multi-intent* conversations, where conversations involve several intents and slot values might be missing in some conversation turns [9]. Having a component that helps recall such information from user utterances throughout the conversation is therefore necessary. This is where the *Slot Memory* (SM) service comes into play. SM stores extracted information from user utterances (e.g. intents, slots/values) sourced from *utterance parser* component in each turn of conversation. In the current version of SM, it uses Redis[9] to store, update and fetch slots/values (e.g. "location": "Barker street") information per intent (e.g. *SearchBusiness*).

7 Validation

In order to explore how our proposed conversational model effectively empowers chatbots to handle *multi-intent multi-turn* conversations we run a user study. To exploit this model within chatbots, we create two services for bot developers: (i) *API-KG*, an API Knowledge Graph which contains information about APIs, their methods and annotated training dataset associated to them. (ii) *Bot Builder*, a service that semi-automates chatbot development and deployment. *Bot Builder* takes any intent of interest from bot developer and deploys a trained chatbot (with our conversation model embedded inside) over third-party platforms (such as *DialogFlow*). We refer the interested reader to our work [29] for more explanation of these services.

Participants. We involved PhD students in Computer Science with experience in cloud services as participants for this user study.

[8] https://github.com/pytransitions/transitions
[9] https://redis.io/

Study scenario. Participants were asked to build a devops chatbot to interact with Amazon Elastic Compute Cloud (EC2)[10]. We chose *devops* domain because of its multiple intents nature. It is a challenging domain for chatbots to handle many intents in conversations. All chatbots in this study should support the following intents: {*Run, Stop, Start, Terminate, Describe*}*Instances*, {*Create, Describe*}*Volumes*, {*Create, Describe*}*Snapshots* and *DescribeImages*. These intents are chosen based on daily basis devops tasks for cloud infrastructure admins; therefore, a devops chatbot is expected to handle such intents for end-users (e.g. cloud admins). Each participant was asked to build four versions of his/her chatbot using the following setups:

- Wit.ai: Participants were asked to use this platform in their chatbots to recognise intents/slots. However, this platform does not have any dialogue management mechanism[3]. Therefore, chatbots that leverage this platform are only able to handle single-turn conversations. The aim of this setup is to emphasize the need for multi-turn conversations.
- Wit.ai + iConverse[11]: In this setup, participants were asked to leverage our proposed conversational model along with Wit.ai in their chatbots. Thus, chatbots are able to manage multi-turn conversations.
- DialogFlow: This platform not only offers intent/slot recognition feature, but it also provides a simple conversational model for chatbots. Thus, chatbots can handle multi-turn conversations. We consider this conversational model as the baseline.
- DialogFlow + iConverse: For this setup, participants used our proposed model instead of the baseline model (provided by DialogFlow). The aim is to see if there is any performance boost in chatbots in handling complex and ambiguous interactions.

Results and Findings. We collected a gold standard dataset of 100 utterances for mentioned 10 intents, on average 10 utterance per intent, from Amazon EC2 CLI guideline[12]. We use this dataset to analyse the performance of chatbots by considering the following criteria: (RQ1) *"how many messaging rounds, including both user and chatbot messages, needed to complete all intents?"*, (RQ2) *how many times, on average, chatbot asks user to fulfill missing information for all intent?*, (RQ3) *"how many times chatbot forgets the topic of conversation (user intent) due to filling a missing information?"*, and (RQ4) *how many times, on average, chatbot forgets user provided values for missing information in all intents? (due to intent changes)*. As we leveraged on third-party NLU platforms (Wit.ai and DialogFlow) for the Utterance Parser (UP) component in our conversational model, we could no measure the intent/slot recognition accuracy of chatbots in this study.

The evaluation results (Table 2) shows that leveraging Wit.ai (without no dialogue management mechanism) leads to have low performance in user intent

[10] https://aws.amazon.com/ec2/
[11] For simplicity, we call our conversational model as iConverse in this section.
[12] https://docs.aws.amazon.com/cli/latest/userguide/cli-services-ec2.html

Table 2. Evaluation results. We report on average values for RQs on the settings, Wit baseline (WA-B), DialogFlow baseline (DF-B), Wit + iConverse (WA-I), DialogFlow + iConverse (DF-I)

Criteria	Settings				Performance Upgrades	
	WA-B	WA-I	DF-B	DF-I	WA-B ->WA-I	DF-B ->DF-I
RQ1	120.5	**34.1**	78.6	39	**56.61%**	50.38
RQ2	NA	**6.4**	40.1	8.5	**84.03%**	78.80%
RQ3	NA	0.7	17.2	**0.3**	95.93%	**98.25%**
RQ4	NA	**0**	26.1	**0**	**100%**	**100%**

accomplishment metric (RQ1) when interactions are in the form of multi-turn. This supports the need for leveraging a conversation engine within the body of chatbots. Combining Wit.ai (as NLU model) with our proposed conversational engine, however, delivers promising performance upgrades in RQ1 and RQ2 metrics comparing to the baseline model (conversation model provided by Wit.ai) by 56.61% and 84.03%, respectively. Moreover, chatbots that benefit from our conversation engine along with DialogFlow (as NLU model only), experienced a boost in performance for RQ3 and RQ4 by 98.25% and 100%. This elevation in performance for RQ4, compare to the baseline model, is because of exploiting the Slot Manager (SM) component inside our proposed model. Due to space limits, gold dataset along with evalution results in-depth details are available to interested reader[13].

8 Conclusions and Future Work

In this paper we proposed a novel approach for the management of *multi-intent multi-turn* conversations based an extended HSM model. We proposed state machine generation techniques to support the automated initiation, monitoring and control of conversatipns. Our work also comes with its own limitations and space for possible improvements. For instance, we plan to extend our current study to involve qualitative studies by exploring how chatbots with our conversation model perform compare to the chatbots using conversation models provided by third-party platforms.

Acknowledgement. We acknowledge Data to Decisions CRC (D2D-CRC) and LIRIS Laboratory for funding this research.

References

1. Athreya, R.G., Ngomo, A.C.N., Usbeck, R.: Enhancing community interactions with data-driven chatbots - the dbpedia chatbot. WWW '18 (2018)
2. Bradley, N.C., Fritz, T., Holmes, R.: Context-aware conversational developer assistants. ICSE '18 (2018)

[13] https://tinyurl.com/t9hqyx4

3. Braun, D., et al.: Evaluating natural language understanding services for conversational question answering systems. ACL '17 (2017)
4. Bunt, H.: The semantics of dialogue acts. IWCS '11 (2011)
5. Chen, H., Liu, X., Yin, D., Tang, J.: A survey on dialogue systems: Recent advances and new frontiers. arXiv preprint arXiv:1711.01731 (2017)
6. Cranshaw, J., et al.: Calendar.help: Designing a workflow-based scheduling agent with humans in the loop. CHI '17 (2017)
7. Cuayahuitl, H., Renals, S., Lemon, O., Shimodaira, H.: Reinforcement learning of dialogue strategies with hierarchical abstract machines. SLT '06 (2006)
8. Deoras, A., Sarikaya, R.: Deep belief network based semantic taggers for spoken language understanding. Interspeech '13 (2013)
9. Fast, E., Chen, B., Mendelsohn, J., Bassen, J., Bernstein, M.S.: Iris: A conversational agent for complex tasks. CHI '18 (2018)
10. Gao, J., Galley, M., Li, L., et al.: Neural approaches to conversational ai. Foundations and Trends® in Information Retrieval **13**(2-3), 127–298 (2019)
11. Henderson, M.S.: Discriminative methods for statistical spoken dialogue systems. Ph.D. thesis, University of Cambridge (2015)
12. Hutchby, I., Wooffitt, R.: Conversation analysis. Polity (2008)
13. Ilievski, V., Musat, C., Hossmann, A., Baeriswyl, M.: Goal-oriented chatbot dialog management bootstrapping with transfer learning. arXiv:1802.00500 (2018)
14. John, R.J.L., Potti, e.a.: Ava: From data to insights through conversations.
15. Jurafsky, D., Martin, J.H.: Speech and language processing, vol. 3. Pearson (2017)
16. Kim, A., et al.: A two-step neural dialog state tracker for task-oriented dialog processing. Computational Intelligence and Neuroscience (2018)
17. Kumar, H., Agarwal, A., Dasgupta, R., Joshi, S.: Dialogue act sequence labeling using hierarchical encoder with crf. AAAI '18 (2018)
18. Li, X., Chen, Y.N., Li, L., Gao, J., Celikyilmaz, A.: End-to-end task-completion neural dialogue systems. arXiv preprint arXiv:1703.01008 (2017)
19. López, A., Sànchez-Ferreres, J., Carmona, J., Padró, L.: From process models to chatbots. In: Giorgini, P., Weber, B. (eds.) CAiSE '19 (2019)
20. Łupkowski, P., Ginz, J.: A corpus-based taxonomy of question responses. IWCS '13 (2013)
21. Mensio, M., et al: Multi-turn qa: A rnn contextual approach to intent classification for goal-oriented systems. WWW '18 (2018)
22. Raux, A., Eskenazi, M.: A finite-state turn-taking model for spoken dialog systems. NAACL '09 (2009)
23. Seo, M., Min, S., Farhadi, A., Hajishirzi, H.: Query-reduction networks for question answering. arXiv preprint arXiv:1606.04582 (2016)
24. Shah, P., et al.: Building a conversational agent overnight with dialogue self-play. arXiv:1801.04871 (2018)
25. Stolcke, Andreas, e.a.: Dialogue act modeling for automatic tagging and recognition of conversational speech. Computational linguistics **26**(3), 339–373 (2000)
26. Yan, Z., Duan, N., Chen, P., Zhou, M., Zhou, J., Li, Z.: Building task-oriented dialogue systems for online shopping. AAAI '17 (2017)
27. Yannakakis, M.: Hierarchical state machines. Springer TCS (2000)
28. Yoshino, K., Hiraoka, T., Neubig, G., Nakamura, S.: Dialogue state tracking using long short term memory neural networks. IWSDS '16 (2016)
29. Zamanirad, S.: Superimposition of natural language conversations over software enabled services. Ph.D. thesis, University of New South Wales, Sydney, Australia (2019), http://handle.unsw.edu.au/1959.4/65005
30. Zamanirad, S., et al.: Programming bots by synthesizing natural language expressions into api invocations. ASE '17 (2017)