Universidad Autónoma de Madrid

Escuela Politécnica Superior

MASTERS THESIS

AUTOMATED SYNTHESIS OF CHATBOTS FOR CONFIGURING SOFTWARE PRODUCT LINES

Masters in Formal Methods in Computer Science and Engineering
UCM/UAM/UPM

Jerry, Blessing
Tutor: Juan de Lara Jaramillo
Department of Computer Science

September, 2021

AUTOMATED SYNTHESIS OF CHATBOTS FOR CONFIGURING SOFTWARE PRODUCT LINES

Autor: Jerry, Blessing
Tutor: Juan de Lara Jaramillo

Modelling and Software Engineering Research Group(MISO)
Department of Computer Science
Escuela Politécnica Superior
Universidad Autónoma de Madrid

September, 2021

Abstract

Abstract — Software product lines are a method for creating a family of products that share a typical managed set of features, satisfy the precise needs of a selected domain, and provide an improved quality of software systems by systematically reusing software artefacts at reduced cost and time. A feature model represents the space of all possible and allowed configurations of all products in an SPL. Various predefined feature combinations enable the product to be personalized based on specific user requirements. However, because some features are interdependent and the feature models may have many options, users must understand the implications of selecting the correct feature combinations for the product derivation. Chatbot support can address this challenge by guiding the user through a suitable set of features for the product configuration process. Users can interact with a chatbot using natural language in a familiar environment like Telegram, Slack, or Facebook. In this work, we propose chatbots in the configuration of software product lines based on feature models and present SPLBOT, an approach for SPLs chatbot generators. The methodology relies on Eclipse, FeatureIDE, and CONGA (for Dialogflow chatbot generation). Furthermore, we present an evaluation of our approach's effectiveness and scalability using three practical examples.

Key words — Software Product Lines, Feature Model, Configuration, Chatbot, FeatureIDE

Contents

1 Introduction **1**

2 Preliminaries **5**
2.1 Software Product Lines 5
2.2 Feature Modelling 6
2.3 Chatbots 11
2.3.1 Natural Language Processing 14
2.3.2 DialogFlow 14
2.4 CONGA DSL 16
2.5 Related Work 17

3 Approach **21**
3.1 General Approach 21
3.2 Feature Model Mapping to Corresponding Chatbot 22
3.3 Heuristics 30
3.4 Additional Functionalities 30

4 Architecture and Tool **33**
4.1 Architecture 33
4.1.1 FeatureIDE 34
4.1.2 CONGA for Chatbot Synthesis from Feature Model 35
4.1.3 Dialogflow 39
4.2 Tool 42
4.2.1 Using the generated chatbot in Telegram 42

5 Evaluation **47**
5.1 Experiment set-up 47
5.2 Evaluation Description 48
5.3 Discussion and Threats to Validity 49

6 Conclusion and Future Work **51**
6.1 Conclusion 51
6.2 Future Work 52

Bibliography **53**

Appendices **61**

A Create and import an agent in Dialogflow console **63**

B Create a bot for Telegram **67**

List of Tables

2.1 Expressing a feature diagram with logic 9
2.2 Feature model translation to propositional formulas 10
2.3 Sample car feature model abbreviation. 11
2.4 Chatbot creation tools 12

3.1 Information Extraction 23
3.2 Sample Analysis Queries on FM 31

5.1 Size of the feature models 48
5.2 Generated chatbots 49

List of Figures

2.1 Domain and application engineering phases in SPL development 6
2.2 A sample feature model specified using FeatureIDE 7
2.3 Chatbot working scheme 13
2.4 A screenshot of Dialogflow 15

3.1 Overview of our proposal 22
3.2 Flow 29

4.1 Architecture of SPLBOT 34
4.2 Training Phrases of the "Config Intent" 40
4.3 Bot Response for the "Config Intent" 40
4.4 Context for the "Get_Transmission Intent" 41
4.5 Training Phrases of the "Get_Transmission Intent" 41
4.6 Parameter of the "Get_Transmission Intent" 42
4.7 Bot Response for the "Get_Transmission Intent" 42
4.8 Beginning of conversation with the bot 43
4.9 Interaction with chatbot for selection of features 43
4.10 User Makes optional feature selection 44
4.11 User Completes feature selection 44
4.12 User Interaction with chatbot to Request Help 45
4.13 User queries chatbot for the number of features 45
4.14 User queries chatbot for all possible configurations 46
4.15 File content 46

1 Introduction

Software Product Lines (SPLs) have been predominant in the derivation of software products within a well-defined application domain [1]. A SPL refers to a collection of software-intensive products that share a typical managed set of features, satisfy the precise needs of a selected market segment (domain), and are developed from a common set of Core Assets in a prescribed way. A well known approach to understand and define commonalities and variabilities in software product lines and to support product derivation is by means of Feature Modelling (FM) [2, 3, 4, 5, 6, 7].

A feature model is a way to represent the space of possible and allowed configurations of all products in an SPL [8, 3]. It defines the domain, product model constraints, and which features are present in the final products. Feature modelling (FM) and product configuration are central parts of software product line development [9]. The majority of the existing FM approaches are practically derived from the work on Feature-Oriented Domain Analysis (FODA) method in [10].

Product configuration and generation are usually affected by the qualitative feature models. A correct and desired feature model is the basis for all subsequent stages in product line development. Feature modelling and product configuration, however, are manual and thus error-prone tasks [9, 11]. A feature model may have many features and interdependencies, potentially ensuing in a few inconsistencies in the feature model. These inconsistencies can lead to a configuration that does not permit the creation of a valid feature combination. As the configuration of SPLs may be difficult for large and complex feature models, the integration of the stakeholders and users in the configuration process can improve the handling of the feature selection problem [12].

A chatbot also known as conversational agent, is designed to produce an impression of interacting with a human, because chatbots use natural language [13, 14, 15, 16, 17].

Users can chat through text or voice input over an interface with chatbot text output or audio/voice output. Chatbots are increasingly used to facilitate software engineering activities [13, 18], as a customer service agent SPLBOT[13, 19, 20, 21], virtual support agents, virtual sales agents, provide specific information and guidance through a website, help to solve frequently asked questions, or as a learning tutor [13, 16]. Modelling chatbots can be embedded in social networks (like Facebook [22, 23], Slack [18, 23], Twitter [24] or Telegram [25]) to support collaboration between different stakeholders in a natural way, so that non-technical stakeholders can actively participate in model creation [26].

Due to the growing interest in chatbots, many chatbot development tools have emerged, such as DialogFlow[1], the IBM Watson Assistant[2], or the Microsoft Bot Framework[3]. Some of these tools offer a cloud based development environment that supports the design of chatbots, from the application of natural language (NL) processing to the deployment of chatbots in social networks.

In this work, we use chatbots in the configuration of software product lines based on feature models and present an approach for chatbot generators called SPLBOT for SPLs. Our intended goal is to bring in automated support for the feature selection process through the application of synthesized chatbots based on feature models. Product configuration is error prone, and may be difficult if FMs have many options. Therefore, it is important to correctly select the right feature combinations for the product derivation. Clearly, the method to select these features should rely on a measure of the error-free feature model that represents the desired product. The chatbot support is required to handle the resulting complexity of product configuration, by NL.

Our approach provides a system for ensuring configurations that conform to the feature model and ensures the quality of implementation of the product line. To achieve the aim of this study, the following measures are taken:

(i). The availability of a feature model specifying the features, their relationships, and the constraints of feature selection for product configuration in software product lines (configuration space).

(ii). Mapping of a feature model to its corresponding bot.

(iii). Providing a set of algorithms tailored to the feature modelling domain that can be used to provide automated support for product configuration.

(iv). A configuration plan that describes the configuration tasks and the order in which they should be carried out.

(v). Application of natural language processing (NLP) and conversational agents to safely guide stakeholders and users during the configuration process with the help of the conversation flow plan.

[1] https://dialogflow.com/
[2] https://www.ibm.com/cloud/watson-assistant/
[3] https://dev.botframework.com/

(vi). Automatic generation of a chatbot supporting features over instances of a given feature model.

The evaluation is based on a study with feature models from Eclipse FeatureIDE[4] and the online repository SPLOT[5].

The rest of the thesis is structured as follows: Chapter 2 reviews some basic concepts used in the thesis and sets the background knowledge for the next chapters. Chapter 3 presents details of our approach. Chapter 4 describes the prototype tool support. Chapter 5 reports on the evaluation of the proposal. Finally, Chapter 6 concludes and provides some lines for future work.

This work has been funded by Universidad Complutense de Madrid within the framework of the Learn Africa Programme launched by the Women for Africa Foundation.

[4]https://featureide.github.io/
[5]http://www.splot-research.org/

2

Preliminaries

This chapter introduces the main concepts about software product lines (section 2.1). In section 2.2, we introduce feature modelling, an approach to capturing variabilities and commonalities in software product lines. Finally, in section 2.3, we present an overview of the working scheme of a chatbot and how the chatbot design concepts are realized in DialogFlow.

2.1 Software Product Lines

Software Product Lines is a modern approach to software development that utilizes similarities and variations within a family of systems in a specific domain of interest to provide an improved quality of software systems by systematically reusing software artefacts at reduced cost and time [27, 28].

According to [29], two key processes of software product line engineering are domain engineering and application engineering. Domain engineering is the process in which the commonality and variability of the product line are defined and realised (establishing the reusable platform). Application engineering is the process in which applications of the product line are built by reusing domain artefacts and exploiting the product line variability (deriving product line applications from the platform established in domain engineering) [29].

In SPL practice, the two life cycles, domain engineering and application engineering, differ from single system software development [30], as illustrated in Figure 3.1. Domain engineering centers on a family of systems and is concerned with developing for reuse [31]. Application engineering uses integrated information and specializes according to the

needs of a specific service request. The process of domain engineering and application engineering (Figure 3.1) as described in [30] includes three major activities. The activities performed during domain engineering are domain analysis (determines what the family is about), domain design (deciding which platform components are needed), and domain implementation (building and purchasing components and supporting infrastructure). Accordingly, application engineering also requires three activities, application analysis, application design, and application implementation. As seen in [30], a feature model that captures services provided by the applications in a domain in an abstract form can be used to effectively plan and consider reusability during the early phases of the software life cycle and throughout the development process.

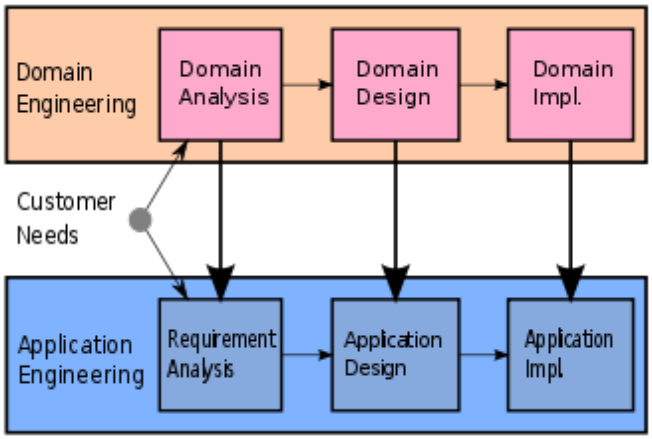


Figure 2.1: Domain and application engineering phases in SPL development

The main idea in SPL is to start with families, identify what is common and what is variable, create an environment including architecture, definitions, tools for producing members of a Product Line and then produce Product Line members according to the wishes of the marketplace. SPL engineering is about producing a family of similar systems rather than the production of individual systems [9, 32]. In a product line, there is a need to plan and enforce reuse by specifying the architecture for the reuse of the components and enforce that all products use the same architecture. The key to the product line is the ability to predict likely changes in the product in the future and across the marketplace. Product lines have been found to bring in improvements to software development in terms of cost, time to market, and productivity [1].

2.2 Feature Modelling

Feature modelling is a feature-oriented approach to commonality and variability analysis and has been widely used for product line engineering [10, 33]. The feature modelling task involves identifying the commonalities and variabilities of the products of a product line based on features and arranging them into a feature model. The term feature is used as an abstraction for core assets and as building blocks for the identification of products in

the product line. This implies that each product is represented by a specific combination of features. Feature models (FM) are visually represented by means of a feature diagram. The notion of representing feature models as a feature diagram was proposed by [10]. Several different studies [9, 28, 34, 35, 36] describe a feature diagram as a tree structure that represents the variability of the product line (i.e., the feature model).

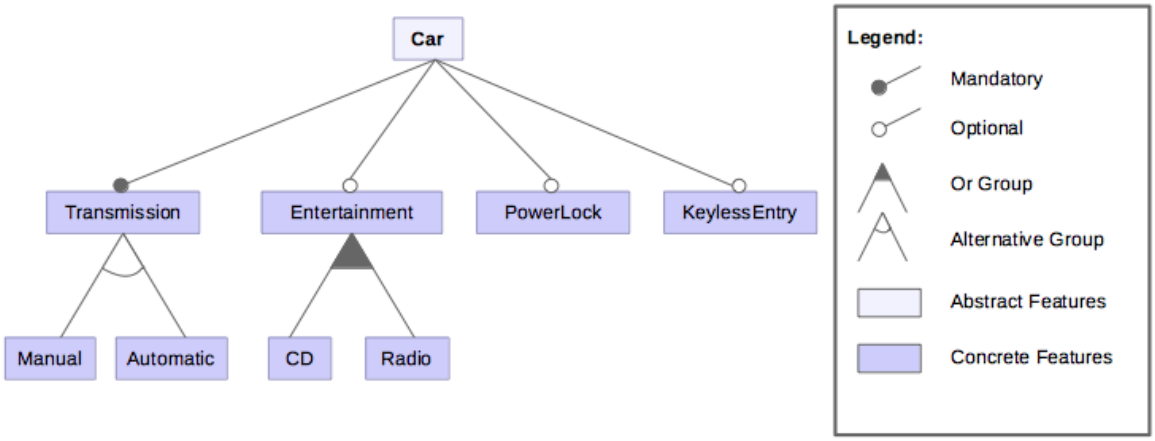


Figure 2.2: A sample feature model specified using FeatureIDE

As an example, Figure 2.2 depicts a sample feature model of a car product line using Eclipse and FeatureIDE (an eclipse based IDE that supports all phases of feature oriented software development for the development of SPLs) [37, 38, 39]. The relationships between a parent (or compound feature) node and its child nodes (sub-features) largely fall into the following categories:

Mandatory relationship: a mandatory relationship is one in which the child feature must be added in all the products in which the parent feature appears (e.g., each instance of "Car" must have Transmission). The mandatory relationship is represented by a simple edge from the parent (P) node to the child node that ends with a filled circle.

Optional relationship: an optional relationship indicates that the child feature is optionally included in all products in which the parent feature appears (e.g., with the car product line example, every instance of Car may have Entertainment, KeylessEntry or PowerLock). This relationship is represented by a simple edge from the parent node to the child node that ends with an empty circle.

Alternative relationship (xor-relationship): an alternative relationship is described when only one child can be chosen when the parent feature is part of the product (e.g., manual or automatic, but not both). The nodes of a set of alternative features are connected by an empty arc.

Or relationship: the “or” relationship is defined when one or more children may be selected when their parent feature is part of the product (e.g., at least one of radio and cd). The nodes of a group of "or" features are connected by a filled arc.

According to [33], the structural relationship of feature models is described as commonalities among all products of a product line modelled as common features, while variabilities among products modelled as variable features, from which product specific

features is selected for a specific product.

The sets of all valid products of the car feature model in Figure 2.2 are:
P1{Car, Transmission, Manual}
P2{Car, Transmission, Automatic}
P3{Car, Transmission, Manual, Entertainment, CD}
P4{Car, Transmission, Manual, Entertainment, Radio}
P5{Car, Transmission, Manual, Entertainment, CD, Radio}
P6{Car, Transmission, Automatic, Entertainment, CD}
P7{Car, Transmission, Automatic, Entertainment, Radio}
P8{Car, Transmission, Automatic, Entertainment, CD, Radio}
P9{Car, Transmission, Manual, KeylessEntry, PowerLock}
P10{Car, Transmission, Automatic, KeylessEntry, PowerLock}
P11{Car, Transmission, Manual, Entertainment, CD, KeylessEntry, PowerLock}
P12{Car, Transmission, Manual, Entertainment, Radio, KeylessEntry, PowerLock}
P13{Car, Transmission, Manual, Entertainment, CD, Radio, KeylessEntry, PowerLock}
P14{Car, Transmission, Automatic, Entertainment, CD, KeylessEntry, PowerLock}
P15{Car, Transmission, Automatic, Entertainment, Radio, KeylessEntry, PowerLock}
P16{Car, Transmission, Automatic, Entertainment, CD, Radio, KeylessEntry, PowerLock}
P17{Car, Transmission, Manual, PowerLock}
P18{Car, Transmission, Automatic, PowerLock}
P19{Car, Transmission, Manual, Entertainment, CD,PowerLock}
P20{Car, Transmission, Manual, Entertainment, Radio, PowerLock}
P21{Car, Transmission, Entertainment, CD, Radio, PowerLock}
P22{Car, Transmission, Automatic, Entertainment, CD, PowerLock}
P23{Car, Transmission, Automatic, Entertainment, Radio, PowerLock}
P24{Car, Transmission, Automatic, Entertainment, CD, Radio, PowerLock}

Feature diagrams can be expressed in terms of logic. To represent the semantics of a feature diagram using logic (i.e., translating every kind of feature relationship into logic), some formal definitions will be introduced.

Definition 1 (Feature model). A feature model FM = (F, Φ) consists of a set of propositional variables $F = \{f_1, \ldots . f_n\}$ called features, and a propositional formula Φ over the variables in F.

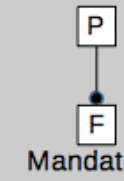
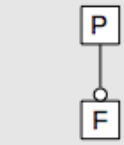
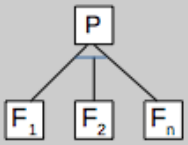
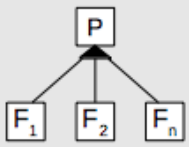
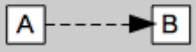
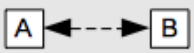
The propositional formula Φ in the feature model is used to determine the possible and allowed combinations of feature values which make the formula true. The set of selectable features from a feature model should be based on its semantics [35, 40]. A feature model shows not only the features, feature attributes, structural relationships and dependencies, but also the overall constraints (typically inclusions or exclusion statements). In a feature diagram, one way to define constraints is by using its hierarchy. This kind of constraints is also called tree constraints as they are expressed by the tree structure [9]. For the basic tree constraints, the root feature must be included in all products. Also, the selection of a feature implies the selection of its parent features (parent-child relationship). In addition, cross-tree constraints (arbitrary propositional formula based on a set of existing

features) can be used to express relationships between features that are not related by a parent-child relationship.The use of cross-tree constraints further restrains configuration options.

Definition 2 (Constraint). Constraints are defined as propositional formulas, using the following regular connectives: ($\wedge$ (conjunction), $\vee$ (disjunction), $\veebar$ (exclusiveor), $\barwedge$ (nand), $\Longrightarrow$ (implication), $\Longleftrightarrow$ (equivalence), and $\neg$(negation) [35, 34, 41].

Cross-tree constraints relate features in different levels and are typically the Requires or Exclusion statements [41], but can be any formula. Requires constraint is used if a feature A requires a feature B. The inclusion of feature A in a product implies the inclusion of feature B (e.g., keylessEntry→power-lock). For the car example, the keylessEntry feature requires the feature power-lock. Exclusion constraint is applied if a feature A excludes a feature B. Both features cannot be a part of the same product. The typical symbols

Table 2.1: Expressing a feature diagram with logic

Relationship	PL Mapping	Car Example
 Mandatory	$P \leftrightarrow F$	Car ↔ Transmission
 Optional	$F \rightarrow P$	Entertainment → Car KeylessEntry → Car PowerLock → Car
 Alternative	$(F_1 \leftrightarrow (\neg F_2 \wedge ... \wedge \neg F_n \wedge P)) \wedge$ $(F_2 \leftrightarrow (\neg F_1 \wedge ... \wedge \neg F_n \wedge P)) \wedge$ $(F_n \leftrightarrow (\neg F_1 \wedge \neg F_2 \wedge ... \wedge \neg F_{n-1} \wedge P))$	(Manual ↔ (¬Automatic ∧ Transmission) ∧ (Automatic ↔ (¬Manual ∧ Transmission))
 Or	$P \leftrightarrow (F_1 \vee F_2 \vee ... \vee F_n)$	Entertainment ↔ (Radio ∨ CD)
 Requires	$A \rightarrow B$	KeylessEntry → PowerLock
 Excludes	$\neg(A \wedge B)$	

for types of feature relationships between a parent and a child node; and the cross-tree constraints are as shown in Table 3.1. The PL mapping column illustrates how the different types of relationships can be expressed with logic and the car example column translates the car feature model example into logic.

Definition 3 (A valid configuration). A subset P $\subseteq F$ is a valid product of a feature model if it includes the root feature and satisfies all feature dependencies in the FM. A valid configuration of the feature model reflects any assignment of boolean values to all features that satisfies the propositional formula [42, 35, 43].
Let (a,b) $\in A$ denote a dependency between feature a and b, the predicates, mandatory, optional, and, or, xor, requires and excludes represents the relationships between a parent (or compound) feature and its child features (or subfeatures)

mandatory(a, b): a $\in P$ implies that b $\in P$.
optional(a, b): a $\in P$ implies that either b $\in P$ or b $\notin P$.
and(a, Y): a $\in P$ implies that $\forall\, bi \in Y : bi \in P$
or(a, Y): a $\in P$ implies that $\exists\, bi \in Y : bi \in P$
xor(a, Y): a $\in P$ implies that ($|Y \cap P| = 1$).
requires(a, b): a $\in P$ implies that b $\in P$ and a != parent(b).
excludes(a, b): a $\in P$ implies that b $\notin P$

A feature configuration is allowed by a feature model if and only if it does not violate constraints imposed on the model. The semantics of a feature model is the set of feature configurations that the feature model permits [38]. Each feature corresponds to a boolean variable and the semantics is captured as a propositional formula. The resulting propositional formula from conjoining the propositional formula from each construct (Table 3.2) in the feature model (Figure 2.2), all cross tree constraints and a formula requiring the root feature describes the semantics of the feature model [38], i.e the valid combinations of features. As seen in Table 3.2, P represents a compound feature with the subfeatures $C_1, C_2, ..., C_n$.

Table 2.2: Feature model translation to propositional formulas

Feature Relationship	Propositional formula
Optional feature (C_i)	$C_i \implies P$
Mandatory feature (C_i)	$(C_i \implies P) \wedge (P \implies C_i)$
Or-group	$P \iff \bigvee_{1 \le i \le n} C_i$
Alternative	$(P \iff \bigvee_{1 \le i \le n} C_i) \wedge \bigwedge_{i<j} (\neg C_i \vee \neg C_j)$

Using the notations and feature names abbreviations (Table 2.3), we give the corresponding propositional formula for our example feature model as:

ϕ_{carPL} =
$(t \implies c) \wedge (c \implies t) \wedge (e \implies c) \wedge (k \implies c) \wedge (p \implies c) \wedge (t \iff m \vee a) \wedge (\neg m \vee \neg a) \wedge (e \iff r \vee d) \wedge (k \implies p) \wedge c$

Table 2.3: Sample car feature model abbreviation.

Feature name	Abbreviation
Car	c
Transmission	t
Entertainment	e
Manual	m
Automatic	a
Radio	r
CD	d
KeylessEntry	k
PowerLock	p

By Definition 1, FM = (car, transmission, entertainment, manual, automatic, ..., (ϕ_{carPL})).

As shown earlier in Figure 2.2, the root feature "Car" is a common feature among all products of the car product line. All cars in our car product line must include a mandatory feature "Transmission", and may include an optional feature "Entertainment, KeylessEntry or PowerLock". Transmission is categorized into "Manual and Automatic", and only one of these can be selected (alternative). Entertainment can have "Radio and/or CD (or)".

To aid in the management of variability in an SPL, feature modelling tools are used. These tools support the representation and management of reusable artifacts. A feature model editor, automated analysis of feature models, product configuration, and tool notation are some of the common functionalities provided by feature modelling tools. With respect to these functionalities, [5] conducted a detailed qualitative analysis of two feature modelling tools, SPLOT[1] and FeatureIDE [37].

2.3 Chatbots

A chatbot is a computer program designed to simulate conversation with human users [44], especially over the internet. As seen in [45], chatbots have the ability to interact with humans and generate reasonable responses. Fundamentally, a chatbot allows a form of interaction between a human and a machine via written messages or voice. It can answer questions formulated to it in Natural Language [14] and comes up with its answers through a combination of predefined scripts and machine learning [23, 44]. The responses may also be generated by combining data coming from calling external information systems [15]. Over time, and multiple interactions, the chatbot gradually gains in scope and relevance. Chatbots have proven useful in automating tasks and improving user experience and productivity [18]. Also, software development support bots (DevBots) are seen as a

[1]http://www.splot- research.org/

promising approach to dealing with the ever-increasing complexities of modern software engineering and development [13]. Examples of some popular chatbots include Amazon's Alexa, Apples Siri, Microsoft's Cortana, etc.

Chatbots are more commonly adopted due to significant progress in the development of platforms and frameworks [23]. Likewise, instant messaging platforms have been widely adopted as one of the main technologies to communicate and exchange information [18, 24]. Chatbots can be integrated in several social networks like Facebook [22, 23], Telegram [18], Twitter [24], Slack [18, 23] or Skype [46]. Examples of some of the tools that have emerged for the development of chatbots (Table 2.4), include DialogFlow[2], the IBM Watson Assistant[3], or the Microsoft Bot Framework[4]. Some offer cloud-based environments to describe the different aspects of the chatbot [47].

Table 2.4: Chatbot creation tools

Tool	DialogFlow	Rasa	Microsoft bot framework	IBM Watson	Chatterbot	Chatfuel	FlowXO
Type	platform	Framework	Framework	platform	Library	Platform	Platform
Multi-language	yes	yes	yes	yes	yes	yes	yes
NLP	yes	yes	yes	yes	yes	yes	no
Support for intents	yes	yes	yes	yes	no	no	yes
Entity support	yes	yes	yes	yes	yes	no	yes
Specifying bot responses	yes	yes	yes	yes	no	yes	yes
Patterns	no	no	yes	yes	no	no	yes
Speech recognition	yes	yes	yes	yes			
Social networks integraton/ websites	yes	yes	yes	yes	yes	yes	yes
Development expertise	low	high	high	low	high	low	low

Table 2.4 compares some chatbot creation tools. These tools fall into the following categories, Platform, Framework and Library. ChatterBot is a Python library designed specifically for building chatbots. Dialogflow, IBM Watson, Chatfuel and FlowXO are chatbot builder platforms. Platforms permit developers and businesses to create bots effortlessly, even with minimal coding basics and bots deployment support. Rasa is an open-source machine learning framework for building chatbots. Overall, these chatbot tools are very powerful. Some like Microsoft Bot Framework, IBM Watson, etc., allow defining patterns or regular expressions for matching user utterances. Others like Dialogflow, chatterbot, IBM Watson or Rasa, require declaring training phrases and apply NLP techniques. Intents, entities, and responses are also supported by some of these tools.

Given the different approaches used by these tools, ranging from low-code form-based platforms to frameworks for programming languages and libraries, it is difficult to determine which tool is suitable for building a specific chatbot, as not every tool supports every possible feature (e.g., only a few provide NLP, multi-language, social network and website integration or speech recognition support). Selecting the most appropriate tool is still a daunting task. In [48], a model-driven engineering approach to chatbot development was proposed. The study describes a neutral meta-model and a domain-specific language (DSL) for the description of chatbots and a platform recommender. The key to selecting the desired tool is awareness and understanding of the technological and operational

[2]https://dialogflow.com/

[3]https://www.ibm.com/cloud/watson-assistant/

[4]https://dev.botframework.com/

capabilities of the chatbot creation tool.

On the whole, the architecture and technology of chatbots are similar. Figure 2.3 shows the technical process of a chatbot operation, built over a set of users intentions. The process begins with a user's request (1), by means of a messenger app like Facebook, Slack, Skype, or text or speech input app like Amazon Echo [15]. For example, with some chatbot creation tools like DialogFlow or Rasa that require declaring training phrases, by applying natural language processing techniques, the chatbot attempts to match (2) the user utterances (user's input) with the corresponding intents (a generalized form of an utterance).

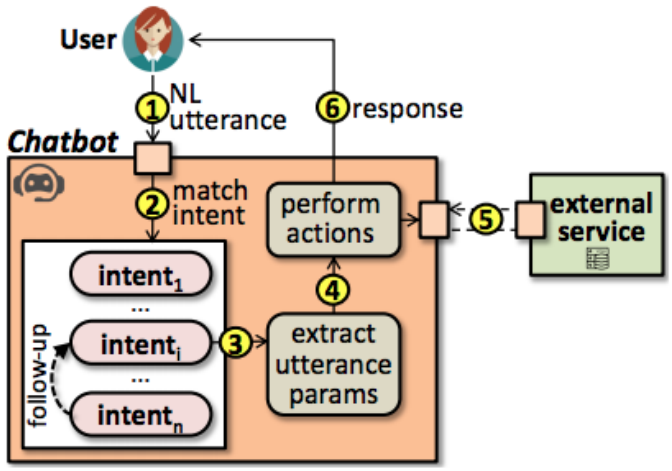


Figure 2.3: Chatbot working scheme [49]

Identifying the intent that conforms to an utterance includes the training phrases (different ways of expressing an intention). Several utterances can lead to the same intent, for example, the utterances "what types of transmission do you have?" and "what are the available transmission?" from the user will match the intent "Transmission".

The chatbot examines the utterance and aims to identify appropriate keywords. Among keywords are some with special meaning for the dialog; these are called entities. Entities are used to add details to intents. It defines a list of possible values (e.g., transmission, entertainment along with synonymns) and are therefore important to the chatbot when deciding on an appropriate response.

Consequently, if a matching intent is found, the chatbot will extract the parameters of interest from the utterances (e.g., types of transmission, (3)). Parameters may be typed by entities (a piece of information that has a specific meaning within a user utterance), which can be either predefined (e.g., dates, colour, currency, email addresses, number, etc.) or user-defined (e.g., transmission type, entertainment type). Also, if the utterance lacks some expected parameters, the chatbot can be configured to prompt the user for the required input. In the event where no matching intent is found, a default fallback intent approach could be adopted.

The chatbot can perform different actions (4) depending on the intent, such as calling

an external service (5) and sending a response to the user (6). The basic response format is text, but some platforms for the deployment of chatbots like telegram and twitter, also support images, URLs, videos or buttons [49].

2.3.1 Natural Language Processing

Natural Language Processing is a branch of computer science, artificial intelligence, and linguistics that enables computers to understand, interpret and manipulate human language (speech or text) [50, 51, 52, 53]. NLP is centered on the aspect of human computer interaction [52]. NLP process and analyze written or spoken text by breaking it down, comprehending its meaning, and determining the appropriate action. It involves parsing, sentence breaking, and stemming and many more tasks.

Some of the researches done in NLP include information extraction, text summarization, tokenization, topic extraction, named entity recognition, parsing, speech recognition, speech generation, language modelling and many other language-related tasks [54, 53, 50].

According to [24], advances in NLP have enabled the proliferation of chatbots that run on social networks and offer services to users upon NL requests, thereby mimicking human responses. Natural Language Processing is what allows chatbots to understand user utterances and respond accordingly.

2.3.2 DialogFlow

Dialogflow provides a cloud-based development environment to describe chatbots with voice and text-based conversational interfaces, as well as support for NL processing in more than 20 languages [47]. It uses machine learning to understand what users are saying[5] and is focused on domain knowledge. Dialogflow provides a framework to build conversational experience powered by artificial intelligence. It has automated support for deploying the bot in mobile apps, web applications, device (such as mobile phones, wearables and other smart devices), interactive voice response systems[6], many different social networks, integration with external services, and support for uploading chatbot description in JSON format (Figure 2.4).

The typical communication pattern for a chatbot implemented in Dialogflow is as described in Section 2.3 and as illustrated in Figure 2.3. Designing a chatbot in Dialogflow, requires the following basic components:

Intent definition: intent is the idea or message that a user intends to convey to the chatbot. An intent matches the user's interaction with an intention. A basic intent is defined by training phrases (different ways of expressing an intent), parameters (pieces of information that the application needs to work), actions (used to trigger specific logic in services) and responses (return responses after the intent is matched). For example,

[5]https://cloud.google.com/dialogflow/es/docs/agents-settings
[6]https://cloud.google.com/dialogflow/docs

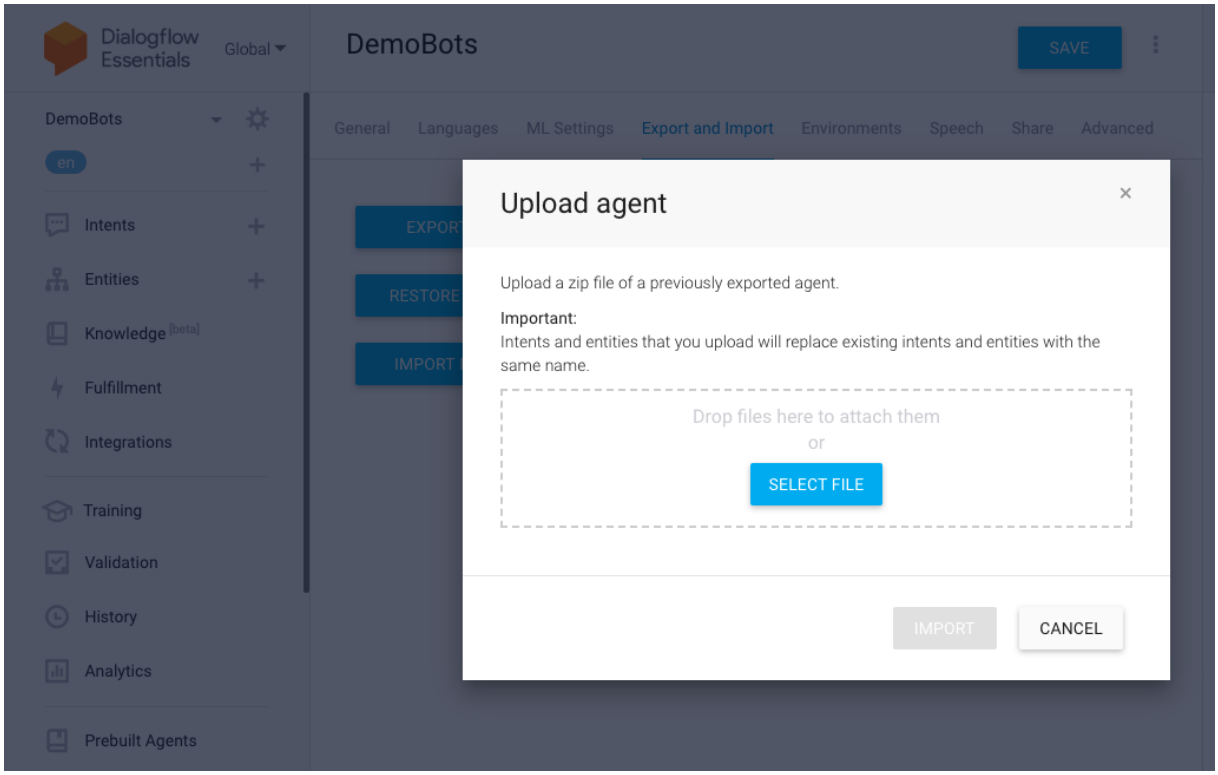


Figure 2.4: A sample screenshot of Dialogflow

if a chatbot receives the user's utterance "what type of entertainment is available?", a chatbot for the sample car product line would recognize that the user intention is obtaining information about the availability of some kind of entertainment, and would reply with a list of entertainment types.

A more complex intent may contain contexts (stores information about a conversation state; such as the values of parameters, in order to reuse it in subsequent intents, e.g., the desired type of entertainment) and events (triggers intents directly by using fulfilment or API without matching training phrases within the conversation). Each intent is assigned possible responses based on keywords. Each intent may have an input and output context which lasts for a specified period (by default, it expires automatically after 20 minutes). However, the life span of the context can be modified.

Intents may have zero or more follow-up intents that can only be activated right after the parent intent has been activated [47]. A fallback intent is usually triggered if a user's input does not match any of the available intents, with a predefined set of responses typically pointing the user to available alternatives. A follow-up intent uses context to connect two intents. In this case, the input context of a follow-up intent is the same as the parent intent's output context. It is only matched after the initial intent is being matched and can be used to match a yes or no answer to some specific questions. By making follow-up intents specific to a single intent, it prevents the accidental matching of any yes or no answer given elsewhere in the conversation. In order for the bot to identify and process the intent that corresponds to the utterance, it would require intent definition, entity definition and bot training with training phrases.

Entity definition: An entity is like an enumeration, and provides a type for the parameters expected by intents in user utterances. it is the major data of intent. Entities are the mechanism used by Dialogflow to identify and extract data from the users' NL inputs [47]. In Dialogflow, training phrases may contain parameters, whose types are given by entities. In the example training phrase, "what type of entertainment is available?", entertainment is the parameter. Entities automatically identify and extract the type of information (parameters) from user utterances. Whatever the user says that triggers the entity becomes the parameter value. Entities can be either predefined (e.g., name, number, text, date, time, colour, etc.) or chatbot-specific. Some entity entries may have several words or phrases that are considered equivalent; one reference value and one or more synonyms for these entity entries.

Action definition: A bot can perform several actions such as sending a text message, sending an image, database queries or doing an external HTTP service request. The action definition, assists in executing logic in the user defined service. Whenever an intent is matched at runtime, Dialogflow provides the action value to fulfilment webhook (an HTTP request that is sent automatically whenever certain criteria is fulfilled) request or API interaction response. This can be used to trigger specific logic in the user defined sevice.

Fulfilment: Fulfilment is a code that is deployed as a webhook; allowing Dialogflow agents to call business logic on an intent-by-intent basis. A more dynamic response to a matched intent can be obtained by using fulfilment. When fulfilment is enabled for an intent, Dialogflow responds to the intent by communicating with the server or database to generate dynamic responses. Whenever an intent with fulfilment enabled is matched, Dialogflow sends a request to a webhook service with information about the matched intent, the action, the parameters, and the response defined for the intent. A webhook is essentially an HTTP push API or web call back, providing an efficient connection and communication between the bot and the web services. The service performs the required actions and sends a webhoook response message to Dialogflow. Dialogflow then sends the response to the end-user.

Flow definition: Dialogflow uses the concept called context to understand which question is being answered and the direction of the conversation flow. It represents the current state of a user's request and allows the agent to carry information from one intent to another. A context acts as a filter and applies a bias to intent matching. For each intent, multiple input and outputs context can be defined.

2.4 CONGA DSL

CONGA (ChatbOt modelliNg lanGuAge) is a textual DSL for chatbot modelling, that was designed based on an analysis of 15 popular chatbot development tools [48]. Chatbot modelling and validation are possible with the CONGA DSL, which is independent of any development platform.

CONGA DSL also provides an extensible recommender that analyzes the chatbot

model and other requirements to generate a ranked list of appropriate development tools, making it easier to choose a development tool for implementing a particular chatbot model. Also, the CONGA DSL's code generator synthesizes chatbot implementations from chatbot models for specific development tools (for example, JSON configuration files for Dialogflow or Python programs and configuration files for Rasa) [48]. Finally, the generated chatbots can then be deployed and made available to users on various platforms (for example, Telegram, Twitter, or Slack).

CONGA supports creation of multi-language chatbots and the definition of intents, entities, and actions, as well as the use of flows to structure conversations. In CONGA, intents can be defined using regular expressions or by a set of training phrases demonstrating common ways for users to express the intention. Training phrases may include Parameters (relevant data that the chatbot requires).

Parameters are formally declared by providing a name, a type, can be a list, can be required, and may define a list of prompts to ask for a value when the parameter is required but the user utterance does not include its value. [55].

Parameters are typed by entities, which can be predefined entities (like "date", "number", or "time") or chatbot-specific ones (class Entity). Chatbot-specific entities can be simple entities, which are defined as a list of words and their synonyms, or composite entities, which are made up of other entities and text [48].

Chatbots can perform a variety of actions, including sending a Text response to the user, which requires specifying the actual text for each chatbot language; sending an Image identified by its URL; performing a HttpRequest to a given URL, optionally providing some headers and data; and sending to the user a HttpResponse for a previous http request. In CONGA, these actions can be configured in the "actions" section.

Finally, the "flows" section allows for the definition of the conversation flows. These are sequences of user intents that are followed by chatbot actions. Flows can be of any length, and there may be multiple user continuations following a chatbot action.

2.5 Related Work

In the configuration of software product lines, the complexity of variability handling poses a significant challenge. As a result, several works [56], [57], [58], [59], [60], [61], [62], [63], [64], [12], [65], [66], [67], have proposed various approaches to reducing the complexity of the configuration process by automating the feature model configuration activity. This section discusses a number of related works in this field.

Visualization. Pleuss et al. [57] presented visual and interactive techniques for reducing (cognitive) complexity and assisting users during the product configuration. The visualized data is the variability. The visualization method is dependent on the underlying data and the task to be supported. Clustering, decision trees, tree-maps, cone trees, tables, flow maps, and UML diagrams are examples of visualization techniques that have

been used or investigated in product configuration in PLE. Other visualization techniques (Data Maps, Cause-effect Chains, Flow Charts, Elastic Lists, Semantic Networks, Venn Diagrams, Cheops, EncCon Trees, Space Trees, and Hyperbolic Trees) that could be useful to support product configuration but have not yet been tested were also proposed.

The strength and weakness of the visualization technique were measured by the dominant aspects it visualizes and the emphasis placed on these aspects [57]. However, no optimal technique was identified based on ratings of how well they support various aspects of variability models relevant for visualization (a) representing hierarchy, (b) visualizing cross-cutting dependencies, (c) visualizing attributes, and (d) visualizing the configuration workflow.

Similarly, Pereira et al. [56] proposed and integrated a set of interelated visualizations to improve the efficiency of product configuration as part of the FeatureIDE tool. Their approach was found to reduce the configuration effort and the complexity of decision making by providing a restricted view of the configuration space and by assisting the decision-makers to reason on a focused set of relevant information about features and non-functional properties.

Interaction. Botterweck et al. [60], [68] designed the S2T2 feature configuration tool, which combines a visual interactive representation of the feature model with a formal reasoning engine that calculates the consequences of the user's actions and provides formal explanations. The reasoning engine uses the mappings between visual elements and their corresponding formal representations to calculate consequences and explanations, which are then communicated in the interactive representation.

Pleuss et al. [58] builds on previous approaches to visualization and interaction by providing a concept for interactive visual tool support for system configuration using feature models and discussed techniques for interactive configuration support based on a reasoning engine, which, for example, ensures the validity of configurations. Their findings were illustrated by the S2T2 Configurator. The primary focus of their interactive support is visualization. The interaction method has a significant impact on visualization efficiency [57], [61].

Collaboration. Chen et al. [69] introduced the use of negotiation as a new methodology for facilitating interactive decision making in product configuration. The configuration process was modelled as a collaborative design problem, and negotiation support system functionality was incorporated into the configurator design.

Pereira [70] proposed a collaborative-based runtime approach that relies on recommender techniques to provide users with accurate and scalable configurations. The proposed approach takes advantage of a simplified view of the configuration space by dynamically predicting the importance of the features and employs a collaborative-based recommender system that learns about the relevant features from the configurations of other users. The set of interactive and automatic visual mechanisms assists users in making valid configuration decisions.

Feature Extraction Approaches From Natural language Specification. Finally, several authors proposed feature model extraction from documents written in natural

language [71], [40]. These approaches provide a set of possible solutions to extract (a) only features [72],[71], (b) only relationships (detecting binary constraints among features, which identifies only requires or excludes relationships among pairs of features) [73] and (c) both feature and feature relationships (candidate feature models) [74], [40], [75], from the requirements documents.

Sree-Kumar et al [40], identified some limitations in previous work on feature model extraction and addressed them with the proposed framework (FeatureX), for extracting FMs from natural language specifications of software product lines. The proposed framework improved recall (the absence of false positives) while maintaining a comparable level of precision (detecting the most relevant features).

Our approach focuses on using chatbots to effectively guide the configuration process using natural language.

3

Approach

The main concepts and approach proposed in this work will be explained in this chapter. Section 3.1 looks into the general approach to chatbot synthesis for software product line configuration. Section 3.2 describes how to map a feature model to a corresponding bot, and Section 3.3 describes the conversation flow plan.

3.1 General Approach

In this project, we propose the use of conversational agents in the configuration of software product lines based on feature models, and we present an approach for chatbot generators for SPLs, based on the motivation presented in chapter 1. The scheme of our proposal is as shown in Figure 3.1. To begin, a feature model is provided (like the one in section 2.2). Then, to map a feature model to a bot, we developed a set of feature modelling specific algorithms as well as heuristics for defining configuration tasks and conversation flow plans that can be used to provide automated product configuration support. The root node and all features from the input feature model must first be identified. Candidate relationships among the features are also identified, allowing for the distinction between a parent feature and its child features. Additionally, any cross-tree constraints between the extracted features are identified. Using the relationship types Mandatory, Optional, And, Alternative/XOR, Or, Include/Require, and Exclude, the information retrieval technique detects meaningful relationships in the context of feature models.

Based on the mapping rules in section 3.2, an output bot model is created using the information extracted from the input (feature model). Intents, entities, actions, and flows are the main components of the bot model, as shown in Figure 3.1.

We rely on FeatureIDE for loading a feature model, information extraction, and analysis on feature models.

The CONGA DSL is used to generate a Dialogflow chatbot, according to the bot definition. Finally, the generated chatbots can then be deployed and made available to users on various platforms (for example, Telegram, Twitter, or Slack).

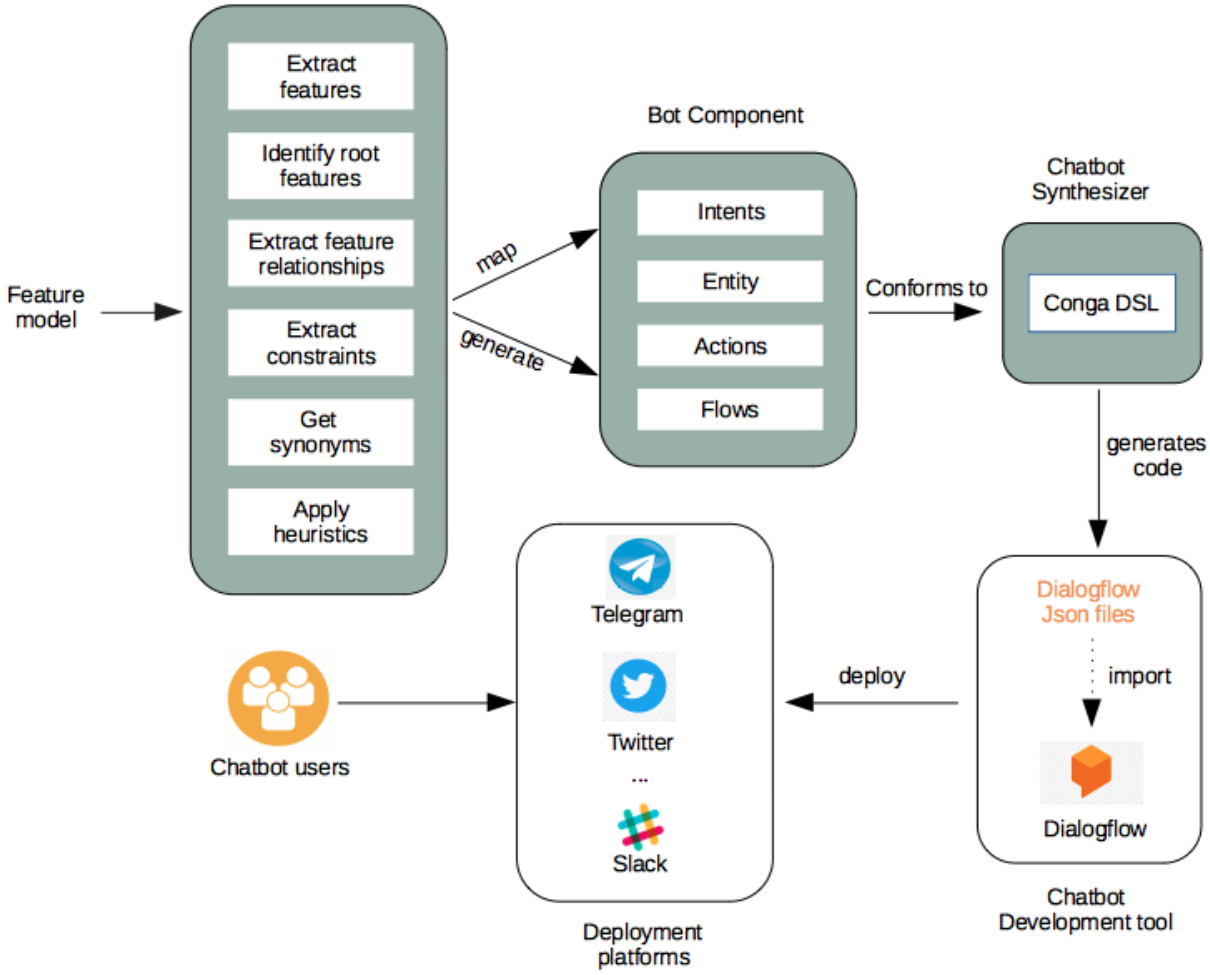


Figure 3.1: Overview of our proposal

3.2 Feature Model Mapping to Corresponding Chatbot

As we discussed in chapter 2, a feature model specifies the domain, product model constraints, and which features are included in the final products. We also discussed the semantics of FM and the structural relationship categories of a typical FM in section 2.2. We now adhere to its layout to realize the feature model mapping to a bot. The candidate features relationship type, parent-child hierarchy between features, and available constraints are identified and extracted from the loaded feature model.

Table 3.1, displays the data obtained from the implemented interfaces for the sample car feature model and associated rules from section 2.2. The information extracted from the feature model is critical for the bot model and the implementation corresponds to the sets and algorithm we defined.

Table 3.1: Information Extraction

Feature	Node Type	Relationship	Constraints
Car	Root	Mandatory	
Transmission	Parent	Mandatory, Alternative	
Entertainment	Parent	Optional, Or	
KeylessEntry	Child (of root)	Optional	Requires PowerLock
PowerLock	Child (of root)	Optional	
Manual	Child (of transmission)	Alternative	
Automatic	Child (of transmission)	Alternative	
Radio	Child (of Entertainment)	Optional	
CD	Child (of Entertainment)	Optional	

Algorithm 1 describes the steps for extracting features and relationships heuristics for intent generation. Detecting the intents to be created and what kind of phrases to generate is based on the position of the occurrence of the features, the hierarchy, and the relationship between the features. Intents are generated for the nodes that are linked to the starting node (root). These are the first-level nodes (these include nodes with sub-features and leaf nodes). Intents are also created for any subsequent node that is not an end-node. As a result, intents will be generated for the Transmission, Entertainment, KeylessEntry, and PowerLock nodes, but not for the automatic, manual, cd, and radio nodes.

The purpose, or goal, expressed in a user's utterance is represented by an intent. The intent should be a concise method of categorizing the utterance tasks. To assign some example utterances to intents, the possible user utterances were generated based on the keywords or intents. The manually entered phrases by the developer become the set of predefined phrases for the intents. Some of the intents required concatenating the phrases with the extracted feature names. Several example phrases that mean the same thing were used as training phrases for each intent. The intent of the utterances "I want Manual" and "Give me Automatic," for example, is to choose a transmission feature; thus, a Transmission intent related to these utterances is defined in the bot description. Furthermore, the relationship type (AND, OR, and XOR) influences the type of phrases generated. An optional or mandatory relationship with a set of features is represented by an AND connection by default in the FeatureIDE tool [37].

Algorithm 1 Intent generation

Input: A feature model
Output: A bot file with Sets of Intent definitions

```
1: for each f ∈ FM do
2:     f ← IdentifyCandidateFeatures                ▷ detect feature names
3:     if f ≠ Root then
4:         IdentifyCandidateRelationship(Alt, or, and)
5:         TF ← AddPredefinedTrainingPhrases(f)
6:         P ← AddParameter
7:         if f = Parent(fp) then
8:             for each c ∈ fp do
9:                 TFc ← AddPredefinedTrainingPhrases
10:            end for each
11:        if f ≠ mandatory then
12:            TF ← AddPredefinedTrainingPhrases     ▷ optional features
13: end for each
```

Algorithm 2 Entity generation

Input: A feature model
Output: A bot file with Sets of Entity definitions

```
1: for each f ∈ FM do
2:     IdentifyCandidateRelationship(Alt, or, and)
3:     if f = Parent(fp) then
4:         ENT ← SetSimpleEntity
5:         for each c ∈ fp do
6:             Generate Entity with synonyms
7:         end for each
8: end for each
```

Algorithm 2 provides the steps for generating simple entity definitions. User-defined entities were used. Entities were created to automatically identify and extract the type of information (parameters) from user utterances. A list of words or phrases that fits the given context was provided from the extracted features, and synonyms assigned to the lists. To create an entity for any feature, WordNet (a database of words and synonyms) [76] is queried for synonyms, and if none are found, the original word is returned.

For each set of features that are children of the same parent feature, a simple entity is generated. For example, simple entity transmission is generated for manual and automatic, since they are both children of transmission. Furthermore, using the CONGA syntax, the manual and automatic entities in the simple entity transmission example in Listing 3.1 can be viewed as specific instances of a general transmission entity. The synonyms are generated automatically using the dictionary, WordNet.

```
 entities:
Simple entity "transmission":
 inputs in en{
"Manual" synonyms "MT", "manual gearbox"
"Automatic" synonyms "auto", "automatonlike"
}

Simple entity "entertainment":
 inputs in en{
"CD" synonyms "compact disk", "compact disc"
"Radio" synonyms "radio receiver", "radio set", "tuner"
}

Simple entity "powerlock":
 inputs in en{
"PowerLock"
}

Simple entity "keylessentry":
 inputs in en{
"KeylessEntry"
}
```

Listing 3.1: Entity definition

Chatbots, as described in section 2.3, can perform a variety of tasks. Corresponding actions are defined in response to user input. The actions and text responses indicate whether the relationship is required or optional, as well as whether it is AND, OR, or Alternative. In comparison to the mandatory feature, the text responses for optional features do not express obligation. The Transmission and Entertainment features text responses example exemplifies this. The standard action (text response) for the required Transmission feature is "Transmission must be configured. Please choose only one of the following features: Manual or Automatic" (Listing 3.2). However, for the optional feature Entertainment, it is "Do you want to add any of these optional features: Entertainment..? (Listing 3.3). The response for Entertainment, which is an "Or" feature, is "You can select one or more of the following features: CD, Radio" (Listing 3.4).

```
actions:
text response TransmissionType:
 inputs{
"Transmission must be configured.
Please choose only one of the following features:
Manual, Automatic"
 }
```

Listing 3.2: Compound Mandatory Feature Response

```
actions:
text response OptSelectPrompt:
 inputs {
 "Do you want to add any of these Optional Features:
 KeylessEntry, Entertainment, PowerLock?"
}
```

Listing 3.3: Optional Feature Response

```
actions:
 text response EntertainmentType:
 inputs{
"You can choose one or more of the following features:
CD, Radio"
 }
```

Listing 3.4: Compound Or Response

There are also phrases to inform users about feature dependencies (cross-tree constraints). The predeterminer "all" is used to represent an implication [40]. If KeylessEntry is selected, for example, the corresponding action is: "PowerLock is generally provided for all cars with KeylessEntry feature." The word "all" denotes an implication in this context. This indicates a "requires" relationship between PowerLock and KeylessEntry.

Finally, in this work, conversation flows (i.e., sequences of user and chatbot interactions) have been divided into three paths: configuration, analysis, and help as shown in Figure 3.2). When a user initiates a conversation with the bot, it responds with the bot's functionalities, which are configuration, help, and analysis. After the user agrees to proceed with the configuration, the bot walks the user through the feature selection process, starting with the core features and working its way down to the optional features. Section 3.3 describes how the chatbot aids in the feature selection process. Listing 3.5, 3.6, 3.7, 3.8 3.9 and 3.10 demonstrate how to configure the feature model using conversations (user utterances and bot responses).

Listing 3.5, 3.6, 3.7, 3.8, and 3.9 are general intents created for initiating a conversation with the bot. These intents are different from the specific intents generated from the feature model. The conversation begins with a greeting from the user, as shown in Listing 3.5. The bot responds with a welcome message (Listing 3.6) and a follow-up response (Listing 3.7), inquiring if the user wants to proceed with the configuration process. Listing 3.8 shows the utterances for proceeding with the configuration process, and Listing 3.9 provides information about the configuration process. Listing 3.10 is an example of a specific intent with some information from the feature model (automatically selected features).

```
 intents:
Greeting:
 inputs {
"Howdy",
"Hi",
"Hello",
"Hey",
"Heya",
"hey there",
"good morning",
"good afternoon",
"good evening"
}
```

Listing 3.5: User Initiates Conversation

```
actions:
text response GreetingRes:
 inputs {
"Hello there! I am here to help you with configuring your software
   ↪ product line.", "Hi, welcome to SPLBot!", "Hello there! I am here
   ↪ to help you with configuring your software product line. To get
   ↪ information about this SPL, enter help for more detail."
}
```

Listing 3.6: Bot Welcome Response

```
text response StartRes:
 inputs {
"Would you like to proceed to product configuration?", "Would you like
   ↪ to proceed to feature selections?", "Would you like to select a
   ↪ configuration?"
}
```

Listing 3.7: Bot Follow-up Respone

```
Config:
 inputs {
"Yes",
"yes I want to select a configuration",
"yes I would like to choose a configuration",
"ys what features do you have",
"yes what features are available",
"yes I'd love to",
"why not",
"yes I do"
}
```

Listing 3.8: User Accepts Configuration Request

```
text response ConfigRes:
 inputs {
 "Great!
Please note that the "Root" and "Core features" are automatically
   ↪ selected, but you will have the option to select from the
   ↪ available optional features."
 }
```

Listing 3.9: Bot Configuration Response

```
text response PreSelectRes:
 inputs {
 "The pre-selected features are: Car, Transmission"
}
```

Listing 3.10: Bot Configuration Response Follow-up

The next step is to create a bot file, which includes the bot language and name definition, intents, entities, actions, and flows definition. CongaDSL can then automatically process the generated bot file to generate a Dialogflow chatbot.

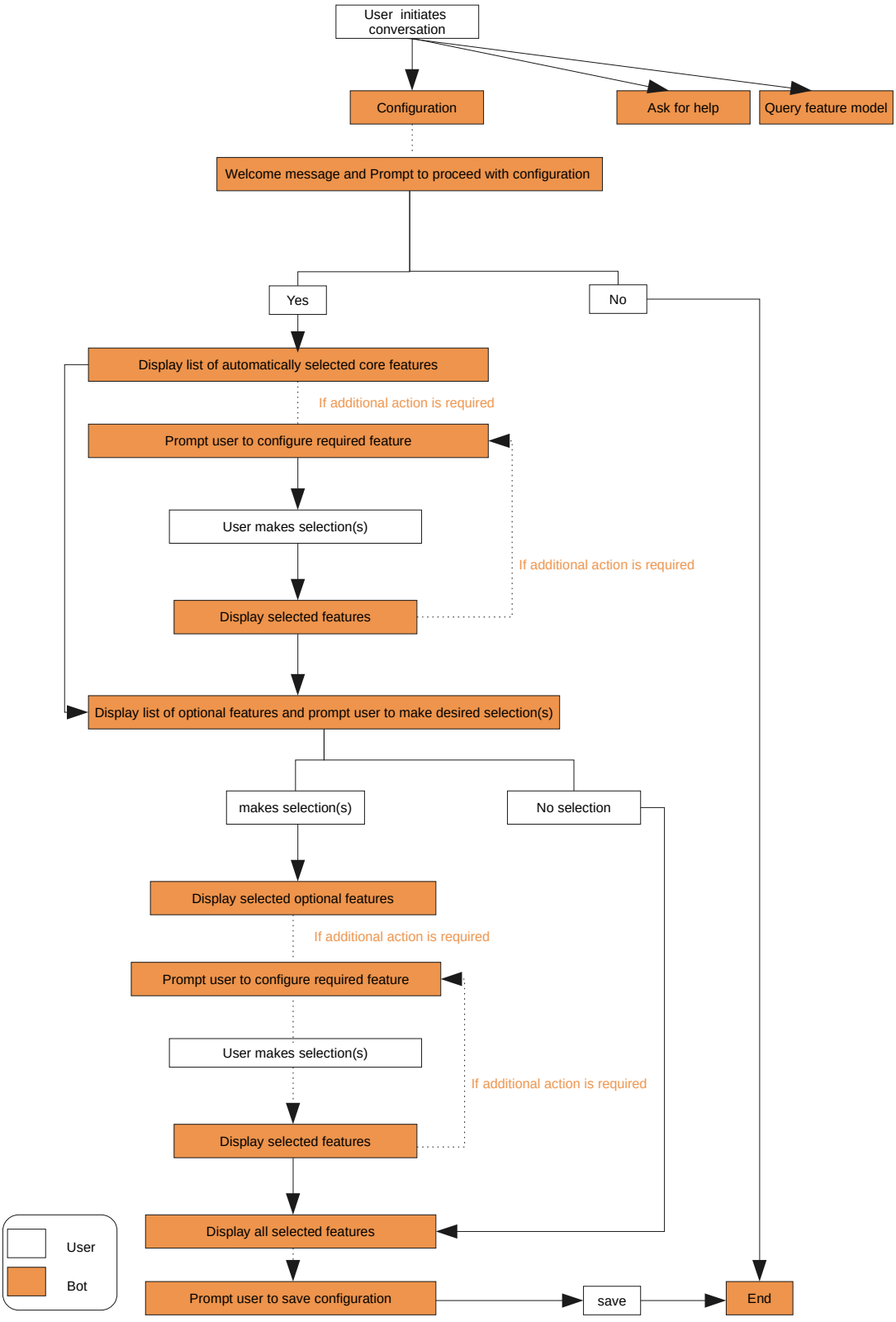


Figure 3.2: Conversation flow

3.3 Heuristics

The conversation flow plans are divided into three categories: configuration, general help, and analysis. The mandatory and optional feature categories are used to define the configuration flow plan. The chatbot facilitates the configuration tasks by guiding the user through the mandatory and optional feature selections. We use the hierarchical structure of the FM, as well as the relationship between the features, exactly as they appear in the model, for the configuration flow. We can extract the parent-child relationship in this manner. The core assets for any feature must be present in all products of an SPL, whereas the optional features are optionally included in all products. As a result of this analogy, the core features become the configuration's defining point.

The relationship type rule differentiates between required and optional features. The core features that have been automatically selected are presented to the user first, based on the mandatory and optional features identified, the position of occurrence, and the presence of compound features. The user is then required to make a secondary selection for first-level compound core features. These are parent features that necessitate the selection of additional child features (Alt, Or, And). For example, because Transmission is both a preselected and a compound feature, a user would be required to select one of transmission's child features. After completing all mandatory selections, the user is presented with a list of optional features from which to choose. All automatically selected and user selected features are displayed as text responses after a user is done with selections and prompted to save configuration.

3.4 Additional Functionalities

The chatbot is also able to provide support for analysis using FeatureIDE's analysis services functionality. This is done during the code generation process, and presented as possible queries (intents) and corresponding bot responses. The various analyses that this chatbot can perform are detailed in Table 3.2.

Listing 3.11 and Listing 3.13 are example query structure to obtain the number of features in the FM and possible configurations. Listing 3.12 and Listing 3.14 are sample responses. The results of the queries can be viewed by sending a text message to the Telegram chat and other platforms. For "All products" output which are sometimes very long, they will be sent in a text file to the conversation.

```
QFeatureNum:
 inputs {
"What is the total number of features?",
"How many features are available?",
"How many features do you have",
"How many features?"
}
```

Listing 3.11: Number of features Intent

Table 3.2: Sample Analysis Queries on FM

Input: Feature Model	
Void/valid features model Query: is the feature model valid?	This query returns a value indicating whether or not the input feature model is void.
Number of products Query: What is the number of products?	This query returns the number of products represented by the input feature model.
All products Query: List products	The output of this query is a text file containing a list of all the products represented by the feature model.
Core Features Query: What are the core features?	Returns the set of features common to all products in the software product line.
Variant/Optional features Query: What are the optional features?	Returns the set of variant features in the model (i.e those features that do not appear in all the products of the SPL.
Number of features Query: What is the number of features?	This query returns the number of features in the input feature model.

```
text response QNoOfFeatRes:
 inputs {
 "There are 9 features in the feature model"
}
```

Listing 3.12: Bot Response to Number of features Query

```
QConfigNum:
 inputs {
"What is the possible number of configuration?",
"How many configurations are possible?",
"How many configurations can I get?",
"Number of valid configurations?",
"number of configurations",
"How many configurations are possible?",
"How many configuration?",
"How many products?"
}
```

Listing 3.13: Number of Possible Configurations

```
All Products:
P1:  Car, Transmission, Manual
P2:  Car, Transmission, Automatic
P3:  Car, Transmission, Manual, Entertainment, CD
P4:  Car, Transmission, Manual, Entertainment, Radio
P5:  Car, Transmission, Manual, Entertainment, CD, Radio
P6:  Car, Transmission, Automatic, Entertainment, CD
P7:  Car, Transmission, Automatic, Entertainment, Radio
P8:  Car, Transmission, Automatic, Entertainment, CD, Radio
P9:  Car, Transmission, Manual, KeylessEntry, PowerLock
P10: Car, Transmission, Automatic, KeylessEntry, PowerLock
P11: Car, Transmission, Manual, Entertainment, CD, KeylessEntry,
    ↪ PowerLock
P12: Car, Transmission, Manual, Entertainment, Radio, KeylessEntry,
    ↪ PowerLock
P13: Car, Transmission, Manual, Entertainment, CD, Radio, KeylessEntry,
    ↪ PowerLock
P14: Car, Transmission, Automatic, Entertainment, CD, KeylessEntry,
    ↪ PowerLock
P15:  Car, Transmission, Automatic, Entertainment, Radio, KeylessEntry,
    ↪ PowerLock
P16:  Car, Transmission, Automatic, Entertainment, CD, Radio,
    ↪ KeylessEntry, PowerLock
P17:  Car, Transmission, Manual, PowerLock
P18:  Car, Transmission, Automatic, PowerLock
P19:  Car, Transmission, Manual, Entertainment, CD,PowerLock
P20:  Car, Transmission, Manual, Entertainment, Radio, PowerLock
P21:  Car, Transmission, Entertainment, CD, Radio, PowerLock
P22:  Car, Transmission, Automatic, Entertainment, CD, PowerLock
P23:  Car, Transmission, Automatic, Entertainment, Radio, PowerLock
P24:  Car, Transmission, Automatic, Entertainment, CD, Radio, PowerLock
```

Listing 3.14: Configurations Response

The user can seek assistance from the chatbot. Listing 3.15 depicts possible user utterances. The format for the bot's assistance is as follows: first, it explains how to select a configuration, and then it explains with examples how to query the bot about the feature model analysis.

```
Help:
 inputs {
"Can you help me",
"I need help",
"what can you do"
}
```

Listing 3.15: Help

4

Architecture and Tool

This chapter describes the architecture of SPLBOT (section 4.1), as well as how to use the tool (section 4.2) on Telegram.

4.1 Architecture

SPLBOT has been implemented in Java. Figure 4.1 depicts the various components of SPLBOT that will be discussed in this section. The implementation relies on FeatureIDE libraries for information extraction from feature models and the analysis on feature models, as well as on the CONGA DSL for the synthesis of the chatbot using DialogFlow. The bot model was created using the described mapping rules (section 3.2) and flow heuristics (section 3.3).

The definition of the chatbot model should be consistent with the CONGA DSL model [48]. The output bot file can then be processed by the CONGA DSL to generate a Dialogflow chatbot.

Nodejs was used to create the server. The chatbot employs a webhook architecture. Dialogflow is in charge of processing the user's natural language utterances. When Dialogflow receives user input, it interprets it and sends it to the server in the form of a POST request. These interpretations are analyzed on the server, and appropriate responses are sent to Dialogflow, which is then sent to Telegram.

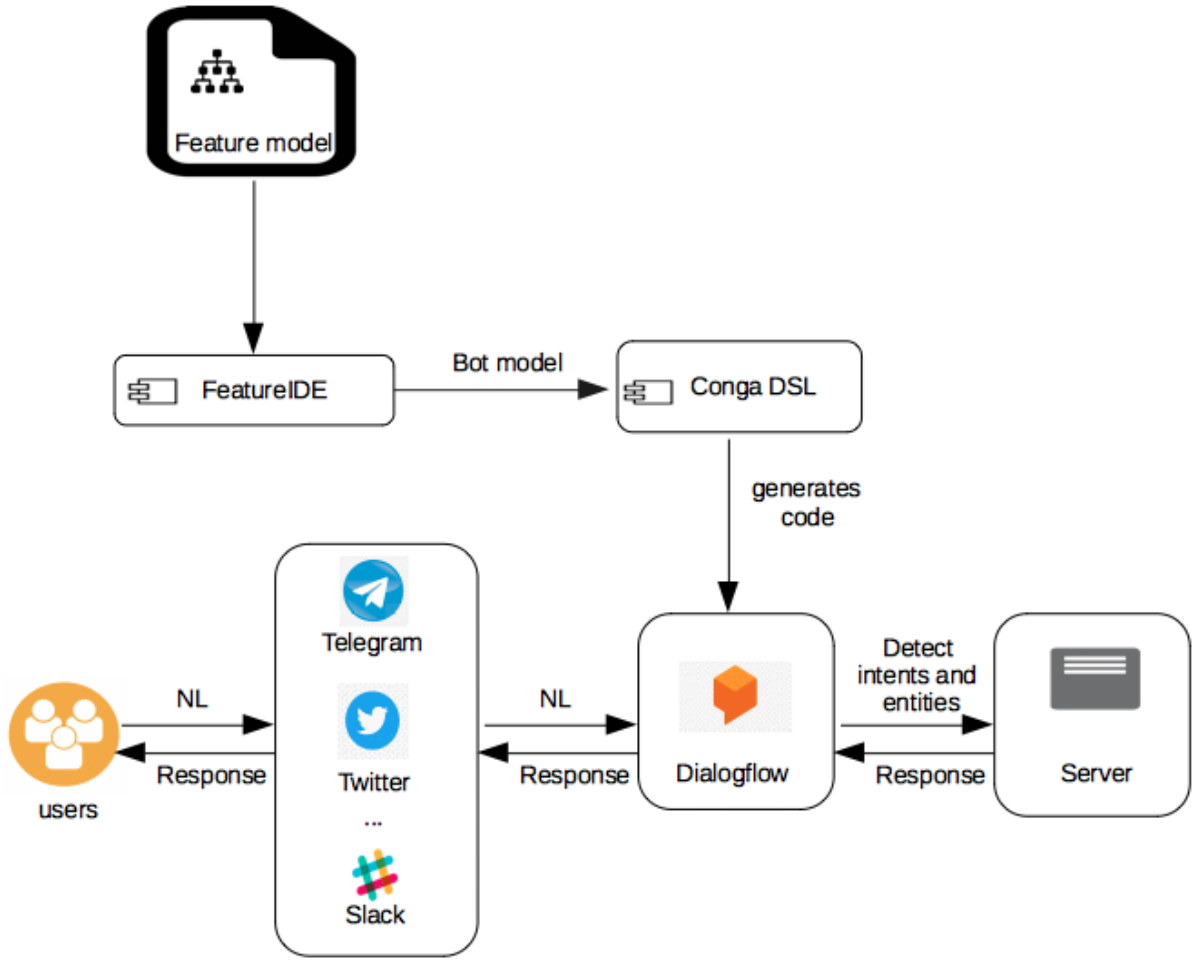


Figure 4.1: Architecture of SPLBOT

4.1.1 FeatureIDE

The FeatureIDE libraries [77], are used to load the feature model as objects in memory and to extract the necessary information from the feature model. The interfaces IFeatureModel, IFeature, and IConstraint are used to identify and extract the candidate feature names, features relationship type, parent-child hierarchy between features, and available constraints from the loaded feature model. FeatureIDE encourages the use of external feature model implementations by providing interfaces to FeatureIDE's classes for feature model, feature, and constraint, as well as a factory framework for creating concrete instances.

To use an external feature model, feature, or constraint within FeatureIDE, the corresponding interfaces was implemented. As well as a factory implementing IFeatureModelFactory that instantiates the corresponding classes. Additionally, FeatureIDE was made aware of the new classes before the classes were used. FeatureIDE's fm.core plug-in was added and the factories were added programmatically during runtime by calling the method addExtension of the factory manager and an instance of the implemented factory

was provided. During runtime, additional associations of factories to file paths or file formats were created and modified during runtime. The FeatureIDE was also used for the automated analysis of feature models support. The operation takes a set of parameters as input and returns a result as output.

4.1.2 CONGA for Chatbot Synthesis from Feature Model

This section explains the structure of the generated CONGA chatbots.

```
1  Chatbot CarBot language: en
2
3   intents:
4  Greeting:
5   inputs {
6    "Howdy",
7    "Hi",
8    "Hello",
9    "Hey",
10   "Heya",
11   "hey there",
12   "good morning",
13   "good afternoon",
14   "good evening"
15 }
16 Help:
17  inputs {
18   "Can you help me",
19   "I need help"
20 }
21 ConfigureIntent:
22  inputs {
23   "Yes",
24   "yes I want to select a configuration",
25   "yes I would like to choose a configuration",
26   "ys what features do you have",
27   "yes what features are available",
28   "yes I'd love to",
29   "why not",
30   "yes I do"
31 }
```

Listing 4.1: Intent definition

The declared language for this work is English (en), but multi-language chatbots are also possible with CONGA. The first line defines the chatbot name (CarBot in Listing 4.1) and the supported languages (English). The name of the bot is defined by the root node (Car), which represents the concept that the feature model is characterizing.

Chatbots can define intents, entities, actions and flows (to structure the conversation). In CONGA, intents can be defined using regular expressions or a set of training phrases demonstrating common ways for users to express the intention. Listing 4.1 (lines 4–15,

16-20, and 21-31) shows intent definition examples for the generated bot.

Training phrases may contain parameters, which are relevant data that the chatbot requires, such as the entertainment and transmission ("entertainment_type" and "transmission_type" in Listing 4.2, lines 1-2 and 24-25). Lines 11-22 define an intent named "Get_Transmission" with a set of training phrases. The intent defines the parameters in lines 24-25. The training phrases can refer to them (for example, [Transmission_type] in line 25) and assign a value to them in the context of the phrase (e.g., automatic in line 19).

```
1  parameters:
2  Entertainment_type: entity entertainment, isList, required, prompts ["
       ↪ What type of Entertainment?"];
3
4  Transmission:
5   inputs {
6    "I want to know the type of Transmission that is available",
7    "Transmission",
8    "See available Transmission",
9    "what Transmission type is available"
10 }
11 Get_Transmission:
12  inputs {
13
14   ("Manual")[Transmission_type],
15   "I want" ("Manual")[Transmission_type],
16   "I'll go with" ("Manual")[Transmission_type],
17   "Give me" ("Manual")[Transmission_type],
18   ("Automatic")[Transmission_type],
19   "I want" ("Automatic")[Transmission_type],
20   "I'll go with" ("Automatic")[Transmission_type],
21   "Give me" ("Automatic")[Transmission_type]
22 }
23
24 parameters:
25 Transmission_type: entity transmission, required, prompts ["What type of
       ↪  Transmission?"];
```

Listing 4.2: Sample parameters for intent definition

Each parameter is formally declared by specifying its name, type, can be optional or required, can be a list, and may define a list of prompts to ask for a value when the parameter is required but the user utterance lacks its value (lines 2 and 15).

Parameters are typed by entities. CONGA supports both predefined entities and chatbot-specific entities. Chatbot-specific entities can be simple entities, which are defined as a list of words and their synonyms, or composite entities, which are made up of other entities and text. The definitions of the simple entities "transmission," "entertainment," "powerlock," and "keylessentry" are shown in Listing 3.1. This declares the admissible transmission, entertainment, powerlock, and keylessentry together with their synonyms.

Listing 4.3 illustrates action definition and text responses (TransmissionRes, Enter-

tainmentRes, TransmissionType, EntertainmentType...). Parameter values can be used in text responses. For example, [“Get_Transmission.Transmission_type”] in line 5 for Transmission selection and [“Get_PowerLock.PowerLock_type”] in line 33 for powerLock selection.

```
1  actions:
2  text response TransmissionRes:
3   inputs {
4   "Selected feature: "
5   ["Get_Transmission.Transmission_type"]
6  }
7
8  text response TransmissionType:
9   inputs{
10 "Transmission must be configured. Please choose only one of the
       ↪ following features:
11 Manual, Automatic"
12  }
13  text response EntertainmentRes:
14  inputs {
15  "Selected feature: "
16  ["Get_Entertainment.Entertainment_type"]
17 }
18
19 text response EntertainmentType:
20  inputs{
21 "You can choose one or more of the following features:
22 CD, Radio"
23
24  }
25
26 text response PowerLockRes:
27  inputs {
28  "Selected feature: "
29  ["Get_PowerLock.PowerLock_type"]
30 }
31
32 text response OptSelectPrompt:
33  inputs {
34  "Do you want to add any of these Optional Features: KeylessEntry,
       ↪ Entertainment, PowerLock?"
35 }
```

Listing 4.3: Action definition

Finally, a chatbot can define conversation Flows (i.e., sequences of user and chatbot interactions). A flow is made up of UserInteraction objects that are linked to an intent and BotInteraction objects that include one or more actions. A flow must begin with a user interaction, followed by a bot interaction, which may be followed by more user interactions, and so on. Listing 4.3 is an example of a conversation flow. It specifies a number of flows, each of which must begin with a user interaction and the associated intent. When the user utterance matches the Greeting intent, the flow in line 2-19 occurs, and the chatbot asks for confirmation to proceed or not to proceed with the configuration.

The flow is divided based on the user's response (yes or no). When the user utterance matches the Help intent, the flow in lines 22 occurs, and the chatbot provides a response to the user.

```
1  flows:
2  - user Greeting => chatbot GreetingRes, StartRes{
3   => user Config => chatbot ConfigRes, PreSelectRes,Type{
4
5    =>  user "Mandatory" => chatbot CoreFeatSelectRes, TransmissionType{
6    => user "Get_Transmission" => chatbot TransmissionRes, OptSelectPrompt
       ↪ {
7    => user OptYes => chatbot OptYesRes;
8    => user OptNo => chatbot SaveConfigRes{
9  => user SaveYes => chatbot SaveRes;
10 => user SaveNo => chatbot NoConfigRes;
11    };
12
13 };
14   };
15
16 };
17 => user NoConfig => chatbot NoConfigRes;
18
19 };
20
21
22 - user "Help" => chatbot HelpCRes, HelpARes;
23
24 - user "QFeatureNum" => chatbot QNoOfFeatRes;
25
26 - user "QOpt" => chatbot QOptFeatRes;
27
28 - user "QCore" => chatbot QCoreFeatRes;
29
30 - user "QConfigNum" => chatbot QConfigRes;
31
32 - user "QProducts" => chatbot QProductsRes;
33
34 - user "QCoreIsValid" => chatbot QIsValidRes;
```

Listing 4.4: Flows definition

The flow in line 24 occurs when the user utterance matches the QFeatureNum intent, in which case the chatbot executes the QNoOfFeatRes action defined in Listing 4.5, lines 6-9. When the user utterance matches the QOpt intent, the flow in line 26 occurs, and the chatbot executes the QOptFeatRes action defined in Listing 4.5, lines 14-17. The flow in line 28 occurs when the user utterance matches the QCore intent, in which case the chatbot executes the QCoreFeatRes action defined in Listing 4.5, lines 10-13. When the user utterance matches the QConfigNum intent, the flow in line 30 occurs, and the chatbot executes the QConfigRes action defined in lines 18-22.

```
text response ConfigRes:
 inputs {
 "Great!
Please note that the "Root" and "Core features" are automatically
    ↪ selected, but you will have the option to  select from the
    ↪ optional features."
 }
text response QNoOfFeatRes:
 inputs {
 "There are 9 features in the feature model"
}
text response QCoreFeatRes:
 inputs {
 "The following features are automatically selected: Car, Transmission"
}
text response QOptFeatRes:
 inputs {
 "You can select any of these Optional Features: KeylessEntry,
    ↪ Entertainment, PowerLock"
}
text response QConfigRes:
 inputs {
 "There are 24 possible configurations.
 The possible solutions are: [Car, Transmission, Manual], [Entertainment
}
```

Listing 4.5: Responses

The DSL code generator [48] is used to generate the Dialogflow platform's codes. Java was used to implement this. The file structure is made up of Json files that define the intents, entities, and conversation flow. The generated files are then uploaded as a zip file to the Dialogflow platform. The chatbot is then integrated into the Telegram platform. Finally, the bot was deployed on the Telegram platform.

4.1.3 Dialogflow

This section, show Dialogflow screenshots of some of the generated intents (general and specific) for the chatbot. Each intent serves a distinct purpose and displays sample user utterances in the Training phrases (Figure 4.2) section of Dialogflow, as well as the associated Text response (Figure 4.3) in the Response section. When a user types a query, Dialogflow detects the intent and responds appropriately.

After the user has agreed to proceed with the configuration, the generated intent provides the user with information on the feature to be configured. The response exemplifies both the obligatory and alternative relationships, as well as the parent-child relationship. In Figure 4.3, the response "the pre-selected features are Car and Transmission" illustrates that the root node and the mandatory children of the root node have to be selected.

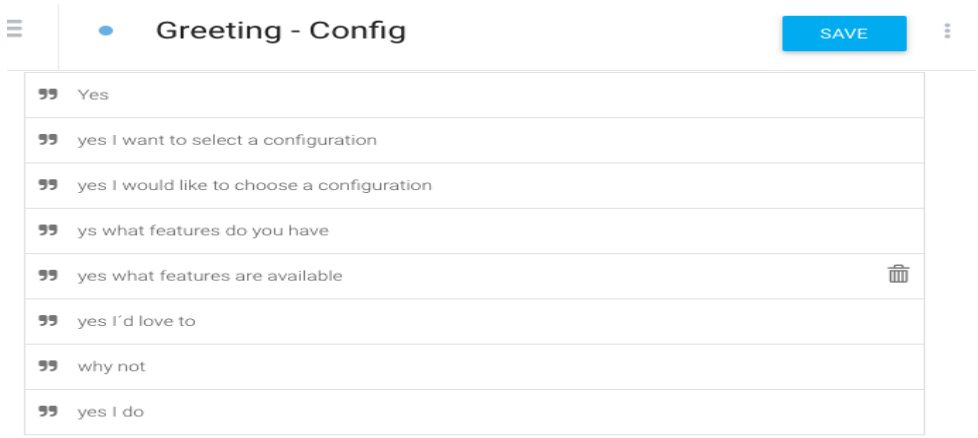


Figure 4.2: Training Phrases of the "Config Intent"

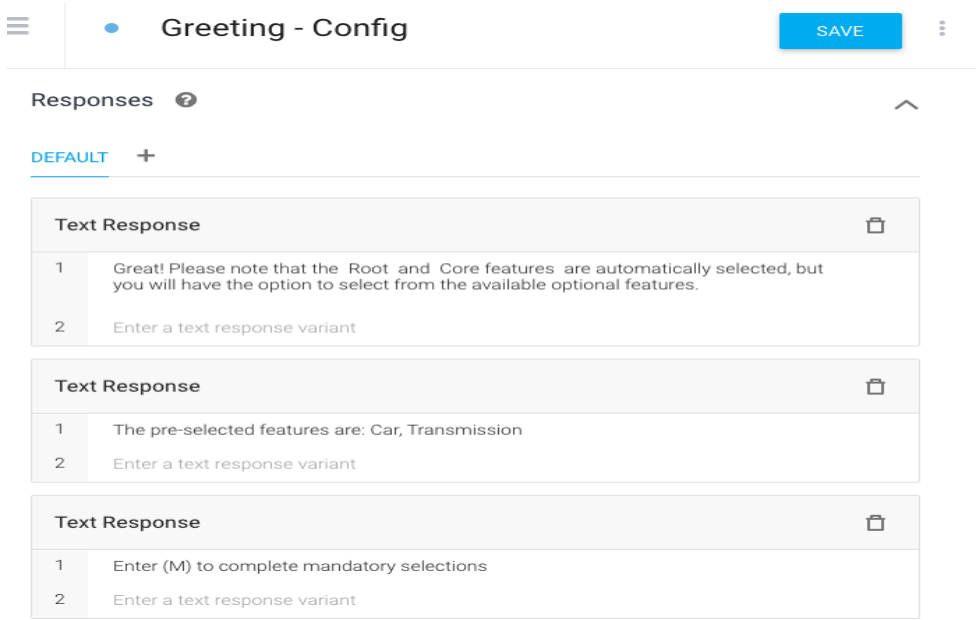


Figure 4.3: Bot Response for the "Config Intent"

The "Get_Transmission" intent, for example, recognizes the user's intention to select a transmission type. The context(Figure 4.4) is used to store the user's choices, followed by the training phrases(Figure 4.5), the intent parameter (Figure 4.6), and finally the bot's responses (Figure 4.7).

The Action and parameter section of the intent in Figure 4.6 captures the defined parameters in the Entities, "@transmission". These are the set of keywords that can be extracted as a parameter. This is useful for obtaining specific data from a user's utterance and is selected from the Entity-type that best meets the requirements.

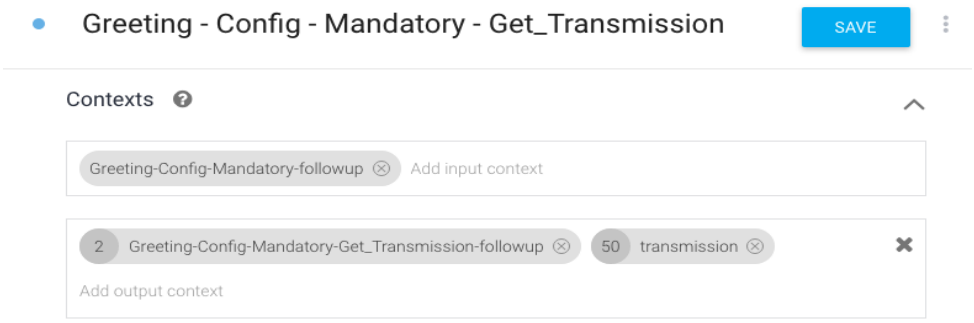


Figure 4.4: Context for the "Get_Transmission Intent"

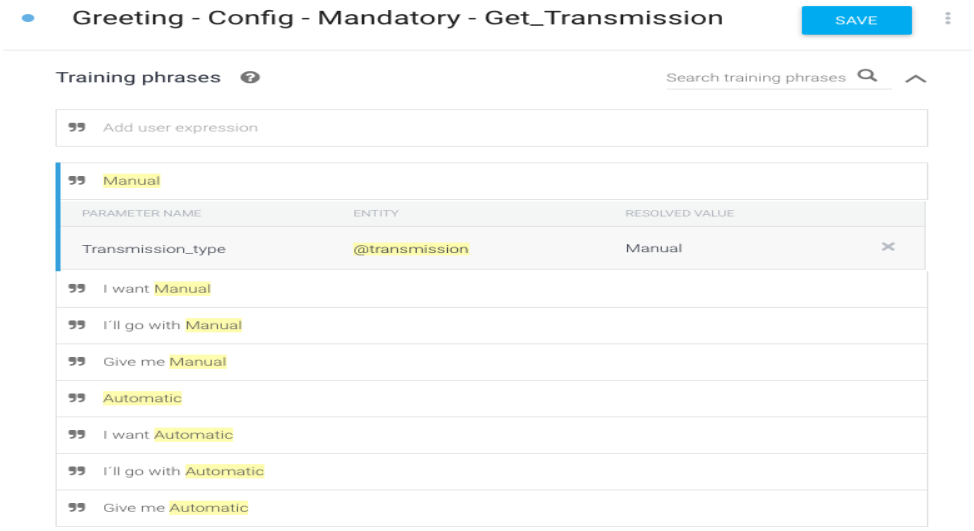


Figure 4.5: Training Phrases of the "Get_Transmission Intent"

The defined entity in Figure 4.5 is "transmission," with the entry "manual" and the defined synonyms manual, MT. Another entry is "automatic," with synonyms auto and AT defined. When the user says, "I want manual," Dialogflow recognizes the reference and extracts the parameter as "Transmission type," which is then displayed in the response, as shown in Figure 4.7. With Fulfillment and webhook enabled, a service was deployed to respond to some of the user inquiries. A URL for the back-end is provided in the Webhook section, and Dialogflow sends the user's query and Parameters to the back-end as a POST request.

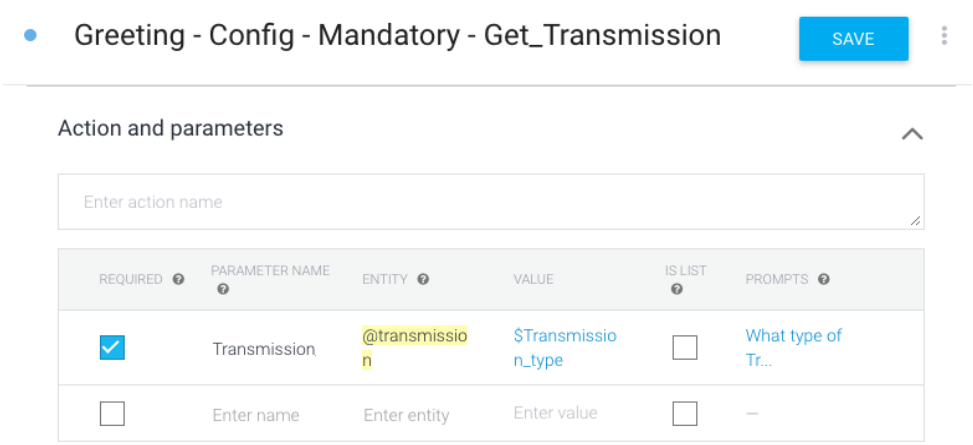


Figure 4.6: Parameter of the "Get_Transmission Intent"

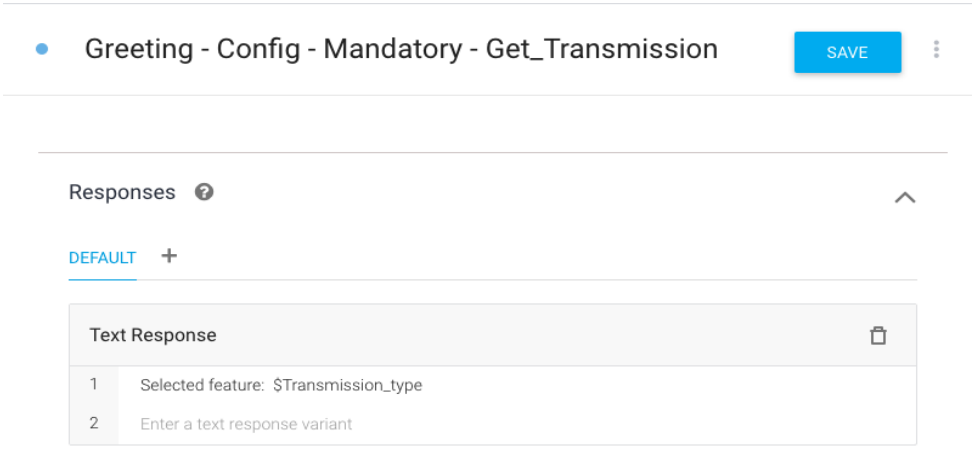


Figure 4.7: Bot Response for the "Get_Transmission Intent"

4.2 Tool

This section describes how users interact with the generated chatbot through a social network. The user will converse in English using the Telegram messaging application.

4.2.1 Using the generated chatbot in Telegram

Telegram is a cloud-based messaging service. Users can interact in private chats with an individual, bot, or group with up to 200,000 members. Telegram has a Bot API, a developer platform that enables anyone to build specialized tools (e.g., bots) for Telegram and integrate any services. Bots can be run directly within Telegram using the Telegram Bot API. The generated chatbot integrated into Telegram allows users to use natural language to converse with and configure a software product line. It also supports queries, enabling users to obtain necessary information about the SPL. Here are some user-bot interaction examples on the Telegram platform.

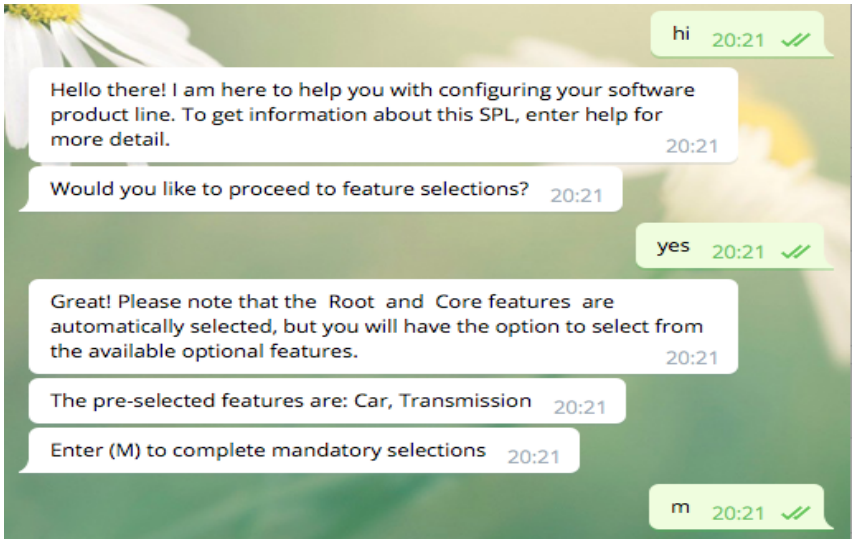


Figure 4.8: Beginning of conversation with the bot

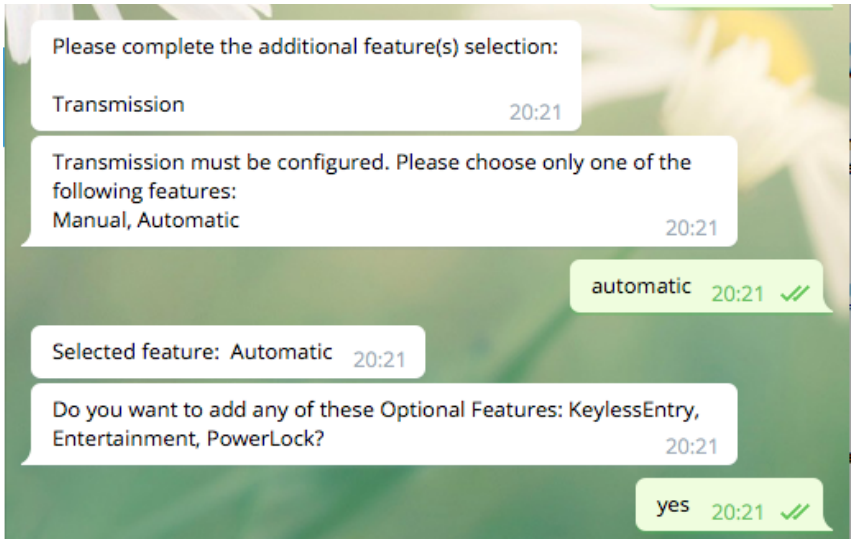


Figure 4.9: Interaction with chatbot for selection of features

Figure 4.8 depicts how the user initiates the interaction with the bot. Then, Figure 4.9 illustrates the way in which the user enters the feature to be selected for core features requiring additional feature selection. In Figure 4.10, the user begins selection of optional features.

Finally, once the user has completed the feature selections, the resulting configuration file in XML format (compatible with FeatureIDE) can be downloaded.

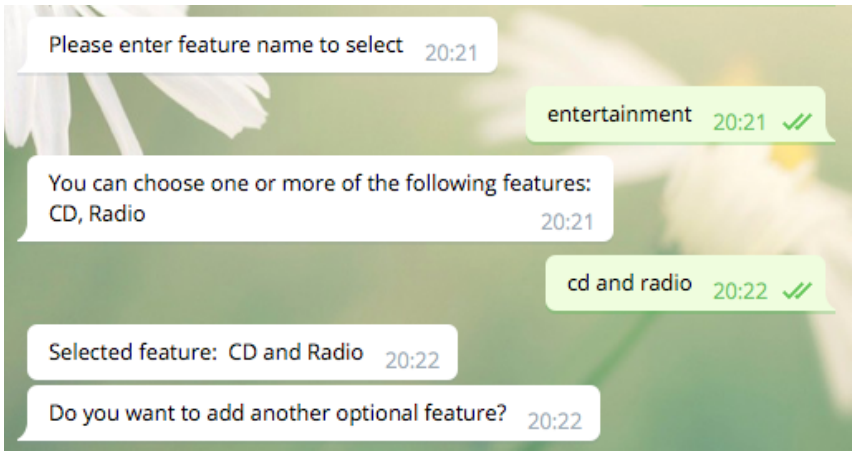


Figure 4.10: User makes optional feature selection

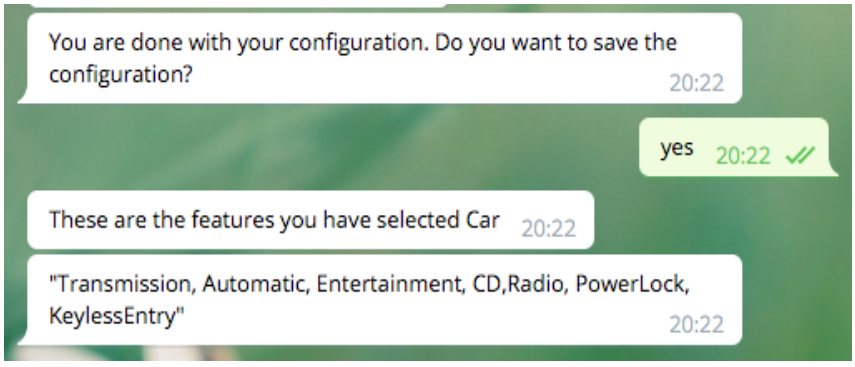


Figure 4.11: User Completes feature selection

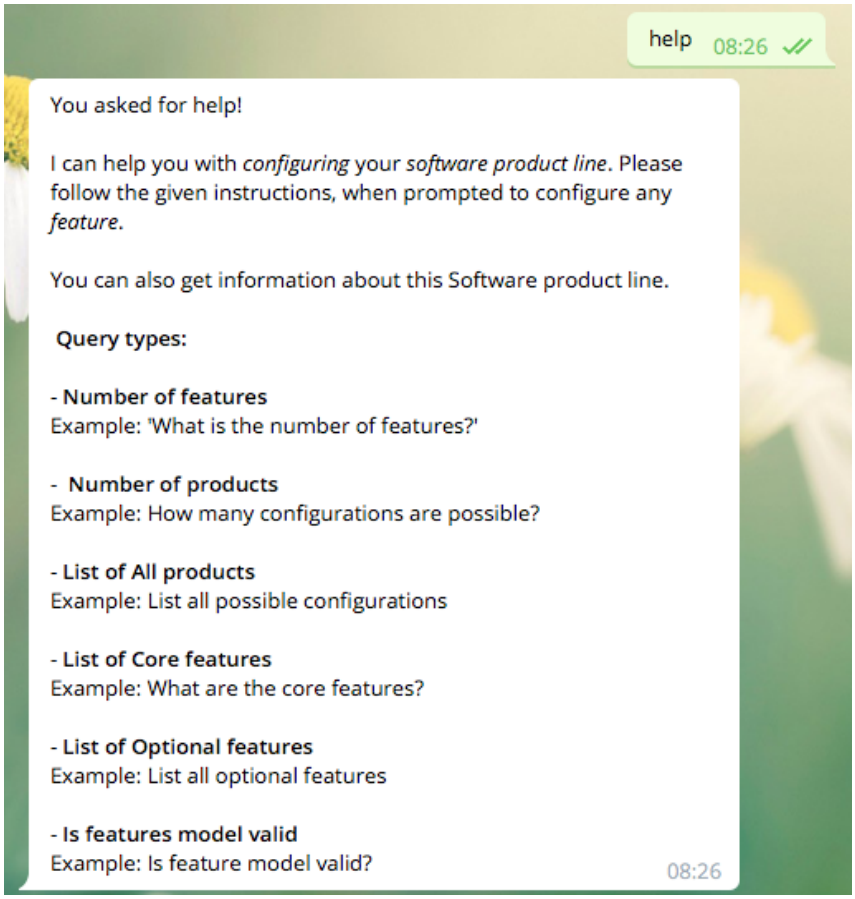


Figure 4.12: User Interaction with chatbot to Request Help

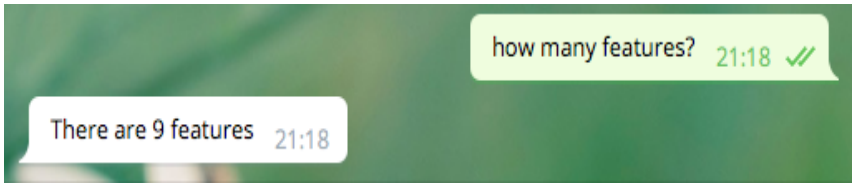


Figure 4.13: User queries chatbot for the number of features

Figure 4.12 shows how the user requests help from the bot, while Figure 4.13 and Figure SPLBOT4.14 depict how the user queries the bot for information on feature model analysis. Figure 4.14 displays the results, along with a link to the text file. The contents of the file are depicted in Figure 4.15.

Our approach reduces configuration effort and complexity by providing conversational guidance support throughout the configuration process. Furthermore, users can interact with the chatbot using natural language in a familiar environment such as Telegram, Facebook, or Slack without having to install any additional software, as is required with FeatureIDE. A non-technical user can interact with the chatbot to configure the product with little or no training.

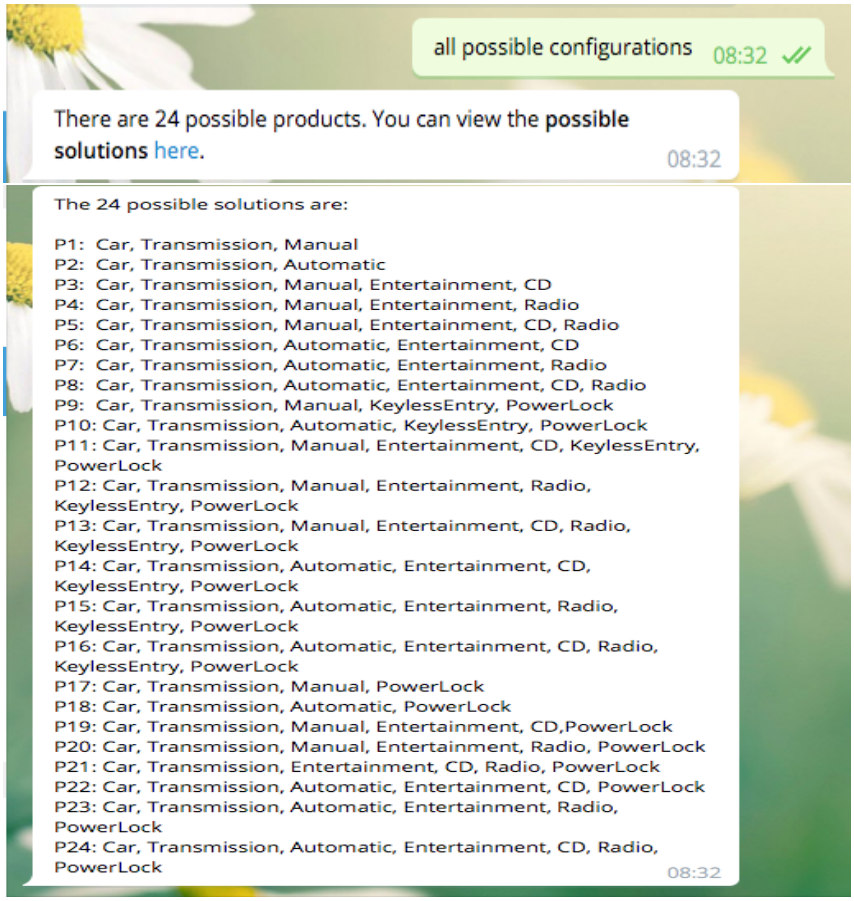


Figure 4.14: User queries chatbot for all possible configurations

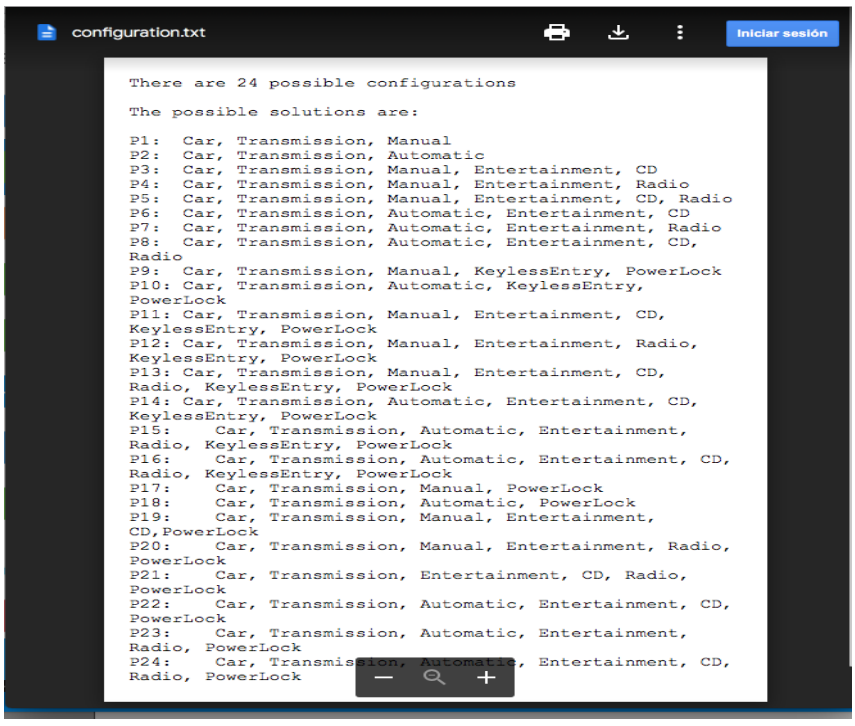


Figure 4.15: File content

5 Evaluation

To assess our approach, we present the findings of an experiment required to address the following two research questions (RQs):
RQ1: (Effectiveness). Is the proposed automated chatbot synthesis from feature models capable of assisting in the configuration of realistic Product Lines?
RQ2: (Scalability). Is our approach suitable for large feature models?

5.1 Experiment set-up

We compare the chatbots generated by our tool using (a) one of our feature models and (b) two feature models from http://www.splot-research.org/. SPLOT is a web-based feature modelling tool [5] that puts Software Product Lines research into practice by providing cutting-edge online tools to academics and practitioners in the field. It supports real-time feature model editing, debugging, analysis, configuration, sharing, and downloading [1].

The Car feature model has 9 features, two of which are mandatory, three of which are optional, one of which is an alternative feature, one of which is an or feature, and one of which is a constraint. The Mobilephone [2] feature model has ten features, two of which are mandatory, two of which are optional, one alternative feature, one or feature, and two constraints. The MyDental [3] feature model has 12 features, 5 mandatory features, 1 optional, and 2 Or feature. Table 5.1 shows a summary of the feature models. All the

[1] http://www.splot-research.org/

[2] `http://ec2-54-213-92-199.us-west-2.compute.amazonaws.com:8080/SPLOT/models/model_20120110_139114401.xml`

[3] `http://ec2-54-213-92-199.us-west-2.compute.amazonaws.com:8080/SPLOT/models/model_20140403_1425048263.xml`

	Feature model properties						
	Features	Mandatory	Optional	XOR groups	OR groups	CTC	Valid configurations
Car	9	2	3	1	1	1	24
Mobilephone	10	2	2	1	1	2	14
MyDental	12	5	1		2		42
eLearning	24	5	8	1	3	2	4608

Table 5.1: Size of the feature models

experiments were carried out on an Intel Core @2.4GHz with 4GB of RAM.

5.2 Evaluation Description

For each of the feature models, we created the corresponding chatbots. We collected the size of the feature models, the number of generated intents, entities, chatbot size, and chatbot generation time during our evaluation.

Table 5.2 summarizes the assessment metrics of the generated chatbots to provide an understanding of the synthesized chatbots. CarBot is used to configure a car product line; MobilephoneBot is used to configure a mobile phone product line; ELearningBot is used to configure an eLearning System product line; and MyDentalBot is used to configure a dental MyDental product line. The CarBot, with a feature model of 985bytes, generated a chatbot with a size of 99,139bytes and a generation time of 5338ms. The MobilephoneBot, with a feature model of 1.082bytes, generated a chatbot size of 39,021bytes in 4928ms. The MyDentalBot, with a feature model of 936bytes, generated a chatbot size of 30,129bytes in 4916ms. This demonstrates the effectiveness of our tool.

Two aspects, however, required manual intervention. First, referencing contexts from other intents; some chatbot responses rely on parameter values, which CONGA does not currently support. Second, the Dialogflow agents had Nodejs backends that were integrated with Dialogflow. These situations necessitated manual intervention. Our flow approach did not favor large feature models because the generated chatbots could not be imported into Dialogflow because the length of the display name exceeded the Dialogflow limit (100). This provides an answer to our second research question about the scalability of our approach. In future work, we intend to conduct additional research with large feature models to overcome this limitation.

	Assessment metrics				
	FM size (bytes)	Number of intents	Number of entities	Chatbot size (bytes)	Generation time (ms)
CarBot	985	25	6	99,139	5338
MobilephoneBot	1.082	25	6	39,021	4928
MyDentalBot	936	27	7	30,129	4916

Table 5.2: Generated chatbots

5.3 Discussion and Threats to Validity

There are several validity threats to the design of this evaluation. In terms of internal validity, we identify the characteristics of the feature models used to assess the efficiency of our approach. We used feature models designed in FeatureIDE and publicly available feature models on SPLOT. The corresponding Dialogflow chatbots for product line configuration were generated automatically from the input feature models.

Our conversation flow options for the product configuration are limited to a single flow pattern. In addition, the flow path for the optional and mandatory features, which required multiple sequences of user-bot interactions, was observed to have impacted the size of the chatbot. We should consider more conversation flow approaches and feature models of varying sizes to expand on this work. It would provide more data and allow for better and more detailed analysis with broader and more realistic coverage of feature model sizes in SPLs.

Our experiments to check the scalability of our approach rely on the size and characteristics of the feature models and the generated bots. These factors could have influenced the outcome. To strengthen this approach, we need to run more experiments with larger feature models using at least two other conversation flow approaches to see how they affect the outcome.

Another limitation of this study is that CONGA currently does not support contexts referenced within another intent. Some contexts have to be set in the Dialogflow console to preserve the parameter values of the user-selected features to overcome this limitation.

Furthermore, the generated Dialogflow chatbot is not deployed automatically; neither is the Telegram bot. The chatbot must be imported into Dialogflow and integrated with a Telegram bot through the Dialogflow console. Also, using the Telegram bot called

BotFather, the Telegram bot is created. However, a guide for importing the Dialogflow agent into the Dialogflow console and creating a bot in Telegram is required to overcome this limitation.

6

Conclusion and Future Work

This chapter ends the thesis with the conclusions (Section 6.1) and open lines for future work (Section 6.2).

6.1 Conclusion

An approach for automated synthesis of chatbots for configuring software product lines has been proposed for this work. By interacting with the conversational agent, we guide the user through the product configuration process. SPLBOT creates a bot model from a given feature model, and CONGA creates a Dialogflow agent that is imported into Dialogflow and deployed on Telegram.

This approach makes use of Dialogflow's natural language processing capabilities, the widespread use of social networks, and the growing use of chatbots for a variety of purposes. The Dialogflow bot is deployed on Telegram to accomplish the configuration process. Other social networks, such as Twitter, Facebook, and Line, can be integrated as well.

By providing conversational guidance support throughout the configuration process, we reduce configuration effort and complexity while also assisting users in making valid configuration decisions. Furthermore, users can interact with the chatbot using natural language in a familiar environment such as Telegram, Facebook, or Slack without the need to install any additional software, as is required with FeatureIDE. A non-technical user can converse with the chatbot to configure the product with little or no technical training. Through these chatbots, users can ask questions about the product line. The recognized query types are those that have been predefined for the chatbot. Our experimental results

show that the proposed method is effective.

6.2 Future Work

The following are the points that are going to be worked on to improve and complete the project:

Explore the use of buttons to select between various feature options and significantly improve the user experience by specifying responses instantly without the need to type. The current version of CONGA does not support button responses, but it is presently under development for the next update.

Add a help function that notifies the user of features that have been configured and those that are yet to be configured. Thus, providing the user with an overview of the configuration process and the number of decisions to make.

Experiment with deploying chatbots on intelligent speakers (such as Google Home) to allow for voice interaction.

Use machine learning to analyse sets of configuration files and recommend selecting some further configurations based on the user's current selections.

Improve the conversation flow approach to support large feature models and draw conclusions about our approach's scalability.

Formally examine the configuration files for correctness and completeness, thereby providing feedback to the user on any violations.

Finally, perform a user study to evaluate the usability of the conversational product configuration process.

Bibliography

[1] Klaus Pohl, Günter Böckle, and Frank Van Der Linden. *Software product line engineering: Foundations, principles, and techniques*. 2005, pp. 1–467. ISBN: 3540243720. DOI: 10.1007/3-540-28901-1.

[2] Krzysztof Czarnecki and Michal Antkiewicz. "Mapping features to models: A template approach based on superimposed variants". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 3676 LNCS. 2005, pp. 422–437. ISBN: 3540291385. DOI: 10.1007/11561347_28.

[3] Ebrahim Bagheri et al. "Configuring software product line feature models based on stakeholders' soft and hard requirements". In: *International Conference on Software Product Lines*. Springer. 2010, pp. 16–31.

[4] Krzysztof Czarnecki et al. "Cool features and tough decisions: a comparison of variability modeling approaches". In: *Proceedings of the sixth international workshop on variability modeling of software-intensive systems*. 2012, pp. 173–182.

[5] Juliana Alves Pereira et al. "Software variability management: An exploratory study with two feature modeling tools". In: *Proceedings - 7th Brazilian Symposium on Software Components, Architectures and Reuse, SBCARs 2013 - In Conjunction with CBSoft 2013 - 4th Brazilian Conference on Software: Theory and Practice*. 2013, pp. 20–29. ISBN: 9780769551562. DOI: 10.1109/SBCARS.2013.13.

[6] Luciano Baresi and Clément Quinton. "Dynamically evolving the structural variability of dynamic software product lines". In: *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE. 2015, pp. 57–63.

[7] A. F. Al Azzawi. *PYFML- A TEXTUAL LANGUAGE FOR FEATURE MODELING*. 2018. DOI: 10.5121/ijsea.2018.9104.

[8] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. "Formalizing cardinality-based feature models and their specialization". In: *Software Process Improvement and Practice* 10.1 (2005), pp. 7–29. ISSN: 10774866. DOI: 10.1002/spip.213.

[9] Jens Meinicke et al. *Mastering software variability with featureIDE*. 2017, pp. 1–243. ISBN: 9783319614434. DOI: 10.1007/978-3-319-61443-4.

[10] A. Spencer Kang, Kyo C; Cohen, Sholom G.; Hess, James A.; Novak, William E.; Peterson. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Tech. rep. Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, 1990. DOI: CMU/SEI-90-TR-21.

[11] Jacob Stein, Ingrid Nunes, and Elder Cirilo. "Preference-based feature model configuration with multiple stakeholders". In: *ACM International Conference Proceeding Series*. Vol. 1. 2014, pp. 132–141. ISBN: 9781450327404. DOI: 10.1145/2648511.2648525.

[12] Sabrine Edded et al. "Collaborative configuration approaches in software product lines engineering: A systematic mapping study". In: *Journal of Systems and Software* 158 (2019), p. 110422. ISSN: 01641212. DOI: 10.1016/j.jss.2019.110422.

[13] Linda Erlenhov et al. "Current and future bots in software development". In: *Proceedings - 2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering, BotSE 2019*. 2019, pp. 7–11. ISBN: 9781728122625. DOI: 10.1109/BotSE.2019.00009.

[14] Nicole Radziwill and Morgan Benton. *Evaluating quality of chatbots and intelligent conversational agents*. Apr. 2017. arXiv: 1704.04579 [cs.CY].

[15] Darius Zumstein and Sophie Hundertmark. "Chatbots-An Interactive Technology for Personalized Communication, Transactions and Services Web Analytics View project Fuzzy Managament Methods View project CHATBOTS-AN INTERACTIVE TECHNOLOGY FOR PERSONALIZED COMMUNICATION, TRANSACTIONS AND SERVICES". In: *IADIS International Journal on WWW/Internet* 15.1 (2017), pp. 96–109. ISSN: 1645-7641. URL: https://www.researchgate.net/publication/322855718.

[16] Yi Fei Wang and Stephen Petrina. "Using Learning Analytics to Understand the Design of an Intelligent Language Tutor – Chatbot Lucy". In: *International Journal of Advanced Computer Science and Applications* 4.11 (2013), pp. 124–131. DOI: 10.14569/IJACSA.2013.041117. URL: http://dx.doi.org/10.14569/IJACSA.2013.041117.

[17] Raphael Meyer von Wolff, Sebastian Hobert, and Matthias Schumann. "How May I Help You? – State of the Art and Open Research Questions for Chatbots at the Digital Workplace". In: *Proceedings of the 52nd Hawaii International Conference on System Sciences*. Hawaii International Conference on System Sciences, 2019. DOI: 10.24251/hicss.2019.013.

[18] Carlene Lebeuf, Margaret Anne Storey, and Alexey Zagalsky. "Software Bots". In: *IEEE Software* 35.1 (2017), pp. 18–23. ISSN: 07407459. DOI: 10.1109/MS.2017.4541027.

[19] Anbang Xu et al. "A new chatbot for customer service on social media". In: *Conference on Human Factors in Computing Systems - Proceedings*. Vol. 2017-May. 2017, pp. 3506–3510. ISBN: 9781450346559. DOI: 10.1145/3025453.3025496.

[20] Lei Cui et al. "Superagent: A customer service chatbot for E-commerce websites". In: *ACL 2017 - 55th Annual Meeting of the Association for Computational Linguistics, Proceedings of System Demonstrations*. 2017, pp. 97–102. ISBN: 9781945626715. DOI: 10.18653/v1/P17-4017.

[21] Martin Adam, Michael Wessel, and Alexander Benlian. "AI-based chatbots in customer service and their effects on user compliance". In: *Electronic Markets* (2020). ISSN: 1422-8890. DOI: 10.1007/s12525-020-00414-7. URL: https://doi.org/10.1007/s12525-020-00414-7.

[22] Juanan Pereira and Oscar Díaz. "A quality analysis of facebook messenger's most popular chatbots". In: *Proceedings of the ACM Symposium on Applied Computing*. 2018, pp. 2144–2150. ISBN: 9781450351911. DOI: 10.1145/3167132.3167362.

[23] Rohan Kar and Rishin Haldar. "Applying Chatbots to the Internet of Things: Opportunities and Architectural Elements". In: *International Journal of Advanced Computer Science and Applications* 7.11 (2016). ISSN: 2158107X. DOI: 10.14569/ijacsa.2016.071119. arXiv: 1611.03799.

[24] Sara Perez-Soler et al. "The rise of the (modelling) bots: Towards assisted modelling via social networks". In: *ASE 2017 - Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*. 2017, pp. 723–728. ISBN: 9781538626849. DOI: 10.1109/ASE.2017.8115683.

[25] Sara Pérez-Soler, Esther Guerra, and Juan De Lara. *Collaborative Modeling and Group Decision Making Using Chatbots in Social Networks*. 2018. DOI: 10.1109/MS.2018.290101511.

[26] Sara Pérez-Soler et al. "Towards automating the synthesis of chatbots for conversational model query". In: *Lecture Notes in Business Information Processing*. Vol. 387 LNBIP. 2020, pp. 257–265. ISBN: 9783030494179. DOI: 10.1007/978-3-030-49418-6_17.

[27] Marcilio Mendonca and Donald Cowan. "Decision-making coordination and efficient reasoning techniques for feature-based configuration". In: *Science of Computer Programming* 75.5 (2010), pp. 311–332. ISSN: 01676423. DOI: 10.1016/j.scico.2009.12.004.

[28] Jules White et al. "Automated reasoning for multi-step feature model configuration problems". In: *Proceedings of the 13th International Software Product Line Conference* (2009), pp. 11–20. URL: http://dl.acm.org/citation.cfm?id=1753235.1753238.

[29] Günter Böckle, Klaus Pohl, and Frank van Der Linden. "A framework for software product line engineering". In: *Software Product Line Engineering*. Springer, 2005, pp. 19–38.

[30] Kyo C. Kang et al. "FORM: A feature-oriented reuse method with domain-specific reference architectures". In: *Annals of Software Engineering* 5.1 (1998), pp. 143–168. ISSN: 10227091. DOI: 10.1023/a:1018980625587. URL: https://doi.org/10.1023/A:1018980625587.

[31] Ricardo De Almeida Falbo, Giancarlo Guizzardi, and Katia Cristina Duarte. "An ontological approach to domain engineering". In: *ACM International Conference Proceeding Series*. Vol. 27. 2002, pp. 351–358. ISBN: 1581135564. DOI: 10.1145/568760.568822.

[32] Danilo Beuche and Mark Dalgarno. "Software Product Line Engineering with Feature Models". In: *Structure* 15.78 (2007), pp. 1 –7. URL: http://www.pure-systems.com/fileadmin/downloads/pure-variants/tutorials/SPLWithFeatureModelling.pdf.

[33] Jan Bosch and Charles Krueger, eds. *Software Reuse: Methods, Techniques, and Tools, 8th International Conference, ICSR-8, Madrid, Spain, July 5-9, 2004, Proceedings*. Vol. 3107. Lecture Notes in Computer Science. Springer, 2004. ISBN: 3–540–22335–5.

[34] Don Batory. "Feature models, grammars, and propositional formulas". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 3714 LNCS. 2005, pp. 7–20. ISBN: 3540289364. DOI: 10.1007/11554844_3.

[35] Krzysztof Czarnecki and Andrzej Wasowski. "Feature diagrams and logics: There and back again". In: *Proceedings - 11th International Software Product Line Conference, SPLC 2007*. 2007, pp. 23–32. ISBN: 0769528880. DOI: 10.1109/SPLINE.2007.4339252.

[36] Eman Muslah and Said Ghoul. "Requirements Variability Specification for Data Intensive Software". In: *International Journal of Software Engineering Applications* 10.2 (2019), 23–34. ISSN: 0976-2221. DOI: 10.5121/ijsea.2019.10203. URL: http://dx.doi.org/10.5121/ijsea.2019.10203.

[37] Christian Kastner et al. "FeatureIDE: A tool framework for feature-oriented software development". In: *2009 IEEE 31st International Conference on Software Engineering*. IEEE. 2009, pp. 611–614.

[38] Thomas Thum et al. "Abstract features in feature modeling". In: *2011 15th International Software Product Line Conference*. IEEE. 2011, pp. 191–200.

[39] Thomas Thüm et al. "FeatureIDE: An extensible framework for feature-oriented software development". In: *Science of Computer Programming* 79 (2014), pp. 70–85.

[40] Anjali Sree-Kumar, Elena Planas, and Robert Clarisó. "Extracting software product line feature models from natural language specifications". In: *ACM International Conference Proceeding Series*. Vol. 1. 2018, pp. 43–53. ISBN: 9781450363716. DOI: 10.1145/3233027.3233029.

[41] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. "Automated analysis of feature models 20 years later: A literature review". In: *Information Systems* 35.6 (2010), pp. 615–636. ISSN: 03064379. DOI: 10.1016/j.is.2010.01.001.

[42] Pierre Yves Schobbens et al. "Feature Diagrams: A survey and a formal semantics". In: *Proceedings of the IEEE International Conference on Requirements Engineering*. 2006, pp. 139–148. ISBN: 9780769525556. DOI: 10.1109/RE.2006.23.

[43] Amador Durán et al. *FLAME: a formal framework for the automated analysis of software product lines validated by automated specification testing*. 2017. DOI: 10.1007/s10270-015-0503-z.

[44] V. A. Prasad and R. Ranjith. "Intelligent Chatbot for Lab Security and Automation". In: *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. 2020, pp. 1–4. ISBN: 9781728168517. DOI: 10.1109/ICCCNT49239.2020.9225641.

[45] Ebtesam Hussain Almansor, Farookh Khadeer Hussain, and Omar Khadeer Hussain. "Supervised ensemble sentiment-based framework to measure chatbot quality of services". In: *Computing* (2020). ISSN: 1436-5057. DOI: 10.1007/s00607-020-00863-0. URL: https://doi.org/10.1007/s00607-020-00863-0.

[46] Boris Galitsky. *Developing enterprise chatbots: Learning linguistic structures*. 2019, pp. 1–559. ISBN: 9783030042998. DOI: 10.1007/978-3-030-04299-8.

[47] Sara Pérez-Soler et al. "Towards conversational syntax for domain-specific languages using chatbots". In: *Journal of Object Technology* 18.2 (2019), 5:1. ISSN: 16601769. DOI: 10.5381/JOT.2019.18.2.A5.

[48] Sara Pérez-Soler, Esther Guerra, and Juan de Lara. "Model-Driven Chatbot Development". In: *Conceptual Modeling*. Ed. by Gillian Dobbie et al. Cham: Springer International Publishing, 2020, pp. 207–222. ISBN: 978-3-030-62522-1.

[49] Sara Pérez-Soler, Esther Guerra, and Juan de Lara. "Model-Driven Chatbot Development". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Ed. by Gillian Dobbie et al. Vol. 12400 LNCS. Lecture Notes in Computer Science. Springer, 2020, pp. 207–222. ISBN: 9783030625214. DOI: 10.1007/978-3-030-62522-1_15. URL: https://doi.org/10.1007/978-3-030-62522-1{_}15.

[50] Gobinda G Chowdhury. "Natural language processing". In: *Annual review of information science and technology* 37.1 (2003), pp. 51–89.

[51] Nitin Indurkhya and Fred J Damerau. *Handbook of natural language processing*. Vol. 2. CRC Press, 2010.

[52] Alpa Reshamwala, Dhirendra Mishra, and Prajakta Pawar. "Review on natural language processing". In: *IRACST Engineering Science and Technology: An International Journal (ESTIJ)* 3.1 (2013), pp. 113–116.

[53] Abhimanyu Chopra, Abhinav Prashar, and Chandresh Sain. "Natural language processing". In: *International journal of technology enhancements and emerging engineering research* 1.4 (2013), pp. 131–134.

[54] Prakash M Nadkarni, Lucila Ohno-Machado, and Wendy W Chapman. "Natural language processing: an introduction". In: *Journal of the American Medical Informatics Association* 18.5 (2011), pp. 544–551.

[55] Sara Pérez-Soler, Esther Guerra, and Juan de Lara. "Creating and Migrating Chatbots with CONGA". In: *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE. 2021, pp. 37–40.

[56] Juliana Alves Pereira et al. "Visual guidance for product line configuration using recommendations and non-functional properties". In: *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*. 2018, pp. 2058–2065.

[57] Andreas Pleuss, Rick Rabiser, and Goetz Botterweck. "Visualization techniques for application in interactive product configuration". In: *Proceedings of the 15th International Software Product Line Conference, Volume 2*. 2011, pp. 1–8.

[58] Andreas Pleuss and Goetz Botterweck. "Visualization of variability and configuration options". In: *International Journal on Software Tools for Technology Transfer* 14.5 (2012), pp. 497–510.

[59] Klaus Schmid, Rick Rabiser, and Paul Grünbacher. "A comparison of decision modeling approaches in product lines". In: *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems*. 2011, pp. 119–126.

[60] Goetz Botterweck et al. "Visual tool support for configuring and understanding software product lines". In: *Proceedings - 12th International Software Product Line Conference, SPLC 2008*. 2008, pp. 77–86. ISBN: 9780769533032. DOI: 10.1109/SPLC.2008.32.

[61] Ji Soo Yi et al. "Toward a deeper understanding of the role of interaction in information visualization". In: *IEEE transactions on visualization and computer graphics* 13.6 (2007), pp. 1224–1231.

[62] Robert Spence. *Information visualization*. Vol. 1. Springer, 2001.

[63] Raúl Mazo et al. "Recommendation heuristics for improving product line configuration processes". In: *Recommendation Systems in Software Engineering*. Springer, 2014, pp. 511–537.

[64] Marcello La Rosa et al. "Questionnaire-based variability modeling for system configuration". In: *Software & Systems Modeling* 8.2 (2009), pp. 251–274.

[65] David Benavides, Pablo Trinidad, and Antonio Ruiz-Cortés. "Automated reasoning on feature models". In: *Lecture Notes in Computer Science*. Vol. 3520. 2005, pp. 491–503. DOI: 10.1007/11431855_34.

[66] Don Batory, David Benavides, and Antonio Ruiz-Cortes. "Automated analysis of feature models: challenges ahead". In: *Communications of the ACM* 49.12 (2006), pp. 45–47.

[67] Sascha El-Sharkawy and Klaus Schmid. "Supporting the effective configuration of software product lines". In: *Proceedings of the 16th International Software Product Line Conference-Volume 2*. 2012, pp. 119–126.

[68] Goetz Botterweck, Mikolas Janota, and Denny Schneeweiss. "A design of a configurable feature model configurator". In: (2009).

[69] Songlin Chen, Liwei Liu, and Mitchell M Tseng. "Product configuration via negotiation for mass customization: An interactive goal programming approach". In: *2009 16th International Conference on Industrial Engineering and Engineering Management*. IEEE. 2009, pp. 999–1003.

[70] Juliana Alves Pereira. "Runtime collaborative-based configuration of software product lines". In: *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. IEEE. 2017, pp. 94–96.

[71] Noor Hasrina Bakar, Zarinah M Kasirun, and Norsaremah Salleh. "Feature extraction approaches from natural language requirements for reuse in software product lines: A systematic literature review". In: *Journal of Systems and Software* 106 (2015), pp. 132–149.

[72] Chetan Arora et al. "Automated extraction and clustering of requirements glossary terms". In: *IEEE Transactions on Software Engineering* 43.10 (2016), pp. 918–945.

[73] Li Yi et al. "Mining binary constraints in the construction of feature models". In: *2012 20th IEEE International Requirements Engineering Conference (RE)*. IEEE. 2012, pp. 141–150.

[74] Kun Chen et al. "An approach to constructing feature models based on requirements clustering". In: *13th IEEE International Conference on Requirements Engineering (RE'05)*. IEEE. 2005, pp. 31–40.

[75] Mostafa Hamza and Robert J Walker. "Recommending features and feature relationships from requirements documents for software product lines". In: *2015 IEEE/ACM 4th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*. IEEE. 2015, pp. 25–31.

[76] George A Miller. "WordNet: a lexical database for English". In: *Communications of the ACM* 38.11 (1995), pp. 39–41.

[77] Sebastian Krieter et al. "FeatureIDE: Empowering Third-Party Developers". In: *ACM International Conference Proceeding Series*. Vol. 2017-September. 2017, pp. 42–45. ISBN: 9781450352963. DOI: 10.1145/3109729.3109751.

Appendices

A

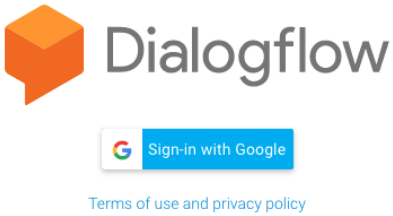
Create and import an agent in Dialogflow console

This appendix provides a guide to importing the generated chatbot into Dialgflow and integrating it with Telegram.

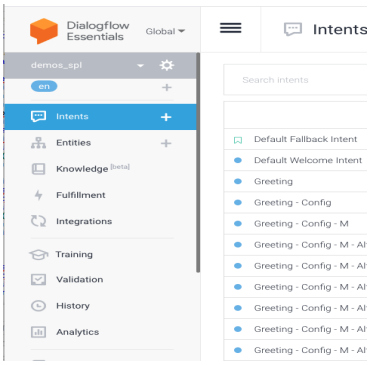
A. Create a Dialogflow Agent.

(i). Open any browser and navigate to https://dialogflow.cloud.google.com

(ii). Sign in with a Google account. If none exists, there will be a need to create one in order to use the Dialogflow console.

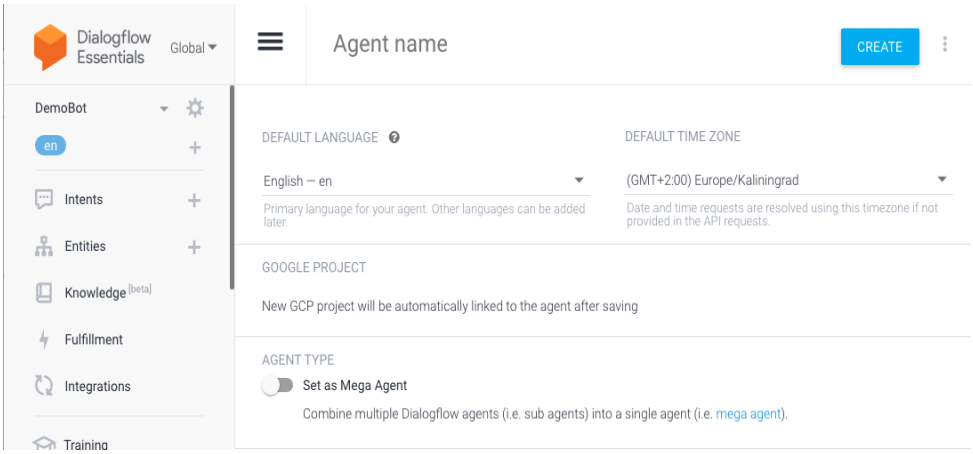


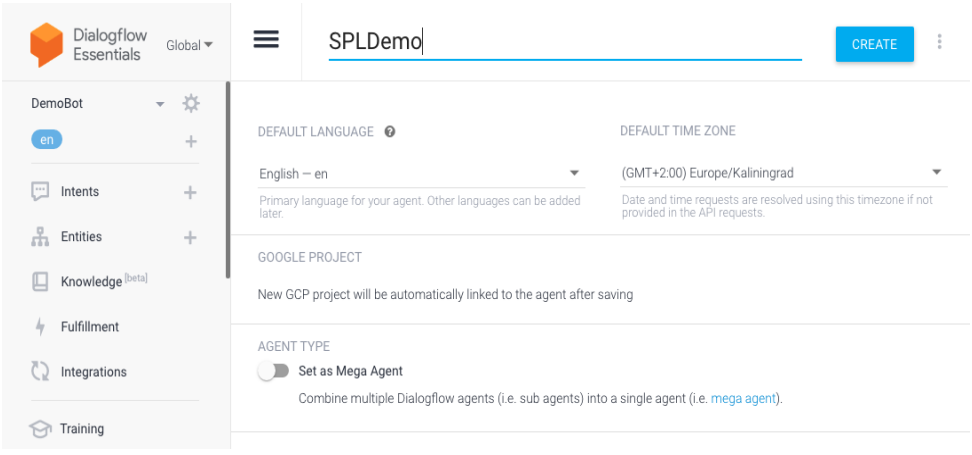
(iii). Access is granted to the Dialogflow console once logged in.



(iv). Create a Dialogflow Agent and Import agent from zip file.

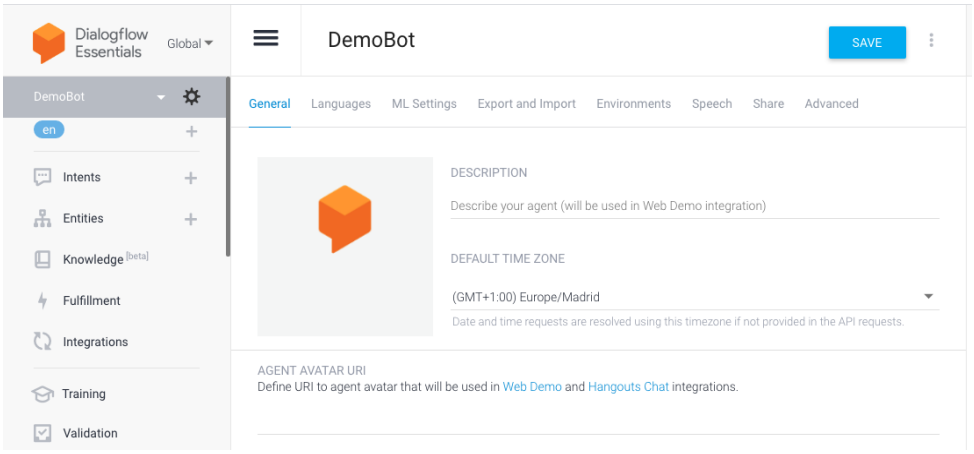
(a) Enter a name for the agent, select the language (English), and then click "Create."



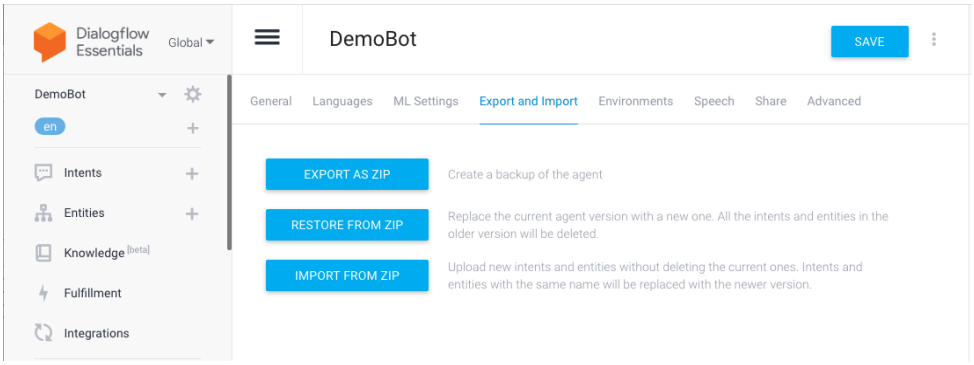


(b) Import the Dialogflow generated chatbot.

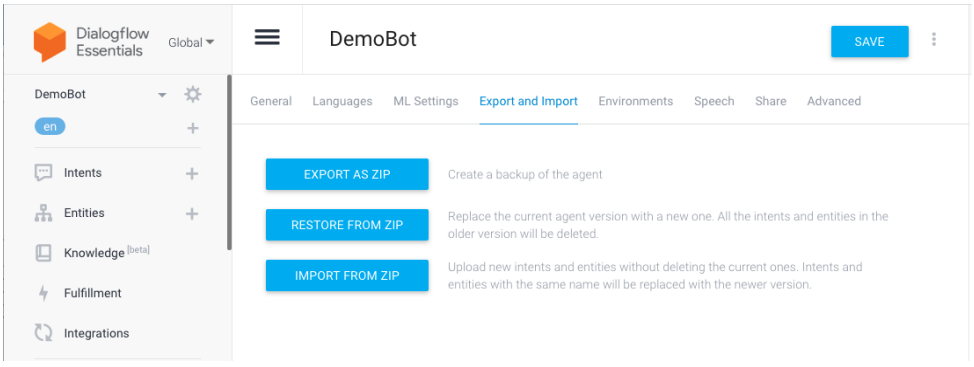
(c) To access the agent's configuration, navigate to "Settings" on the left hand side panel of the Dialogflow console.



(d) Select “Export and Import” from the list of menus at the top.

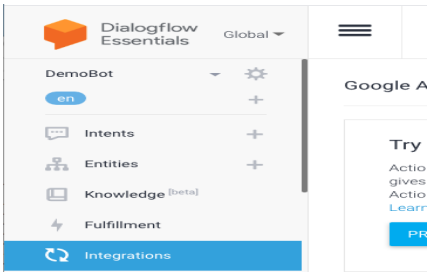


(e) Click "Import from zip" in the "Export and Import" section. Choose the chatbot-generated zip file, type “IMPORT,” and click the import button. Then wait for the training of the agent to be completed.

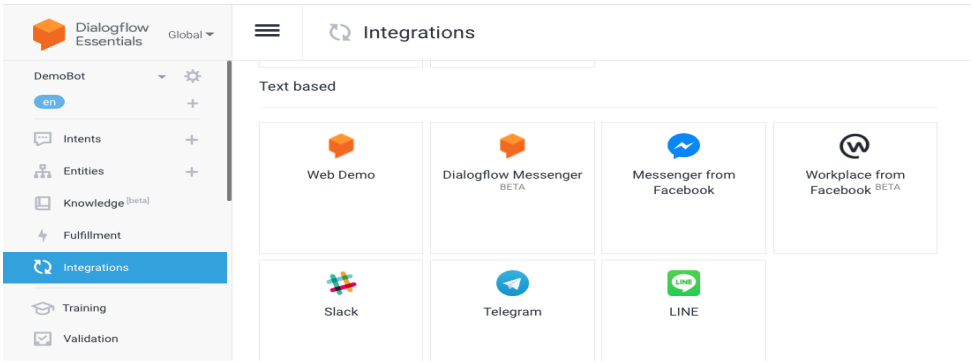


B. Integrate the agent with the Telegram bot.

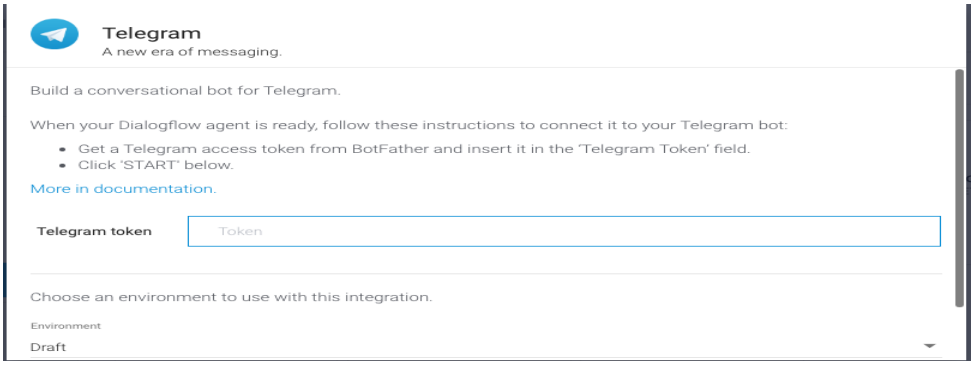
(i). From the left hand side panel of the console, click on "integrations"



(ii). Under "integrations," look for "Text-based" integration and select "Telegram."



(iii). Enter the token of the Telegram bot (obtained from the last step as shown in Appendix B) to which we want to connect the Dialogflow agent, and then click "START" to connect it to the Telegram bot.



The agent is now ready for use via the Telegram bot to which it is linked.

B

Create a bot for Telegram

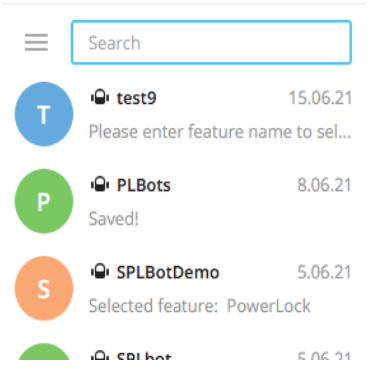
This appendix demonstrates how to build a Telegram bot.

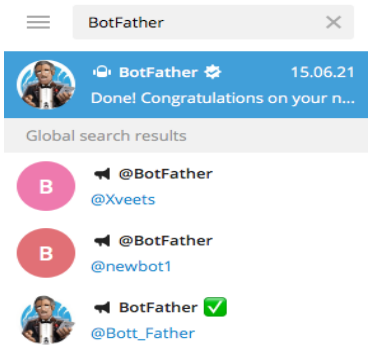
(i). Log into Dialogflow account.

(ii). Launch Telegram (from any device with Telegram installed).

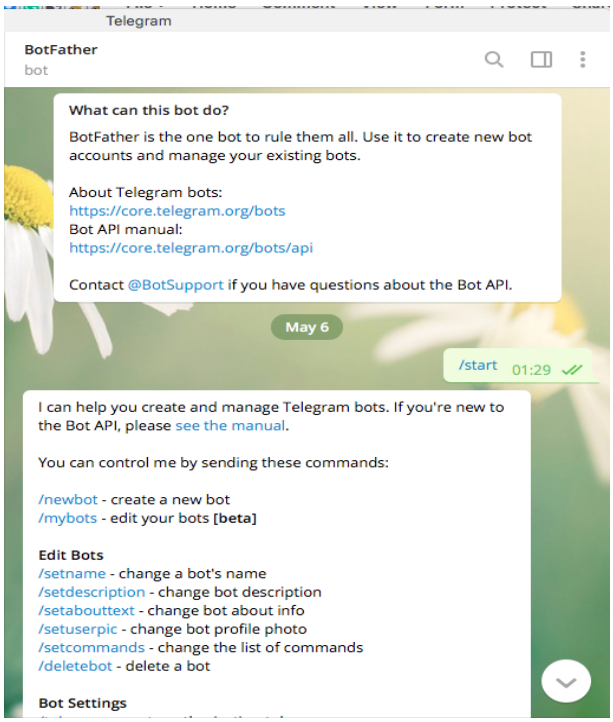
(iii). Start a dialogue with BotFather.

(iv). Enter "BotFather" in the "Search box" at the top left of the page and select BotFather (the first option) to begin a conversation.



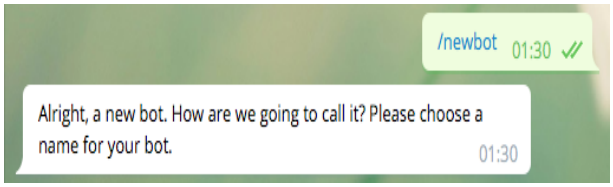


(v). Bot Father initiates a conversation and provides instructions on how to proceed.

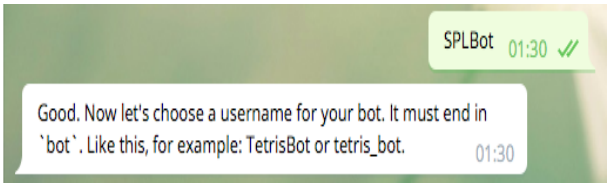


(vi). Converse with BotFather and provide all of the information required to build the new bot.

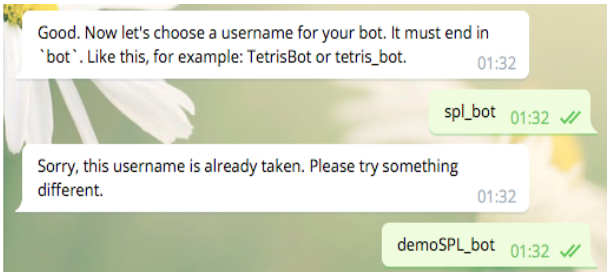
(vii). Enter "newbot" or display the command list and select "newbot." This command initiates a conversation in order to create a new bot.



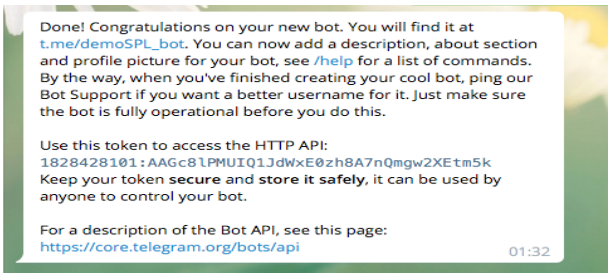
(viii). BotFather requests that you provide a name for the bot that is being created.



(ix). You must now select a nickname for the bot and send it. The nickname must end in "bot" and be unique. If another bot with the same nickname already exists, BotFather will notify you so that you can choose another one.



(x). The bot responds with a message that says, "Done! Congratulations on your new bot", provides a direct link to the bot, instructions on how to find the bot, and other information on bot customization.



(xi). The token required to access the new bot created via the Telegram HTTP API is also provided as shown above. It is a private code that will grant access to the bot in order to communicate with it, modify it, read messages, or connect it to other services.

(xii). Enter the Telegram BotFather token into the Dialogflow console integration section for telegram, as shown in Appendix A, and then click "START" to connect it to your Telegram bot.