Chatbots as Assistants: An Architectural Framework

Adam Di Prospero, Nojan Norouzi, Marios Fokaefs and Marin Litoiu
Dept. of El. Eng. and Comp. Sci., York University
Toronto, Ontario, Canada
dipro@yorku.ca,nojanno@my.yorku.ca,fokaefs@yorku.ca,mlitoiu@yorku.ca

ABSTRACT

Automated text-based or speech-based personal assistants, also known as chatbots, have been prevalent in several domains including marketing and technical support. Through mainstream applications, such as Siri or Alexa, their popularity has increased and we now see them being used in even more domains. Although the purpose of chatbots varies among domains, there are common elements that all chatbots share. By identifying these elements, it is possible to streamline the development of chatbots en masse and in a structured manner. Additionally, there can be common challenges in the development of such applications, for example, how to treat novice versus expert users or how to establish memory of the conversation. In this work, we propose a reference architecture for chatbots using concepts from Software Product Lines and Feature Models, where we outline the common elements as well as the common challenges. Using Watson and Bluemix as the basic platforms, we also present the creation of two chatbots, for different purposes, based on this reference architecture to highlight these commonalities.

CCS CONCEPTS

•Human-centered computing → Human computer interaction (HCI); •Computing methodologies → Cognitive science; •Computer systems organization → Distributed architectures; •Software and its engineering → *n-tier architectures;*

KEYWORDS

chatbots, cognitive assistants, software component architectures

1 INTRODUCTION

In recent years, chatbots have been most prominent at the intersection of artificial intelligence and social media. A large number of bots are active on Twitter [34] and Facebook [37]. There is also a growing number of chatbots active on platforms such as Amazon Echo with Alexa [4], Google Assistant on Google's api.ai platform [1]. To support their development, a number of tools are already available to enable advanced actions that require an understanding of context, flexibility in interaction, and a bot memory structure. This leads to a new breed of bots, "Cognitive Agents", that augment user capabilities, rather than replace them. In the context of this paper, the terms "bot" and "chatbot" refer specifically to the concept of a "Cognitive Agent"

Despite significant advances in chatbot technology, it is still challenging to implement Cognitive Agents in new domains. Challenges range from *cognition*, for example how to design and implement different chatbot personalities to address different types of users, all the way to *development and architectural* challenges, for instance, the need to orchestrate and integrate different types of web services, user engagement monitoring and improvement, bot memory management among others.

The goal of this paper is to outline our experiences, describe some of the challenges and to propose and prototype a framework that will encompass the tools, components and considerations necessary to facilitate and enhance the development of bots that are intended to operate in distinct domains. More specifically, the contributions of the paper are (a) requirements specification for two bots on specific domains and for a personality service; (b) a product line architecture for streamlining the development of bots for several domains; (c) prototype implementation of the architecture and a discussion of implementation and research challenges.

The remainder of the paper is as follows. In Section 2 we describe the requirements for two sample bots and their particular abilities, which will help us motivate the abstraction of the common development framework. Section 2.2 illustrates the importance of identifying user personalities and the value of formulating the bot's own personality and adapting its output to appeal directly to the current user. Section 3 explores the architectural elements, challenges and tools common across the different cognitive agents. While the bots share a common platform, each instance incorporates components only necessary for a particular bot. Section 4 describes the commonalities and variations of the bot feature model, and provides details on how to capture the various features of each bot. Section 5 focuses on particular technical choices and design challenges for the two bots. Section 6 discusses concepts and ideas that we are looking to implement, moving forward. Section 7 outlines the related work that motivated and guided the development and investigation of the bots.

2 REQUIREMENTS FOR A COGNITIVE AGENT PLATFORM

To motivate our cognitive and development framework, we begin with two case studies and highlight their requirements. The following sections present an overview of the requirements for two types of chatbots and the requirements and characteristics of a "Personality" service needed for both case studies.

2.1 Case Studies

2.1.1 YU Student Services Bot. The YU Services bot is intended for York University students with disabilities. The goal of the bot

CASCON'17, Toronto, Canada

DOI: 10.1145/nnnnnnn.nnnnnnn

is to inform students of available services, resources, and tools they might need. Since each student has an individualized learning approach and method of communication, a requirement is to implement a flexible information delivery style. For example, in the case of a student who is blind, the bot can interact with speech instead of text and visual cues. Another important consideration is student data confidentiality. If a user were to provide YU Services bot with confidential information, it must be stored securely, and not accessible by other users. As a chatbot, this application complements in-person assistance by providing a service that is convenient and aimed at smaller tasks that do not require much expertise.

2.1.2 System Administrator Minion. The System Administrator Minion (SAM) acts as an interactive alternative for cloud management services. Traditionally, administrators would use multiple tools to monitor their systems, retrieve data required to make informed decisions, and subsequently apply corrective actions to those systems. SAM can be configured to handle the busywork involved in getting the necessary data. Users can ask SAM to provide precise information that is filtered and prepared in the format they desire, including graphs and charts. Users can make adjustments to their systems through SAM, such as increase or decrease resources to keep systems running smoothly. The goal of the bot is to integrate management service and act as a proxy for the actual IT staff.

2.2 The Personality of Cognitive Agents

The personality of each cognitive agent plays an important role in engaging the users. To successfully interact with the users, each cognitive agent not only requires to produce logically sound responses, but those responses also need to emphatically match with each of the user's sentiments. In other words, the cognitive agent should be able to "feel" the user's input and adjust its subsequent response to the user's emotional tone. In order to endow our cognitive agent with a "faculty of feeling", the agent requires to possess a personality [28]. This approach to the personality of our bot does not entail it to constitute a singular core, or a singular "self". Rather, in order to bridge the possible gaps between the faculty of feeling and the other functions, the personality of our cognitive agent resides in a set of procedures and functions that are implemented along the other mechanisms that handle the responses. In our model, the personality of a bot has a static component, which we call *the principal self*, and a dynamic component, which we call *the adaptive self*. The latter is further divided into three subcomponents that fine-tune each response according to (1) the user's personality type, (2) the user's individual needs, and (3) the user's real-time feedback.

2.2.1 The Principal Self. Let us begin with the static aspect of the cognitive agent's personality. The main purpose of this component is to create a sense of coherence in the responses that the cognitive agent creates, and to make sure that each response adheres to a set of standards based on the agent's role and responsibilities. Thus, the principal self is determined at design time based on the purpose of the bot and the requirements around its function. In the case of YU Services bot, for instance, the principal self of the agent should take into account the traits that are generally expected of a counsellor or a helpdesk agent. For example, the cognitive agent should be empathetic -but not emphasize the disability of the students- as well as informative and solution-oriented. The principal self creates a sense of stability and acts as a core around which the dynamic aspects of the agent revolve, so the agent's personality would not be completely relative to the personality of the users. This way the users know what to expect from the bot, they become more engaged and eventually feel comfortable with using an automated service. The principal self should raise user satisfaction because most users have expectations from an agent, according to the agent's role, that should be addressed by the agent regardless of the user's specific personality. The way to apply the principal self is to create a list of standard personality traits, and have them in mind when generating the generic and neutral responses that are implemented in the cognitive agent's dialog flow. It is important to include key stakeholders in the requirements gathering phase. For example, counselling staff for the YU Services bot, and System Administrators for SAM can provide design guidance concerning common statements and a response tone that is acceptable based on the domain context.

2.2.2 The Adaptive Self. Similar to a person who adjusts her tone according to her targeted audience, each cognitive agent would also require to adjust its responses according to the personality of the users. Our goal is to maximize the ability of our bot to communicate and be engageable. Therefore, our bot needs to be adept at recognizing the personality of the end-users and adapting to it. Humans perform this mechanism intuitively, when their tone and language differs from one person to another according to their discursive situation and their relation to each other. Each cognitive agent, similarly, will categorize the personality, needs, and the tone of each user from the conversation and adjust to them according to the procedures explained below.

2.2.3 Adapting to User's Personality Types. Our model here follows the approach of Horzyk *et al.* [23] on man-machine interactions. In that work, the authors follow a psycholinguistic approach to recognizing human personalities, which aims to categorize people's personalities based on their choice of words as well as other textual patterns in their sentences. The authors present a list of eleven main personality types and the metrics for recognizing each one. For each human personality type (HPT), there are specific human personality needs (HPN) that the cognitive agent needs to take into account while producing the responses. In order implement that model, we utilize Watson's Natural Language Classifier [5], which uses machine learning algorithms on short text inputs in order to classify each input based on the metadata. Once we input the textual patterns and word choices for each personality type, the service will go into "training mode" to learn those categories and subsequently, recognize them while interacting with the users. Vertical operators are used to assess the possibility of each personality category, and during the interactions, neutral responses adjust to the most plausible personality type to include the specific psychological needs for each type.

2.2.4 Adapting to User's Needs. This component is specifically useful in YU Services bot, which will interact with students with

disabilities. It is crucial to create responses that take the user's disability into account. Since the cognitive agent should not emphasize the user's disability or mention it directly without the user's reference, it should hypothesize the disability through: (1) the user's inquiry about a particular technology that is associated with a specific disability, (2) the user's explicit reference to a disability, or (3) hypothesizing the type of disability through the user's interaction with the bot. Responses vary according to the type of disability. For instance, in the case of a user with ADHD (Attention Deficit Hyperactivity Disorder), responses should be short and concise.

2.2.5 Adapting to User's Tone. During their interaction with the chatbot, users may express certain feelings towards the bot or the conversation. Regardless of each user's personality type and disability, the bot should reply to those feelings in an appropriate manner. In order to do so, we use Watson's Tone Analyzer [7], which detects each response's sentiment and its intensity. If the intensity of an expressed feeling reaches above 50 percent, the chatbot will affix a standard sentence to the beginning of its response to address that feeling. If for instance, the user feels frustrated, the bot may add the following sentence: "I understand that you are frustrated, [user's name]."

2.3 Eliciting and Validating Content with Stakeholders

When initially planning the design of a chatbot, it is essential to include key stakeholders in the process. One approach is to hold meetings with potential users, along with the group or organization that is most familiar with the user base. These sessions can be organized and delivered by the development team themselves, or by an external group who specializes in holding such sessions [20]. The sessions could be used to determine the types of content the users would inquire about, what the users overall goals would be for using the bot, and how the users would most likely interact with the bot, such as text-based methods, or by using speech-to-text technologies. Once the exploratory phase is complete, and the bot is able to communicate with the user, the next step is to validate the statements that the cognitive agent is making. For example, in the case of the YU Services bot, it is important to include the staff from the Disability Services Office in this process. The goal is to confirm that the bot's statements are in line with the tone and content that staff would typically use. It is also important to validate with students who would be using the bot. The answers must meet the expectations of students before continuing with the development process.

An important takeaway from this step is to ensure that the objectives of the chatbot, and the objectives of the users do not conflict, and how to elegantly handle conflicts when they arise. For example, during the early steps of the exploratory phase, it was mentioned that there may be "high risk" students who are seeking help that the bot may not be able to provide. In cases like these, it would be important to offer the student a direct line to a 24-hour support line [1]. In this case, the objective of the bot - to assist students autonomously - comes in conflict with the student-based objective - their safety. The needs of the student should override the objective of the bot. For each bot, identifying the possible users and including them in the process early on is beneficial, but the process is iterative and should happen throughout the development cycle.

[1] https://good2talk.ca/

3 REFERENCE ARCHITECTURE FOR CHATBOT APPLICATIONS

Having covered the cognitive aspects of chatbots, we move on to discuss the development and architectural considerations. The proposed architecture, as seen in Figure 1, is a three-tier architecture with a user interface (UI) component, the bot application core, a personality processing module, and external data sources and services. Practically speaking, the core consists of the services relevant to chatbot applications in general and which are contained in the development platform, while external sources and services are relevant to the domain of the specific application. Within the same tier, multiple modules or services can be used depending on the specific domain of an application. For example, the Text-to-Speech module is implemented for the YU Services bot but not for SAM. This is according to a *software product line* approach [35], where the available modules are considered *assets* and the architectural tiers consist of *variability* points. Such an approach can easily guide the development of various bots by identifying viable products from a feature model [15]. Software product lines provide the structure to develop multiple products using a core asset base. The following sections describe individual features. Section 4 provides more detail on how these features are either common across both bots, or are variable and interchangeable.

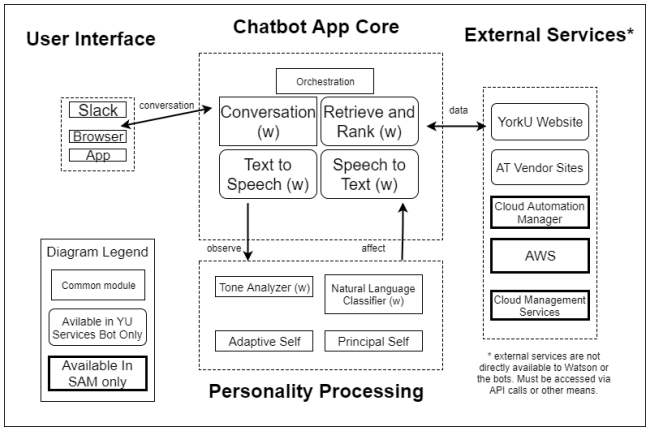

Figure 1: Reference Architecture for chatbot applications.

3.1 User Interfaces

Naturally, every bot may use a different UI. It is important for the UI to suit the users in the target domain. Development platforms, like IBM Bluemix, offer a variety of UI modules and certain orchestration tools, like Node-RED [19], allow the capability to switch between UIs seamlessly with little or no impact to the rest of the application. This makes it easy to suit the needs of each application, even if these change after deployment. For example, SAM initially used a

browser UI, but having users interact with SAM through Slack [39] was ideal for team management.

3.2 External Services and Sources

Depending on the purpose of the chatbot, every application may have a need to access domain specific data sources or services. These sources can either provide information that cannot be included in the bot at design time, or they can be used to provide dynamic content for the bot and improve its human-like behavior and subsequently increase user engagement. For example, the YU Services bot can access content about university services, even if it is not within its capabilities or when the response confidence is low. Similarly, SAM can access the cloud monitoring services to provide runtime information about the managed system on demand. In most cases, the application core can connect to external sources via REST web service calls.

3.3 Chatbot Application Core

The core contains the logic of the bot and is implemented with a combination of available platform services, like conversation services. These services complement the basic dialog, improve user experience and provide alternative means to interact with the bots. For example, Text-to-Speech service [6] transfers written text into spoken speech. Conversely, Speech-to-Text service [11] takes a user's spoken statement and converts it to text. The text is then passed into the application core and sent to the other services. These services are particularly useful when the data is in the wrong format. For example, a service requires the data to be in a text format, but it is currently in an audio/speech format.

The core also contains the dialog logic itself, which handles the progress of the exchange between the bot and the user, as well as the content of the bot's statements based on external stimuli. Processing steps handle bot memory (i.e. keeping track of the user-provided context).

When the bot is not sufficiently equipped with the content to answer a particular question, it will not provide an incomplete response. Instead, the core can call the Retrieve and Rank service [10]. This service connects with external data sources and finds relevant information. The service returns a ranked list of documents or data sources back to the core. The application, then, can present the information for which it is most confident, and leave the final choice to the user.

3.4 Text Analysis Services - Personality Processing

The chatbot core relies on services provided by the personality processing tier. The personality processing uses analysis services, combined with custom functionality, in order to provide the user with an experience tailored to their needs. Custom components are responsible to preprocess user statements as they go through the dialog and before being sent to other services. Once results are returned from a service, the responses pass through custom post-processing steps and are sent to the user. Processing steps modify statements between the principal and adaptive self. For example, the Tone Analyzer can detect the user's tone from written text and then output the analysis to the custom processing steps. This tone information is used to modify the bot's response accordingly. The Natural Language Classifier processes user input and determines the intent behind statements so as to bolster the data, which is then used when determining a suitable response.

In our implementation, all of the above mentioned services are from the Watson suite, which is indicated with a "(w)" in Figure 1. The bots, however, can utilize non-Watson services, such as the Cloud Automation Manager [26] service. The Cloud Automation Manager is used by SAM to connect to, monitor, and scale cloud systems. As the bots mature in their development, additional services can be added via Node-Red.

3.5 Dialog Service

Our implementation of chatbot applications rely on the Watson Conversation [27] service. Here is where the developer can define the flow of the dialog between user and bot and the content of the bot's response. In this section, we outline the basic elements of the service as well as some processes we define to build the dialog.

3.5.1 Establishing Entities and Intent. Consider typical conversation flow with a human assistant, and how that interaction maps to components in the Conversation service. In general, an assistant is able to identify a purpose of the conversation and continue the interaction. If someone were to ask, "Where is the meeting", the assistant implicitly understands the purpose is to find a location. Conversation employs the concept of "intents" [25] to identify the purpose of a discussion. In the Conversation service, an intent is the purpose that can be inferred from the user statement. With SAM, an intent might be "whatTime", which indicates the user wants to know the time of an event. A sample user statement could be, "When did the load balancer crash?". Even though the user does not explicitly state "what time", the system still recognizes the statement as having the "whatTime" intent.

Another aspect of conversation with a human assistant is the topics being discussed. With the statement "Where is the meeting", the assistant infers that the current topic is, "meeting". Watson identifies topics as "entities" [24]. With SAM, example entities would be component names, for example CPU, virtual machines and so on. Much like intents, entities are defined within the Conversation service. All entities reside in an entity group, which is made up of entity values and any possible synonyms for each entity value.

3.5.2 Conversation Flow. Conversations consist of a back-and-forth flow between the people involved. Since the scope of these bots is limited to conversations between a person and an assistant, they generally follow a structured flow, but unexpected cases must be handled. With Watson Conversation, it is possible to guide the user along the conversation flow and direct the user into a structured path, while still handling any tangent ideas. Conversation flow in Watson Conversation is handled by creating a suitable *dialog structure* [8].

There are two main considerations when implementing a dialog structure. The dialog should attempt to get the user to explicitly state their intent. The dialog should also elicit the precise aspect that goes along with the declared intent. For example, if a user asks, "Where is the location?", with no prior statements, the bot should follow up with "Which location are you referring to?". The

dialog structure must handle unstructured conversation flow, such as when the user jumps between unrelated topics. This is done using the *Jump To* [12] function in Conversation, which skips complete branches, jumping to a specific dialog node. While the flow itself is highly structured at design time, the reality is that users may be unpredictable. The dialog structure can take the form of a tree, but in the case of the bots outlined in Section 2, it resembles a network. A network structure is capable of handling the unpredictable nature that will arise during actual use.

3.5.3 Establishing Memory. Entities, intents, and dialog structure only work if there is a system to keep track of the values at any point in the conversation. In a conversation between two people, the question "Where is it?" has no intrinsic meaning, the person must have some prior context to understand the question. In an actual conversation, this is handled by the memory of each person. Since the Conversation service is stateless, memory is handled by the application itself, which must depend on *context variables* [9]. Context is any value that the developer deems necessary to store and track between calls to the Conversation service. Consider the question, "Where is the meeting". For this first turn of the conversation, the application might capture the value of "`topic = meeting`". On the next turn, when the person asks "When is it?", the system can see that in this instance, "it" refers to the value currently stored in the topic context. The user can continue to ask questions without explicitly referring to "meeting" and the system can recognize the context. Context values - stored in JSON files - can be added, removed and modified via the application, or Watson itself.

At times, the system needs to collect multiple context values, and if the system is missing context, it should redirect to the dialog node that is capable of collecting that context. In the YU Services bot, if a user asks to make an appointment to see a product, the system must know the name of the product, as well as the desired date. If the system is missing any of these values, the bot should be able to collect missing values by soliciting the values from the user.

3.5.4 Suitable Data Structures for Context. Consider cases where the user asks questions about multiple topics. It is important for the bot to keep a history of the topics, and for the bot to have a sense of which topic the user is currently referring to. For an illustration of the problem, see Figure 2

Student: tell me about Kurzweil
Bot: Kurzweil is a text-to-speech application
Student: is that like Text Help?
Bot: Yes, Text Help is a text-to-speech application too
Student: can I see a demo?
Bot: You want to see a demo of Kurzweil?

Figure 2: Example of a typical dialog

In Figure 2, the user is asking about an assistive technology application (AT app) called "Kurzweil"[2] and also mentions another AT app called "Text Help"[3]. Even though the last app mentioned was Text Help, The bot should still provide info on a demo of Kurzweil, not Text Help. The bot should recognize that the main focus is on Kurzweil, and that the student is asking about Text Help only to better understand what Kurzweil is. Since there is room for confusion, the bot also asks for confirmation before actually providing the content. This may also help create a natural flow which keeps the user engaged and active in the chat.

In order to store and track a history of contexts, a stack data structure is used. As new topics and context are requested by the user, they will be added to the stack. The items in the stack can be removed when all possible branches for a given topic or context are exhausted. For example, the YU Services bot has a branch for student technology. Within this branch, there are four nodes that deal with different aspects of the technology available. Once each of these nodes have been traversed, the bot can infer that the student does not have any remaining questions, and the "technology" context can be popped off the stack.

3.5.5 System Prompts. The bots provide guidance by prompting users with statements they can enter. This gives first-time users an idea of what they can ask. Klopfenstein *et al.* [31] describe the importance of "onboarding", whereby the user is given direction on how they can interact with the bot, what statements are understood, and the range of topics that the bot can handle.

One approach to increase user satisfaction is to also accommodate advanced users, or users who do not necessarily need to rely on the prompts provided by the bot. These users can ignore the prompts and enter exactly what they are looking for, without having to go through every step of the dialog branch. For example, if a user provides all of the necessary context in a single statement, the bot does not need to traverse all of the intermediary dialog nodes that collect individual context values.

4 DEFINING VARIABILITY AND COMMONALITY

Along with the architecture, we also frame the design and development of chatbots using *Feature Models.* A feature diagram is a useful visual on many fronts. First introduced by Kang *et al.* [29], a feature diagram is a simple and clean way to represent variability and commonality of a software system. A feature diagram represents both the common and variable features, and their dependencies [15]. There are multiple notations and approaches for creating a feature diagram. The feature diagram in Figure 3 is based on the notation proposed by Czarnecki *et al.* [15], which was built on the approach of the Feature-Oriented Domain Analysis (FODA) method [29]. We also referred to a later work by Czarnecki [16], which modifies the feature diagram by integrating a number of additional notation extensions. We opted to go with Czarnecki's initial notation method as it was able to capture the variability of the chatbot product line without any ambiguity, though the feature diagram of our chatbot applications can be modified to include additional notation such as cardinality [14].

Another strength of a feature diagram is that it is useful for both the development team, and also as a tool for explaining the product

[2]https://www.kurzweiledu.com/default.html

[3]https://www.texthelp.com/en-us/

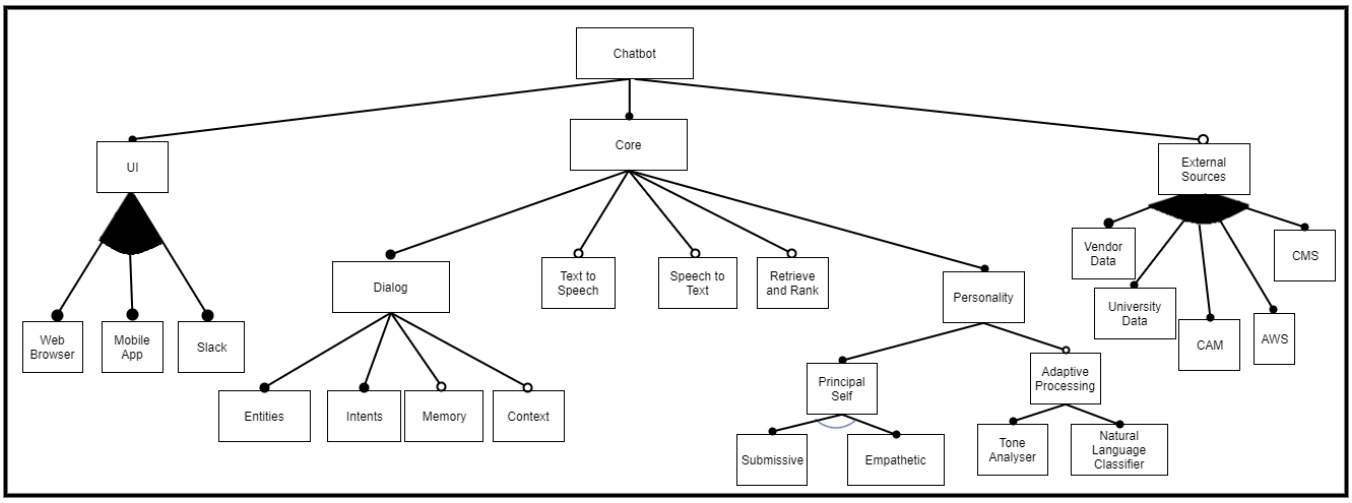

Figure 3: Feature diagram of the chatbot platform

to stakeholders. The diagram is straightforward and easy to understand after a short explanation of the notation. The root node at the top of the diagram is the *concept* node, which describes the general idea of the system. Each box below are *features* and *subfeatures*. The features and subfeatures are connected by lines with circles at their ends. The solid circles indicate that the feature is mandatory. The empty circles indicate that the feature is optional. The dark arc between multiple subfeatures indicate that the particular feature set contains *or-features*. With a set of or-features, any non-empty set is possible, that is, there must be at least one feature selected from the set.

Our feature diagram for chatbot applications has the following qualities. The concept node encompasses chatbots in general. In the case of the UI features, at minimum one UI must be chosen between a browser, mobile app, or Slack, but all three can be integrated as well. The same applies for the external sources. Since the external sources feature is an empty circle, the whole group is optional. If the group is chosen, at least one external source needs to be integrated. Any number greater than one can also be integrated, depending on the bot being developed. The core contains two mandatory features; Dialog and Personality. These must be implemented in each bot instance. The other nodes are not mandatory and might only be implemented in one of the two bots, though they are not restricted from being implemented in both. Within the Dialog feature set, Entities and Intents are mandatory, whereas Memory and Context are optional. While memory and context are important, it is possible to implement a basic bot without these features. The subfeatures of the Principal Self node are connected by an arc. The arc denotes that only one of the features can be chosen. In this case, only two primary selves are in development, a submissive self for the System Admin bot, and Empathetic self for the Student Services bot. Additional selves can be added to the feature diagram when other bots are incorporated into the product line. The Adaptive Processing node is optional and contains the Tone Analyzer and the Natural Language Classifier. After the Tone Analyzer and Natural Language Classifier analyze the user statement, the Adaptive Processing component functions in a similar way to the decorator design pattern in object oriented programming. It does this by *injecting* prefixes and suffixes to the bot's response, effectively customizing the reply back to the user. While both bots implement this Adaptive Processing feature, additional bots added to the product line may only make use of the principal self.

It is important to note that this is a feature diagram and not a full feature model. A feature model includes the diagram, along with additional information such as feature descriptions, specifications for binding sites, binding modes, details about stakeholders, users, customers, and other details [16]. The plan for the near future is to also complete the future model for the chatbot architecture.

5 IMPLEMENTATION OF THE FRAMEWORK

A closer look at both bots illustrates our experience implementing the proposed framework, and the particular challenges we faced. The following bot-specific components and development activities are built on the modules found in the reference architecture and feature diagram. Each bot was built using IBM Bluemix as the platform, along with the Watson Conversation service. They share the use of a common development tool, Node-RED. Beyond these technologies, each bot incorporates unique features via other services available in Bluemix.

5.1 YU Student Services Bot

The YU Services bot will be housed in the Assistive Technology Lab on the York campus as a physical kiosk. It will allow students to interact with the bot and have staff on site in case they run into any difficulties. The bot will also be available to students via any device with an Internet connection and browser. Our goal is for the kiosk to introduce the students to the idea of using a bot, and then have them transition into accessing it on their own devices. The purpose of the bot is to handle the tasks that would typically inundate staff. The goal is to provide students with instant access to information that might be time consuming to track down by other means, such

as making an appointment with a staff member, or searching the web.

The YU Services bot builds upon the reference architecture and is tailored to a range of student abilities. Figure 4 - a modification of Figure 1- depicts the specific features we implemented in the YU Services bot. In the first prototype, the UI is browser-based and uses HTML and Javascript. The UI must be simple and uncluttered, with a high contrast color pallet and large sans-serif fonts for users with low-vision and reading-based exceptionalities. We opted for a web UI as the stock interface, since students may prefer their own browsers with specialized accessibility features[4] or extensions[5] that they feel comfortable using. The bot incorporates Text-to-Speech service for students with dyslexia or other reading exceptionalities. Users who have challenges with typing can use the Speech-to-Text module. Both are available through Bluemix and are used to make the bot completely accessible.

The YU Services bot changes between an Empathetic personality at its core, to other personalities that meet student learning styles and personality types. This process starts with the Tone Analyzer, which infers information about the user questions. Bot response is augmented to match the user's detected tone. The Natural Language Classifier service parses out possible intentions of the user, along with the specific entities in their statement. Once the entities have been parsed, a suitable dialog path can be traversed in the Conversation service.

The bot adapts its output to meet the needs of the students. Consider students who prefer a high level of detail in their responses. This can be inferred by the number of follow-up questions a user asks. Once detected, the bot should adapt to provide more details for subsequent questions. Although the bot will not ask students to state their disability, students may disclose such information. The bot uses disclosed information to adapt its responses. While the bot's principal personality is suitable for a wide range of users, the adaptive personality modifies responses to increase user satisfaction. It will be important for the bot to use natural language. Murgia *et al.* [33], conducted an experiment where a bot answered questions on StackOverflow, first with a human name, and again where it identified itself as a bot. The experiment showed that users were more likely to accept responses from the bot when it identified as a person. Conversely, Edwards *et al.* [17], found that a Twitter feed managed by a bot was considered by users to be a reputable source of information. If the YU Services bot can convey its responses in a natural tone, it should increase user acceptance.

Within the Conversation service, the bot is structured around content areas. Each content area has dialog branches that further specializes into subgroups. One of the popular content areas is assistive technology (AT) available for students. This branch has four topics that are popular with students; availability of AT on campus, downloadable demos, appointment booking, and general information. For each area, the bot calls external data sources such as the York University website or AT vendors and returns the latest information. In cases where the external sources have publicly available APIs, the data can be queried and stored. In other cases, the information is scraped from the website. When a topic is not recognized, the bot calls the Retrieve and Rank service and returns a ranked search result to the user. The goal is to meet the user needs with an exact answer to their question. If this is not possible, a ranked result is a strong alternative to simply stating that the bot does not have any information.

While the bot is designed specifically for York University, the data can be modified to be used by other universities and colleges, as well as any institution that provides services to persons with disabilities. In practice, migration to another institution would involve changing the data sources and the entities, but little else.

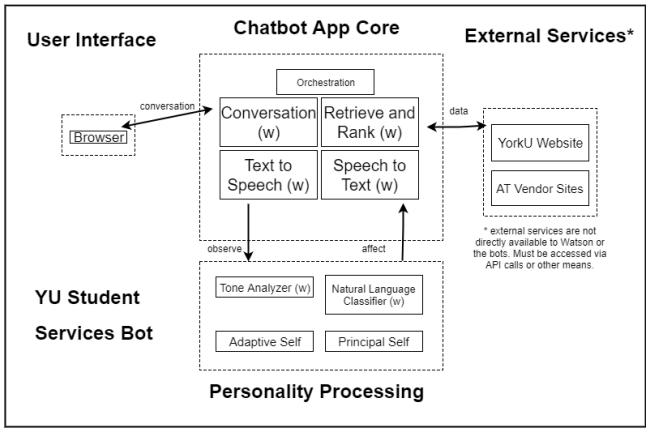

Figure 4: Architecture Diagram showing only the features implemented for the YU Student Services Bot.

5.2 SAM: A Systems Administrator Minion

SAM is a bot designed to act as an intermediary between a systems administration team and the systems they manage. SAM differs greatly from the YU Services bot in terms of its goals and users, but both share the common platform. It was important for us to design two bots that differ greatly on a superficial level, but share a number of commonalities in terms of their architecture. This allowed us to get a sense of how to efficiently develop the bots using a software product line approach. Figure 5 - a modification of Figure 1- shows the specific features we used to implement SAM. The user interface is Slack, as opposed to the browser interface of the YU Services bot, since it is a popular tool among system administrators and an acknowledged collaboration tool. SAM is integrated with IBM's Cloud Automation Manager (CAM) to scale the components of each system in its portfolio. CAM uses Terraform templates[6] that specify components needed for an application. A Terraform template lists out all components a software system requires to run. Templates include items like virtual machines, storage, virtual private networks, and other infrastructure components. Using CAM, it is possible to implement thresholds on performance indicators for an application, and deploy new templates to modify components and keep applications within acceptable performance standards. SAM can be configured to automatically act when a threshold is passed, or when an alert is triggered. SAM provides data to the

[4]https://support.mozilla.org/en-US/kb/accessibility-features-firefox-make-firefox-and-we

[5]https://addons.mozilla.org/en-US/firefox/tag/colorblind

[6]https://www.terraform.io/

systems administrator in the form of graphs, charts, and summary statements.

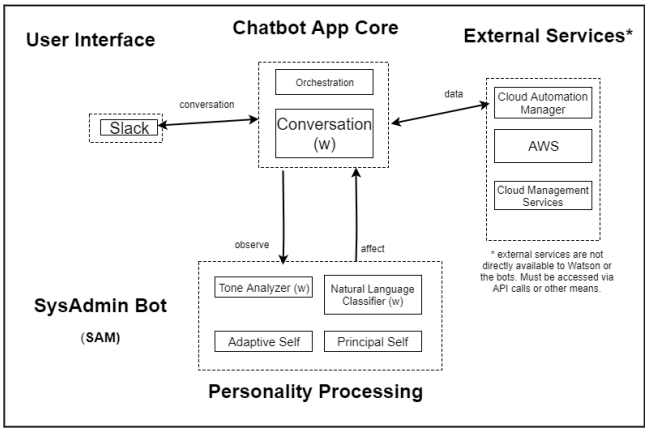

Figure 5: Architecture Diagram showing only the features implemented for the SysAdmin Bot - SAM.

SAM's conversational style is similar to a command-line interface as opposed to a human-to-human chat. Klopfenstein *et al.* [31] describe the benefits of keeping natural language to a minimum, such as reducing misunderstandings. Users type short commands that SAM understands. SAM accommodates both advanced and novice users. To help accommodate novice or first time users, SAM will include prompts in its statements, while advanced users can choose to ignore these prompts and go directly to the desired branch of the chat. When considering the cognitive aspect of SAM, outlined in Section 2.2, we took a different approach than what we initially did in the YU Services bot. With SAM, the goal is for personality to primarily manipulate the verbosity of responses, and the types of data output the users prefer. The principal self consists of concise statements that provide only the essential information. The principal self will be modeled around the typical statements you would find in the output of a system log, or command line interface. The adaptive self allows for detailed statements, or specific outputs that some system administrators may prefer. For example, if a system administrator constantly requests usage stats be formatted a certain way, the bot adapts to meet the user needs.

6 FUTURE IMPLEMENTATION AND NEW FEATURES

There are a number of features in the early phases of conceptualization. These ideas may hold value for future iterations of the bots. As described in Section 3.5.4, each of the bots will keep track of the context within a given session. The chat topics that one user experiences will be different from any other user. Each of the sessions will be treated independently, and the session memory will be cleared once the session is completed. But there is also another level of data storage that has value in its own right. The bots could potentially store persistent data that is kept across user sessions. Having a mechanism for persistent data would make it possible for the system to learn and adapt, based on user input.

This would involve an additional database layer in the architecture. Security and privacy in this data layer would be of utmost importance, since it would hold confidential user information. A persistent data layer would allow for an experience that users can revisit and build upon. For example, a user could bookmark results from their session, and access those results on subsequent visits. In SAM, a user could store past graphs that were generated and run comparisons over time. Persistent data also provides the basis for customized user experiences that are modified and tailored to particular user needs, as outlined in Section 2.2.

With this being said, there are also potential drawbacks for storing and using persistent data. Users may want their sessions to be kept confidential, and they may not want to create accounts that store their data. In the case of the YU Services bot, students with disabilities may not want details about their disabilities, functional limitations, or even the services they use, to be stored on the system. In this regard, persistent data might hinder user acceptance, or might pose as a barrier for certain users. To ensure that the bot is embraced by the widest possible user base, users will not be required to create an account. If users do not want to store their data, they can simply use the bot anonymously.

It would be interesting to include a method to gather user data that can be used to improve the bot's answers and available information resources. For example, in Figure 6, the YU Services bot can explicitly ask the user for their opinion about their favorite university resources. The student's answer would be cleaned, stored, and used by the bot to improve future interactions with students. This data could be used to help determine the confidence of the answer the system returns, or it could be used to create new entities and intents altogether.

Bot: What is the best place you would recommend to other students for quiet studying?
Student: I like to use the quiet study lab in Scott Library
Bot: Why do you like that place?
User: I find it to be a great place because of its central location on campus.

Figure 6: A future feature that integrates knowledge from users into chatbot resources

The approach demonstrated in Figure 6 could be applied to all of the chatbots that share the common platform. Northrop *et al.* [35] explain that although some components and design decisions are initially specific to one product, taking an iterative approach with software product lines allows developers to propagate features to other products.

7 RELATED WORK

To better inform the decisions made during the development of the two bots, a number of alternate chatbots and platforms were investigated. This investigation helped gain an understanding of what features are popular and how users feel about chatbots in general. It also helped gaining insight to the tools other platforms offer.

Facebook's Messenger app is home for a number of popular bots [21]. A novel UI design choice among Facebook bots is the ability to return responses in the form of cards on a rotating carousel.

Take for example the Ticketmaster bot [38]. Concert results are returned as individual cards on the carousel. Each card includes buttons for users to purchase tickets, view more info, or share on their media feed. Users can swipe through each of the cards to find the concert listing they are interested in. This is a visually appealing design choice that also has practical benefits.

Another bot, Woebot, is an automated counselling agent that helps users monitor their mood and discuss mental health issues through Facebook Messenger. In a randomized controlled trial, Fitzpatrick *et al.* [18] found that Woebot could in fact be an engaging way to provide therapy to users. This study lends credence the notion that chatbots can be used for complex and sensitive tasks such as counselling. Another interesting aspect of Woebot is that while users get the first 14 sessions free, they must pay a subscription fee for continued use. It will be interesting to see if this is a viable approach for monetizing chatbots.

Bots on Facebook Messenger can be developed using the Wit.ai platform [13]. Wit.ai includes functionality to handle natural language processing, as well as speech-to-text and text-to-speech conversion. Much like Watson, Wit.ai also requires developers to define entities that are picked out of the user statements.

Amazon offers a suite of tools for bot development, including Lex for creating conversational interfaces [36]. A unique aspect of Amazon's chat bots is the ability to integrate with their hardware offerings, such as the voice-only Echo, and the new voice and screen-operated Echo Show. Lex allows developers to define the user intent. Along with intents, Lex uses a slot system for intent parameters. When users are engaging with the bot, it will prompt them to enter values for each slot that was defined for a given intent.

Google Assistant, based on api.ai [1] includes a "Smart Reply" functionality, which is an innovative design choice that provides users with shortcuts to bypass typing or speaking their response. Essentially, these are responses that are automatically generated for the user. For example, if the bot is displaying today's weather, the user can choose from a number of smart responses, such as "show me the 3 day forecast" or "what is the humidity index for today?". While this is a convenient feature that may work for some bots, there are practical reasons for requiring users to either type or speak their statements. In our case, requiring users to provide their own answers will be beneficial for two reasons. For one, it doesn't limit the user's response to a small number of stock statements. More importantly, since their responses will be used to analyze the tone and adapt the bot's statements, the more information a user enters, the more of a customized experience the bot can provide. Much like Facebook's Wit.ai, and Amazon's Lex, Google's api.ai development suite also centers around entities, intent and context. A notable feature of api.ai's entity system is the ability to add sub-classes of entities that the bot can recognize.

Besides bots and respective development platforms, this work also relates to the importance of personality aspects in this type of applications. Horzyk *et al.* [22] consider the business value of adapting bot responses to suit the personality traits of the current user. By using language that appeals to the user, they found that it is possible to increase user satisfaction, thereby increasing sales figures. Falling under the component of "adapting to user's personality type," this approach becomes an integral aspect of our model for the personality of our cognitive agents. Our model for personality of cognitive agents, however, takes into account a broader spectrum of static and adaptive factors, in order to induce a more natural flow of conversation.

Bos *et al.* [3] outline an approach to take when evaluating dialog techniques. While the paper was written in 1998, well before the inception of the bots active online today, their concepts are still valid and relevant. Their work asks important evaluation questions that developers can use to gauge how well their bots handle different categories of questions. This article informs the types of questions any bot should be able to handle, and illuminates potential limitations, or areas of improvement, that all bot developers should consider. The authors propose a checklist called "Trindi", which is composed of 12 yes or no questions that can be used to evaluate a bot's dialog model.

Kethuneni *et al.* [30] describe a personal health care assistant bot that exists in the virtual world Second Life. Their work proposes a virtual bot that follows around users in a game environment, offering suggestions and personal healthcare advice while the users are logged in. However, their proposed bot would only exist in the confines of the game. The bots we propose take this core idea and leverage the convenience of mobile technology such as cellphones and tablets, allowing the virtual bot to exist in a physical world.

Klopfenstein *et al.* [31] introduce the concept of "botplications", which look at bots as an indication of a possible paradigm shift that will impact how we currently conceptualize mobile apps. The paper explores the advantages of bots over traditional applications. Some of the concepts include bots as "threads", guiding the users through the conversation flow, and the value of keeping a chronological log of past conversations. This is a valuable concept for the bots covered in this paper, specifically SAM, where multiple systems can be monitored from a single application. In regards to the YU Services bot, threads would be an interesting concept to explore for students who choose to create an account and return to the bot multiple times. These students would rely on the information stored in their profile, and would be gathering useful data from multiple sources.

Bin *et al.* [32] explore the use of bots for DevOps. Their article is an important starting point for understanding how bots can be used to manage systems. Their paper also includes surveys that provide insight into other ways technical teams are using chatbots. Some notable uses include team and task management, customer support, and deployment support.

The tenets of a software product line are defined in Northrop *et al.* [35]. The article focuses on essential activities including core asset development and product development, While the tenets are applicable to software development in general, it will be interesting to apply this framework to bot development in particular. Van Grup *et al.* [40] explore the software product line concept of variability. We look to apply this to the development of multiple chatbots using a shared platform. Some notable variability points in the platform relates to the choices of user interface, the addition or absence of IBM services available through Bluemix, and the behavior of the principal self in the personality processing component.

There are a number of works focused on modeling the variablity and commonality in software product lines. Benavides *et al.* [2] provide a thorough survey on the automated analysis of feature models.

While the article is primarily concerned with the computer-aided extraction of information from feature models, it is a useful entry point to understanding the essential concepts of feature modeling. Feature modeling was first introduced in Kang *et al.* [29] in 1990. In 2000 Czarnecki *et al.* [15], built upon Kang's original concepts and extended the notation. This is the notation used to document the features in this paper. Czarnecki [16] further extended the feature modeling notation to include cardinality, amongst other changes.

8 CONCLUSION

In this work, we share our experiences with developing chatbots following a software product line perspective. We propose a reference framework for chatbot applications, which aim to function as virtual assistants. We claim that such an abstract development and architectural framework, by taking out the domain specific details, can generally facilitate and streamline the development of such applications en masse. This will also enable the transfer of knowledge and expertise between chatbot developers. We demonstrate the usefulness of such a framework in the development of two different chatbot assistants; a university services bot and a systems administrator assistant. The ultimate goal of the work is to implement the reference framework into a development platform to systematically guide and aid the development of such applications. The platform would provide the tools necessary to implement notions such as cognitive processing, personality adaptation, external source access and dynamic dialog structure, the importance of which we have identified in this work.

REFERENCES

[1] API.AI. 2017. What is API.AI. https://api.ai/docs/getting-started/basics. (2017). [Online; accessed 27-August-2017].

[2] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. 2010. Automated analysis of feature models 20 years later: A literature review. *Information Systems* 35, 6 (2010), 615–636.

[3] Johan Bos, S Larsson, I Lewin, C Matheson, and D Milward. 1999. Survey of existing interactive systems. *Trindi (Task Oriented Instructional Dialogue) report* D1 (1999), 3.

[4] Grant Clauser. 2017. What is Alexa. http://thewirecutter.com/reviews/what-is-alexa-what-is-the-amazon-echo-and-should-you-get-one/. (2017). [Online; accessed 27-June-2017].

[5] Watson Developer Cloud. 2017. About Natural Language Classifier. https://www.ibm.com/watson/developercloud/nl-classifier.html. (2017). [Online; accessed 26-June-2017].

[6] Watson Developer Cloud. 2017. About Text to Speech. https://www.ibm.com/watson/developercloud/doc/text-to-speech/index.html. (2017). [Online; accessed 25-June-2017].

[7] Watson Developer Cloud. 2017. About Tone Analyzer. https://www.ibm.com/watson/developercloud/doc/tone-analyzer/index.html. (2017). [Online; accessed 26-June-2017].

[8] Watson Developer Cloud. 2017. Building A dialog. https://www.ibm.com/watson/developercloud/doc/conversation/dialog-build.html. (2017). [Online; accessed 25-June-2017].

[9] Watson Developer Cloud. 2017. Context and State. https://www.ibm.com/watson/developercloud/doc/conversation/dialog-context.html. (2017). [Online; accessed 25-June-2017].

[10] Watson Developer Cloud. 2017. Overview of the Retrieve and Rank service. https://www.ibm.com/watson/developercloud/doc/retrieve-rank/index.html. (2017). [Online; accessed 26-June-2017].

[11] Watson Developer Cloud. 2017. Speech to Text - Convert human voice into written word. https://www.ibm.com/watson/developercloud/speech-to-text.html. (2017). [Online; accessed 25-June-2017].

[12] Watson Developer Cloud. 2017. Watson Tutorial. https://www.ibm.com/watson/developercloud/doc/conversation/tutorial-dialog.html. (2017). [Online; accessed 26-June-2017].

[13] Josh Constine. 2017. Facebook Acquires Wit.ai To Help Its Developers With Speech Recognition And Voice Interfaces. https://techcrunch.com/2015/01/05/facebook-wit-ai/. (2017). [Online; accessed 27-August-2017].

[14] Krzysztof Czarnecki. 1998. Generative programming: Principles and techniques of software engineering based on automated configuration and fragment-based component models. (1998).

[15] Krzysztof Czarnecki, Ulrich W Eisenecker, and Krysztof Czarnecki. 2000. *Generative programming: methods, tools, and applications*. Vol. 16. Addison Wesley Reading.

[16] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. 2004. Staged configuration using feature models. In *SPLC*, Vol. 3154. Springer, 266–283.

[17] Chad Edwards, Autumn Edwards, Patric R Spence, and Ashleigh K Shelton. 2014. Is that a bot running the social media feed? Testing the differences in perceptions of communication quality for a human agent and a bot agent on Twitter. *Computers in Human Behavior* 33 (2014), 372–376.

[18] Kathleen Kara Fitzpatrick, Alison Darcy, and Molly Vierhile. 2017. Delivering Cognitive Behavior Therapy to Young Adults With Symptoms of Depression and Anxiety Using a Fully Automated Conversational Agent (Woebot): A Randomized Controlled Trial. *JMIR Mental Health* 4, 2 (2017), e19.

[19] JS Foundation. 2017. About Node-Red. https://nodered.org/about/. (2017). [Online; accessed 26-June-2017].

[20] Boris Fritscher and Yves Pigneur. 2014. Business model design: An evaluation of paper-based and computer-aided canvases. In *Fourth International Symposium on (BMSD) Business Modeling and Software Design. Scitepress*.

[21] Marcellus Gaag. 2017. 30 Best Facebook Bots. https://chatbotsmagazine.com/25-of-the-best-facebook-bots-to-chat-with-f159bca02dce. (2017). [Online; accessed 26-August-2017].

[22] Adrian Horzyk, S Magierski, and Grzegorz Miklaszewski. 2009. An Intelligent Internet Shop-Assistant Recognizing a Customer Personality for Improving Man-Machine Interactions. *Recent Advances in intelligent information systems* (2009), 13–26.

[23] Adrian Horzyk and Ryszard Tadeusiewicz. 2009. A Psycholinguistic Model of Man-Machine Interactions Based on Needs of Human Personality. In *Man-Machine Interactions*. Springer, 55–69.

[24] IBM. 2016. WATSON Conversation - Entities. https://www.ibm.com/watson/developercloud/doc/conversation/entities.html. (2016). [Online; accessed 24-June-2017].

[25] IBM. 2016. WATSON Conversation - Intents. https://www.ibm.com/watson/developercloud/doc/conversation/intents.html. (2016). [Online; accessed 24-June-2017].

[26] IBM. 2017. Cloud Automation Manager. https://www.ibm.com/ca-en/marketplace/cognitive-automation. (2017). [Online; accessed 26-June-2017].

[27] IBM. 2017. What is Watson Conversation. https://www.ibm.com/watson/developercloud/doc/conversation/index.html. (2017). [Online; accessed 28-June-2017].

[28] Octavia J. Gutierrez-Garcia and Emmanuel Lpez-Neri. 2015. Cognitive Computing: A Brief Survey and Open Research Challenges. *2015 3rd International Conference on Applied Computing and Information Technology/2nd International Conference on Computational Science and Intelligence* (2015), 328–333.

[29] Kyo C Kang, Sholom G Cohen, James A Hess, William E Novak, and A Spencer Peterson. 1990. *Feature-oriented domain analysis (FODA) feasibility study*. Technical Report. Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst.

[30] Sukhanya Kethuneni, Stephanie Elizabeth August, and James Ian Vales. 2009. Personal Health Care Assistant/Companion in Virtual World. In *2009 AAAI Fall Symposium Series*.

[31] Lorenz Cuno Klopfenstein, Saverio Delpriori, Silvia Malatini, and Alessandro Bogliolo. 2017. The Rise of Bots: A Survey of Conversational Interfaces, Patterns, and Paradigms. In *Proceedings of the 2017 Conference on Designing Interactive Systems*. ACM, 555–565.

[32] Bin Lin, Alexey Zagalsky, Margaret-Anne Storey, and Alexander Serebrenik. 2016. Why developers are slacking off: Understanding how software teams use slack. In *Proceedings of the 19th ACM Conference on Computer Supported Cooperative Work and Social Computing Companion*. ACM, 333–336.

[33] Alessandro Murgia, Daan Janssens, Serge Demeyer, and Bogdan Vasilescu. 2016. Among the machines: Human-bot interaction on social Q&A websites. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. ACM, 1272–1279.

[34] Michael Newberg. 2017. Nearly 48 Million Twitter Accounts Could be Bots. http://www.cnbc.com/2017/03/10/nearly-48-million-twitter-accounts-could-be-bots-says-study.html. (2017). [Online; accessed 27-June-2017].

[35] Linda M Northrop. 2002. SEI's software product line tenets. *IEEE software* 19, 4 (2002), 32–40.

[36] Sarah Perez. 2017. Amazon Lex, The Technology Behind Alexa Opens up to Developers. https://techcrunch.com/2017/04/20/amazon-lex-the-technology-behind-alexa-opens-up-to-developers/. (2017). [Online; accessed 26-August-2017].

[37] Libby Plummer. 2017. The best bots of 2017. http://www.wired.co.uk/article/chatbot-list-2017. (2017). [Online; accessed 27-June-2017].

[38] Jonathan Shieber. 2017. Ticketmaster's chatbot for facebook is actually not terrible. https://techcrunch.com/2017/06/21/

ticketmasters-chatbot-for-facebook-is-actually-not-terrible/. (2017). [Online; accessed 27-June-2017].
[39] Slack. 2017. What is Slack - Getting Started. https://get.slack.help/hc/en-us/articles/115004071768-What-is-Slack-. (2017). [Online; accessed 26-June-2017].
[40] Jilles Van Gurp, Jan Bosch, and Mikael Svahnberg. 2001. On the notion of variability in software product lines. In *Software Architecture, 2001. Proceedings. Working IEEE/IFIP Conference on*. IEEE, 45–54.