Contextual Reactive Pattern on Chatbot building Platforms

Saurabh Srivastava
ssri@iitk.ac.in
Indian Institute of Technology
Kanpur, India

Sumit Kalra
sumitj@iitj.ac.in
Indian Institute of Technology
Jodhpur, India

T.V. Prabhakar
tvp@iitk.ac.in
Indian Institute of Technology
Kanpur, India

ABSTRACT

Building a chatbot with an iterative development process poses certain challenges for the chatbot developer. The developer is expected to produce a deployable version of the chatbot at the end of a short development cycle. Every iteration should incrementally increase the capability of the chatbot and implement a subset of overall user stories based upon a priority list, similar to any other project developed using iterative development. In this regard, commercial chatbot-building platforms offer multiple advantages to the chatbot developer, provided that the developer can map these user stories in a particular form. To do so, for every query the chatbot is expected to answer, the developer must evaluate the intention of the user. Based on the intention, the query must be processed differently, which may involve execution of some business logic. In addition, the processing of the query may require specific data items which the user must supply as part of the conversation with the chatbot. Thus, the chatbot is defined by supplying a "context" that it may encounter, and the "reaction" that must take place when the context is observed. In this work, we discuss the effects of using a platform to build a chatbot and discuss the Contextual Reactive pattern used for chatbot definition.

CCS CONCEPTS

• **Software and its engineering → Design patterns**; • **Human-centered computing → Natural language interfaces**.

KEYWORDS

Conversational Systems, Chatbots, Architectural Choices

ACM Reference Format:
Saurabh Srivastava, Sumit Kalra, and T.V. Prabhakar. 2020. Contextual Reactive Pattern on Chatbot building Platforms. In *European Conference on Pattern Languages of Programs 2020 (EuroPLoP '20), July 1–4, 2020, Virtual Event, Germany.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3424771.3424815

1 NAME

Contextual Reactive Pattern

2 CONTEXT

- Organisations are rapidly adding chatbots to their business operations to improve customer experience [9].
- Chatbots are being built to serve a wide variety of use cases, such as E-commerce, Customer Service, Information Retrieval and Travel Assistance [5].
- Any Iterative software development process (e.g. Scrum) attempts to build components, including chatbots, in an incremental fashion (e.g. in Sprints) [12] [17].
- There is an organisation which wants to deploy a chatbot to act like a conversational interface towards its business processes. They would like to start with simple and non-essential business operations, and slowly move towards more complex and essential business operations.

3 PROBLEM

- To build a chatbot to cater to a set of user queries, where the chatbot acts like a conversational interface towards accessing a set of business operations.
- To be able to produce a deployable chatbot in a short time span, albeit with minimal features, and update it over multiple iterations to cater to all the user queries.
- To be able to handle the imperfections in Natural NLP tasks [10] [4] associated with the chatbot, in a graceful manner.
- To build the chatbot with no or minimal changes in business operations codebase.

4 FORCES

- An iterative development process that forces existing versions to be modified and extended. For example, Scrum, provides a framework to build software components over multiple iterations, with each iteration building upon or improving the artifact produced in previous iterations.
- Chatbot building platforms must provide crucial NLP support, required to reduce the time for each iteration. They also provide better error handling mechanisms to tackle known issues and imperfections in these NLP tasks.
- Writing additional code for business operations specifically for use by the chatbot can increase the load on the developer significantly. It maybe undesirable for other reasons as well, such as preventing duplication of business logic.

5 SOLUTION

Figure 1 shows the solution's outline. The core idea is to pick a chatbot building platform and an iterative development process. The features to be implemented in each iteration must be mapped to a certain format, and supplied to the platform, which in turn, creates a deployable chatbot which can be used for operations.

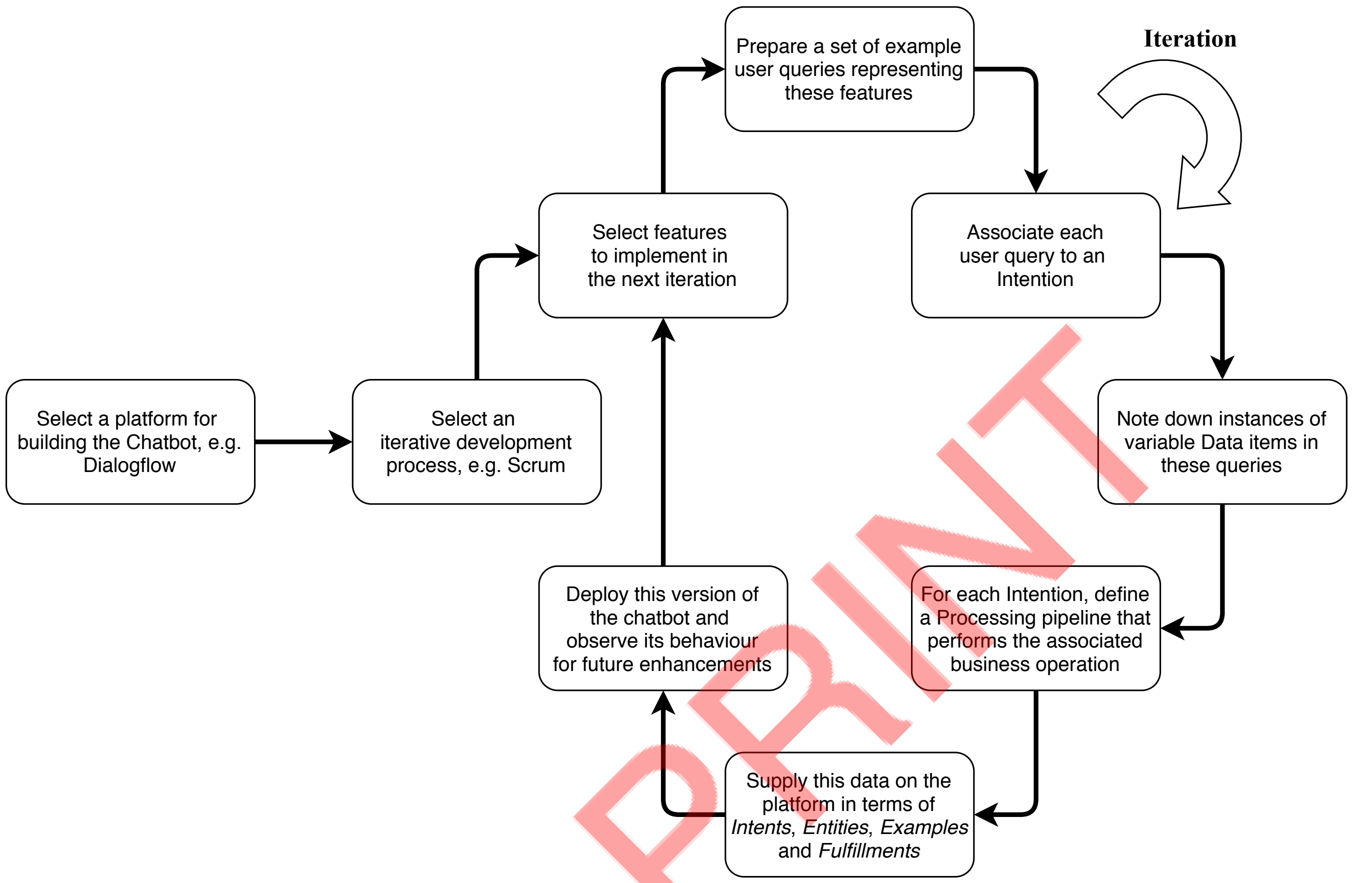
Iteration
Select a platform for building the Chatbot, e.g. Dialogflow
Select an iterative development process, e.g. Scrum
Select features to implement in the next iteration
Prepare a set of example user queries representing these features
Associate each user query to an Intention
Note down instances of variable Data items in these queries
For each Intention, define a Processing pipeline that performs the associated business operation
Supply this data on the platform in terms of Intents, Entities, Examples and Fulfillments
Deploy this version of the chatbot and observe its behaviour for future enhancements

Figure 1: Building Chatbots using a platform with the *Contextual Reactive* pattern.

The details of the solution are highlighted as follows:

- **Pick a chatbot-building platform to perform the NLP tasks.** The chatbot building platform provides two vital services:
  - For every query that the chatbot receives, the platform attempts to guess the user's intention. The chatbot developer predefines a set of possible intentions, and the platform maps the query to one (and only one) of those intentions.
  - A user query may provide implicit hints or explicit inputs required for fulfilling their intention. The platform identifies these inputs, extracts them and supplies them downstream to process the query.
- **Select the set of user stories to be served by the chatbot during the current iteration.** The factors affecting this decision are specific for a particular project, but the most important aspect is the priority of user stories based on the business operation that they represent.
- **Agree upon a set of user intentions to cater.** The platform expects that every user query must be associated with a pre-configured intention. We refer to these intentions as *Intents*. The idea is to assign an appropriate processing pipeline to each user query and generate a suitable response at the end of the processing pipeline. From an implementation perspective, if there are two queries, where the processing logic and the response generation method are exactly the same, it makes sense to categorise them under the same Intent.
- **Figure out the required and optional data items that may be present in a user query.** The platform allows the chatbot developer to define these data items in a fashion similar to defining variables in a program, i.e. they have specific types (e.g. Numeric or Ordinal) and can take values from a finite or infinite input set. We refer to these data items as *Entities*.
- **Work out the relationships between the defined Intents and Entities.** Certain Entities may only be present in queries belonging to certain Intents. We say that an Intent has a *Slot* for an Entity if the values for the Entity can appear in queries associated with the Intent. A Slot may be "optional", i.e. the query can still be processed if the user does not provide a value for it, or, it may be "required", meaning without that particular detail, processing pipeline cannot be invoked. For the Slots of the latter type, platforms usually have a mechanism to produce a temporary response, in the form of a question, asking the user to supply a value explicitly.

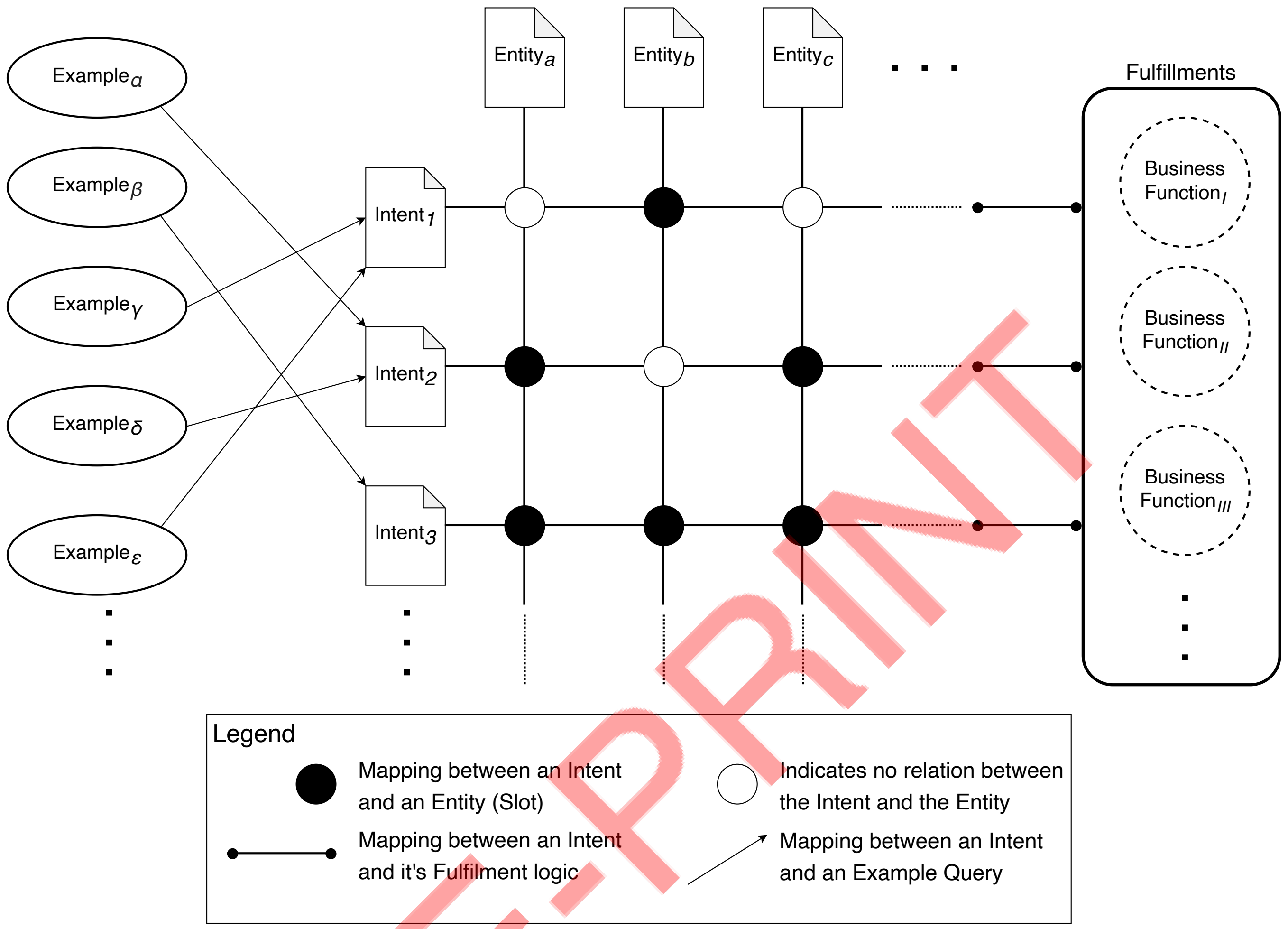
Example$_{\alpha}$
Example$_{\beta}$
Example$_{\gamma}$
Example$_{\delta}$
Example$_{\varepsilon}$
Intent$_1$
Intent$_2$
Intent$_3$
Entity$_a$
Entity$_b$
Entity$_c$
Fulfillments
Business Function$_I$
Business Function$_{II}$
Business Function$_{III}$
Legend
Mapping between an Intent and an Entity (Slot)
Indicates no relation between the Intent and the Entity
Mapping between an Intent and it's Fulfilment logic
Mapping between an Intent and an Example Query

Figure 2: Solution Structure : How Entities, Intents, Examples and Fulfilments interrelate to each other in an application.

- **Prepare a collection of possible user utterances.** The platform requires training data to build models for performing NLP tasks. This data is supplied in terms of sample user queries. We refer to these queries as *Examples*. The chatbot developer provides a few Examples for each defined Intent and tags any instances of values associated with any defined Slots within these queries.
- **Connect the processing pipelines and response generation logic to respective intentions.** The platform maps a user query to its associated Intent. It also attempts to parse instances of any Slot data supplied in the query. Then the platform invokes a pre-configured processing pipeline, which we term as the *Fulfilment* for the mapped Intent. This invocation involves supplying the parsed values of Slots as well as any additional contextual information and expecting a generated Natural Language response which is relayed to the user. Two common examples of processing pipelines are serverless functions [13] and REST API endpoints [14] implementing specific business operations.
- **Handle the user stories to serve in next iterations.** In addition to the defined Intents, another Intent can be defined to cater to the user stories which will be served by the chatbot in future iterations. We refer to such an Intent as the `others` Intent. It involves providing Examples for such user stories, but not connecting them to a processing pipeline. Instead, this Intent could be configured to output static responses (e.g. "We currently do not support this feature").
- **Handle imperfections in NLP tasks.** In addition to the above Intents, define another Intent to handle the "irrelevant" or "ill-formed" queries. Platforms usually have a mechanism to invoke a special processing pipeline for cases where the query could not be mapped to any defined Intent. Theoretically, this can be seen as the presence of a "default" or "fallback" Intent. Typically, the "default" Intent can either be handled via a special processing pipeline (e.g. transferring the chat session to a human) or responded with static responses (e.g. "Sorry, can you rephrase the query?").
- **Deploy the chatbot and collect usage data.** A detailed analysis of the chat data can provide important insights.

In particular, an important dimension of inspection is the proportion of the queries that matched the `others` Intent. A high percentage indicates one of the two possibilities:
  – The current set of user stories catered by the chatbot represent only a small fraction of the overall use cases that the user expects the chatbot to serve.
  – The platform is not able to perform a good job of mapping a user query to its correct intention. It indicates that the platform needs more Examples for training, or the quality of Examples used previously were not up to the required quality mark.

  This data may provide crucial inputs for planning the next iteration.
- **Plan the next iteration.** This involves catering to other user stories and refining the definitions or Examples for the existing user stories. As more Intents are added iteratively, the number of user stories handled by the `others` Intent keeps reducing every iteration, and finally, the Intent is removed.

5.1 Structure

Figure 2 shows the elements of the chatbot definition and their relationships. A summary of the elements is given below:

- Intents: An Intent represents a collection of related user queries, associated with a particular user story. The set of Intents for a chatbot partition all possible queries that the chatbot may receive into mutually exclusive categories.
- Entities: Entities represent real-world objects supplied implicitly in a particular query. Each Entity associated with a chatbot can accept a value from a finite or infinite set, pre-defined at the time of chatbot definition.
- Examples: Examples are sample user queries that a user may fire at the chatbot during a conversation. An Example is mapped to one and only one Intent and may contain values associated with zero or more Entities.
- Fulfilments: Fulfilments represent the processing pipeline that must be executed to prepare a response for any query. In some cases, where the response is a static message, the platform usually provides a mechanism to output the same to the user without making a remote function call. Fulfilments can also be gateways to external API endpoints which can be invoked to prepare the response (as well as perform any actions) for the given query. A fulfilment is associated with a respective Intent and is invoked every time a query is classified to have the said Intent.

In addition, Slot is a term that represents the instantiation of an Entity with respect to a particular Intent. An analogy would be - Slots are to Entities, what Objects are to Classes. A Slot thus represents the mapping between an Intent and an Entity. The relationships between the constituents are represented in Figure 2.

5.2 Dynamics

After deployment, the Chatbot performs some crucial NLP tasks and follows a workflow. A typical workflow is shown in Figure 3. A summary of the workflow is provided below:

- The user asks a query. The query may have implicit values associated with one or more Entities.
- The chatbot performs two NLP tasks - guessing the Intent associated with the query and finding out if there are values associated with any defined Entities, supplied within the query.
- For the detected Intent, the chatbot looks at the Slot details (recall that a Slot represents a relationship between an Entity and an Intent). If there are parsed values which belong to one or more Slots, they are stored temporarily. Next, the chatbot checks if there are any Slots defined as "required". If present, the chatbot checks if any values for these Slots are present in the temporary storage or not. If not, the chatbot starts producing a series of responses, essentially prompting the user to supply these values. This process may continue till values for all the "required" Slots are gathered or the chatbot may give up after a certain number of attempts, providing a specific response configured for such cases (the exact behaviour may vary from one platform to another).
- Once the chatbot has values associated with all "required" Slots (and any "optional" Slots, if provided), the chatbot passes this information to the processing pipeline. This is the step which decouples logic associated with business operations from the intricacies of the chatbot implementation. The processing pipeline can reside inside an external system or maybe part of a legacy system already operational. The query may trigger an idempotent operation such as a lookup in a database, or, it may perform actions in the business domain, such as creating a new booking. The details of the operation are abstracted from the chatbot. The chatbot only expects a response, generated at the completion of the processing.
- The response received after the processing is relayed back to the user. The chatbot now expects another query from the user, and the cycle is repeated.

6 REAL-WORLD EXAMPLES

There are many platforms which provide support for building chatbots. The elements of the chatbot definition discussed in the solution can be observed in almost all of them with minor variations. We discuss one of these platforms - Google Dialogflow [6] - in detail, and present a comparison of terminology and concepts with two other popular platforms - IBM Watson Assistant [8] and Amazon Lex [1].

6.1 Google Dialogflow

- Intents: Each Intent is defined in the dashboard on a separate page. Typical information provided on the definition page is Intent's name and some "Training Phrases". If the Intent has any associated Slots, the same needs to be configured on this page itself. In Dialogflow, Slots are called "Parameters". Prompts can be configured for one or more Parameters on the same page. Dialogflow provides a set of common Intents which can be added to any chatbot (e.g. Intents for casual

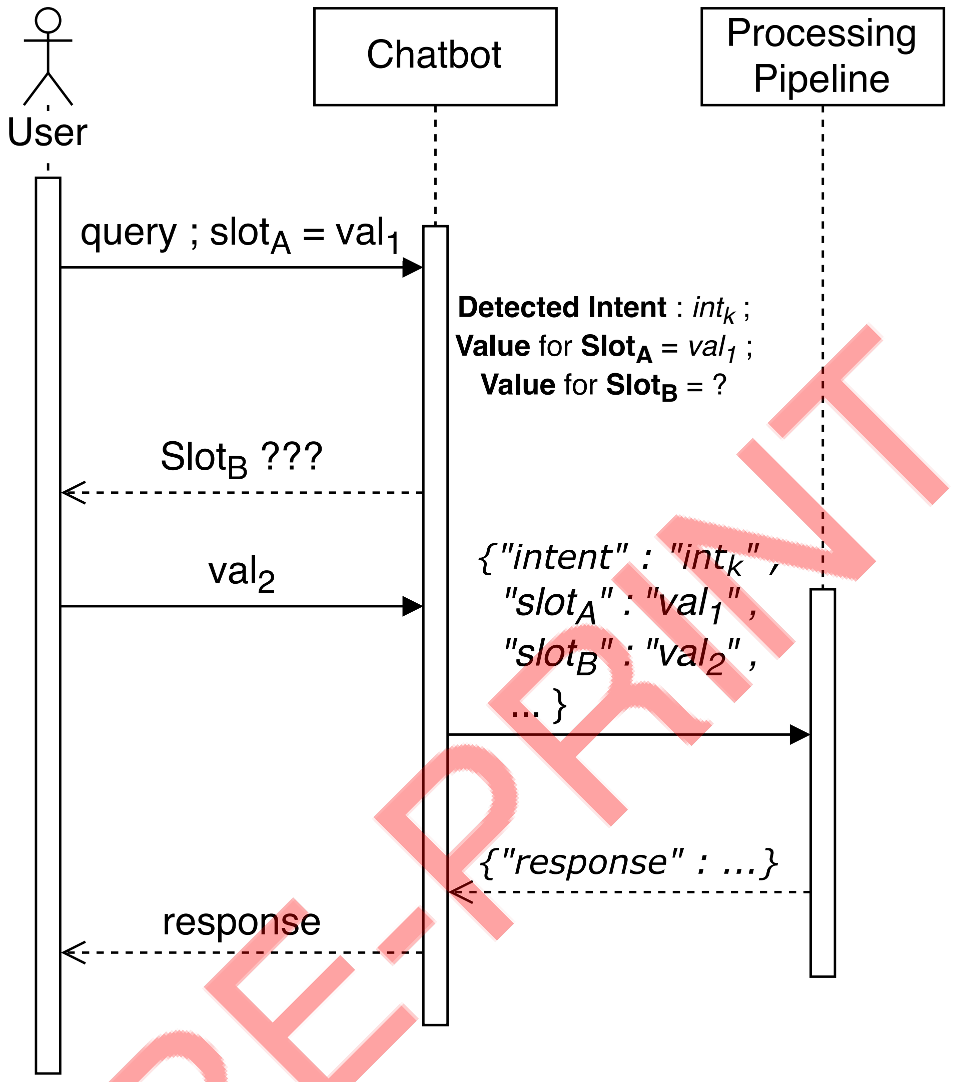

Figure 3: Deployed Chatbot's Sketch : Typical workflow observed during the operation of a chatbot.

conversation with a user). It also adds a Welcome and a Fallback Intent by default to every chatbot to greet and handle irrelevant user queries respectively.

- Entities: Each Entity is defined in the dashboard on a separate page. There are some System Entities, which Dialogflow provides for usage directly, for instance, Geographic Locations or instances of Time and Date. However, for most user stories, the developer may have to define one or more custom Entities, which are usually nominal in nature. For each Entity, a set of possible values are defined. For each value, synonyms can be defined. Dialogflow offers the option to intelligently extend possible values for an Entity, in case it is not feasible to list down all of them beforehand.
- Examples: Examples are called "Training Phrases" in Dialogflow. They are provided directly on the definition page of the associated Intent. Dialogflow expects that instances of Slots are tagged in the Example, to help the Slot Filling task. Negative Examples can be configured on the page of the default Intent called the Default Fallback Intent. These Examples provide explicit hints to trap irrelevant user queries.
- Fulfilments: Fulfilments can be defined in two different ways in Dialogflow. For Intents, where the response is fixed, the response, along with placeholders for Parameters, can be directly supplied on the Intent definition page. For Intents requiring complex Fulfilment Logic, one has to define an Action - a unique tag associated with the Intent - and provide a common Fulfilment webhook on Fulfilments page. For such Intents, the chatbot sends a POST HTTP request to the configured URL with the name of the Action and other inputs, such parsed values of Slots. The webhook must provide a JSON reply with a fixed schema. The response is then relayed back to the user.

6.2 Google Dialogflow vs IBM Watson Assistant vs Amazon Lex

We omit similar details for two other popular platforms, but highlight the differences in terminology and concepts:

7 EXAMPLE RESOLVED

As an example, we show how a chatbot can be built with a platform using the Contextual Reactive pattern. Consider a fictitious airline called Chanakya Airlines. Assume that they operate a limited number of flights daily between a few cities. The airline already has a website through which users can perform basic business operations such as searching for an appropriate flight, booking a ticket, cancelling a booked ticket etc. The airline, in an attempt to provide better user experience, wishes to deploy a chatbot on their website, which can help users do the same through a conversational interface. After a few sprint planning meetings, it was decided that the chatbot will be built using the Dialogflow platform. A set of user stories were also finalised. The airlines are open to sharing a REST endpoint with the developer, which can be invoked to perform the business operations. The endpoint takes a JSON object as input and returns another JSON object as the response after the execution of the operation.

7.1 User Stories

The chatbot developer has to implement certain user stories in the priority of their business importance. The user stories, in decreasing order of their priority, are provided below (the format used to express these stories is loosely based on the format discussed by Mike Cohn [3]):

7.1.1 *Search for a flight.*

- Description: As a customer of Chanakya Airlines, I want to search for a flight between two cities on a particular date.
- Example: As a customer of Chanakya Airlines, I want to know about all the flights, if there are any, between Delhi and Mumbai on coming Friday.
- Conditions of Satisfaction:
  - The user shall be prompted to supply a source, a destination and a date.
  - A list of applicable flights (if any) shall be shown.
  - Nice to have - show only day or night flights (as per user's preference).

7.1.2 *Book tickets on a flight.*

- Description: As a customer of Chanakya Airlines, I want to book one or more tickets on a particular flight for a particular date.
- Example: As a customer of Chanakya Airlines, I want to book a ticket on the flight EX-101 for tomorrow.
- Conditions of Satisfaction:
  - The user shall be prompted to supply an email and the date of booking.
  - On success, the user shall be given a Booking Id for future reference.
  - Nice to have - multiple tickets could be booked in a single booking (if the user wishes to book more than one ticket on the same flight on the same date).

7.1.3 *Show previously made bookings.*

- Description: As a customer of Chanakya Airlines, I want to find out all the previous bookings I made with Chanakya Airlines (that are not cancelled).
- Example: As a customer of Chanakya Airline, I want to know the details (such as Flight Number, Source, Destination etc.) of my previous bookings.
- Conditions of Satisfaction:
  - The user shall be prompted to supply the email used for the bookings.
  - A list of all bookings (if any) made with the email (that are not cancelled) shall be shown.
  - Nice to have - show bookings for a particular travel date only (as per user's request).

7.1.4 *Cancel a booking.*

- Description: As a customer of Chanakya Airlines, I want to cancel a previous booking made with Chanakya Airlines.
- Example: As a customer of Chanakya Airlines, I want to cancel a booking with Booking Id 1001.
- Conditions of Satisfaction:
  - The user shall be prompted to supply a Booking Id and the email used for making the booking.
  - On success, the user shall be informed about the cancellation, and the booking details must be removed from the bookings database.

7.2 First sprint

In the first sprint, the chatbot developer picks the top two user stories to implement. In addition, the developer has decided to achieve only the core conditions of satisfaction - keeping the "Nice to have" aspects for the second sprint. The chatbot developer maps the user stories to following elements on Dialogflow:

7.2.1 *Intents.*

- The *search* Intent caters to inquiries about flights.
- The *book* Intent handles requests for booking flight tickets.
- The *others* Intent handles requests related to user stories to be handled in the next sprint(s).
- Dialogflow provides a *Default Welcome* Intent and a *Default Fallback* Intent automatically. The former can cater to mundane greeting queries (e.g. *Hi* or *Hello*), whereas the latter produces a response to cater to queries that may be "irrelevant" for the chatbot (e.g. *How's the weather today?*).

7.2.2 *Entities.*

- The *FlightNumber* entity is a regex entity, which can take values in the form *EX-ddd*, where *d* represents a digit. It represents the flight number of a particular Chanakya Airlines flight.
- The *Source* entity is a system entity which can take a value of any major city such as Delhi or Mumbai. It represents the origin of a particular Chanakya Airlines flight.

Table 1: Comparison among Google Dialogflow, IBM Watson, Amazon Lex

	Dialogflow	Watson	Lex
Intents	Two Intents are added automatically - Welcome and Default Fallback Intent	Every Intent must be created or added by the developer	Every Intent must be created or added by the developer
Entities	No implicit version control	No implicit version control	Entities are implicitly version controlled - every time a change is made to an Entity
Slots	Slots are called Parameters, Slots are defined on the Intent definition page	Slots are defined in the Dialog tree	Slots are defined on the Intent definition page
Examples	Examples are called Training Phrases, Values associated with any Entity in the examples, are explicitly tagged, e.g. Search for a flight tomorrow -> journey-date	Examples are called User Examples, Values associated with any Entity in the examples, are explicitly tagged, e.g. Search for a flight tomorrow -> journey-date	Examples are called User Utterances, Lex does not expect actual values of Entities in the examples, but a placeholder for them shall be provided explicitly, e.g. Search for a flight journey-date
Fulfillment	Simple responses can be configured on the Intent definition page, For invoking external business logic, a common fulfilment webhook could be defined for the chatbot	Responses can be configured in the Dialog tree for many complex scenarios, Fulfilment logic can be configured at external API endpoints or IBM Cloud functions [7]	Simple responses can be configured on the Intent definition page, Fulfilment logic can be configured as AWS Lambda [2] functions
Other notable aspects	Dialoflow provides a set of pre-configured Intents, such as those for having a casual conversation, External business logic must be exposed through a single API endpoint	Watson Assistant provides a set of pre-configured Intent group, known as Skills, Different Intents can invoke different API endpoints	Lex provides a set of pre-built intents, mostly useful for playing media (e.g. PauseIntent, ShuffleOnIntent, LoopOffIntent etc.), Different Intents can invoke different AWS Lambda functions

- The *Destination* entity is a system entity similar to the Source entity and can take a value of any major city. It represents the destination of a particular Chanakya Airlines flight.
- The *Date* entity is a system entity which can take a value of valid date. Dialogflow also maps relative utterances such as "tomorrow" or "yesterday" to a date string in ISO 8601 format, e.g. *2020-05-01T12:00:00+05:30* (representing 1st May 2020 in Indian Standard Time). It represents the date of departure of a particular Chanakya Airlines flight.
- The *Email* entity is a system entity which can take a value of a valid email address. It represents the email used to make a booking with Chanakya Airlines.

Since the first sprint does not cover the user stories related to cancellation or display of previous booking, the training queries for these Intents (such as "`cancel my bookings`" and "`show my bookings`") are associated with the *others* Intent. Queires like these are handled with the static response:

```
We request you to connect to our Customer Care
for assisting you with this issue ....
```

In addition, the queries related to searching for a flight or booking tickets perform only their core tasks, ignoring some details. For example, the queries "`show flights from Mumbai to Delhi`" and "`show night flights from Mumbai to Delhi`" produce the same response, even though, the latter must only display the night flights.

7.3 Second sprint

The second sprint involved adding functionality to the chatbot built in the first sprint, and improving it by adding the "Nice to have" features. The main agenda of the second sprint is summarised below:

- Implement the two remaining user stories, and remove the others intent created to handle queries related to them.
- Implement the "Nice to have" features for related to each user story.

We only discuss the details that were added in the second sprint:

7.3.1 Intents.

- The *show* intent caters to inquiries about existing (non-cancelled) bookings.
- The *cancel* intent handles requests for cancelling existing bookings.

The *others* Intent is no longer required, since we now have Intents related to all user stories. Thus, it is removed in the second sprint.

7.3.2 Entities.

- The *Time* entity is a nominal entity, which can take two values - *day* or *night*. We define a few synonyms for both values, such as *morning* for *day* and *late* for *night*.
- The *NumberOfTickets* entity is a system entity that can take any integer as value. It represents the number of tickets to book as part of a single booking.
- The *BookingId* entity is a system entity that can take any integer as value. It represents the Booking Id for a particular Chanakya Airlines booking.

The second sprint gets rid of the *others* Intent, since the queries that it was matching are now handled through their respective Intents (*show* and *cancel*). Also, the queries related with *search* and *book* Intents can now handle more details, such as night or daytime flights, and more than one tickets can be booked through the same query.

The configuration files for creating the Chanakya Airlines chatbot on the Dialogflow platform is available at [16]. The code that imitates the backend operations of Chanakya Airlines can be found at [15]. Both repositories are licensed under the MIT license [11].

8 CONSEQUENCES

8.1 Benefits

- The "others" Intent can cover a large number of user stories in initial iterations and its role in the chatbot operation decreases as new Intents are added in future iterations. The "default" Intent can provide graceful responses to cover imperfections in NLP tasks.
- The chatbot-building platform requires only a limited number of real-world examples to train its models, and can extrapolate in multiple directions as required.
- The logic associated with business operations can be executed as part of the processing pipeline. The details of the logic are abstracted and decoupled from the core NLP tasks performed by the chatbot.

8.2 Liabilities

- The chatbot associates each query with one and only one Intent. A complex query which can be associated with more than one Intent cannot be handled.
- The chatbot supplies the inputs to the processing pipeline and expects the response in a platform-specific format. This may require creating wrappers over existing cloud functions or REST API endpoints.

9 CONCLUSION

In this work we discussed the problem of building a chatbot to act like a conversational wrapper over some business operations. We viewed this problem from a practical perspective, where the chatbot developer has to quickly produce a deployable version and improve it over multiple iterations - a common practice in the industry to build software components. We argued that using a chatbot building platform can help the developer in more than one way. In particular, the platform provides support for crucial NLP tasks, provides a flexible format to define the chatbot, facilitates iterative implementation of features and offers a robust mechanism for error handling. The platforms enforce a pattern on the definition of the chatbot, that we term as Contextual Reactive pattern. The pattern also allows decoupling of the background business operations from the core objectives of the chatbot, enabling a crucial abstraction towards the separation of concerns. Picking a suitable platform, an iterative development process such as Scrum, and following the Contextual Reactive pattern can be very helpful for the chatbot developers towards achieving their goals.

10 ACKNOWLEDGEMENTS

We are highly indebted towards our shepherd, Stefan Holtel for his valuable comments that helped us improve our work significantly. We would also like to express our gratitude for the EuroPLoP community for helping us shape the paper better.

REFERENCES

[1] Amazon.com Inc. [n.d.]. Amazon Lex – Build Conversation Bots. https://aws.amazon.com/lex/ Online, Access Date: 2020-07-01.
[2] Amazon.com Inc. [n.d.]. AWS Lambda – Serverless Compute - Amazon Web Services. https://aws.amazon.com/lambda/. (Accessed on 08/08/2020).
[3] Mike Cohn. 2013. User Stories and User Story Examples by Mike Cohn. https://www.mountaingoatsoftware.com/agile/user-stories. (Accessed on 08/20/2020).
[4] discover.bot. 2018. Pitfalls of Natural Language Processing Chatbots - discover.bot. https://discover.bot/bot-talk/natural-language-processing-common-challenge/. (Accessed on 08/19/2020).
[5] Daniel Faggella. 2019. 7 Chatbot Use Cases That Actually Work | Emerj. https://emerj.com/ai-sector-overviews/7-chatbot-use-cases-that-actually-work/. (Accessed on 08/18/2020).
[6] Google LLC. [n.d.]. Dialogflow. https://dialogflow.com/ Online, Access Date: 2020-07-01.
[7] IBM Corporation. [n.d.]. IBM Cloud Functions. https://cloud.ibm.com/functions/. (Accessed on 08/08/2020).
[8] IBM Corporation. [n.d.]. Watson Assistant. https://www.ibm.com/cloud/watson-assistant/ Online, Access Date: 2020-07-01.
[9] Business Insider. 2016. 80% of businesses want chatbots by 2020.
[10] AI Multiple. 2020. Why chatbots fail in 2020 (and why natural languages are hard). https://research.aimultiple.com/why-chatbots-fail/. (Accessed on 08/19/2020).
[11] Open Source Initiative and others. 2006. The MIT license.
[12] openwt.com. [n.d.]. Building a Chatbot leveraging Artificial Intelligence Technologies | Open Web Technology - Go Digital. https://openwt.com/en/cases/building-chatbot-leveraging-artificial-intelligence-technologies. (Accessed on 08/18/2020).
[13] pubnub.com. 2019. What is a Serverless Function? | PubNub. https://www.pubnub.com/blog/what-is-a-serverless-function/. (Accessed on 08/19/2020).
[14] SMARTBEAR. 2018. What is an API Endpoint? | SmartBear Software Resources. https://smartbear.com/learn/performance-monitoring/api-endpoints/. (Accessed on 08/19/2020).
[15] Saurabh Srivastava. 2020. AirlineOperationsBackend Repository — Bitbucket. https://bitbucket.org/ssri5/airlineoperationsbackend/. (Accessed on 08/21/2020).
[16] Saurabh Srivastava. 2020. ChanakyaAirlinesConfiguration Repository — Bitbucket. https://bitbucket.org/ssri5/chanakyaairlinesconfiguration/. (Accessed on 08/21/2020).
[17] thebotforge.io. [n.d.]. Forging A Successful Chatbot Project | The Bot Forge. https://www.thebotforge.io/forging-a-successful-chatbot-project/. (Accessed on 08/18/2020).