Applications of Existing Organized Data, in the Creation of a Chatbot, Improving Customer Experience

Kristin Aleksandrova

University of Sofia St. Kliment Ohridski, Sofia, Bulgaria
kristinia@uni-sofia.bg

Abstract. In today's world, people expect instant reaction, when seeking a resolution to their issues. This principle applies to all industries. Sampling the IT and healthcare industries, we can see that there is a similarity in the currently available information and the way it is structured and the industry requirements for a fast and adequate response. The question is how much of that already existing information is usable in its current state. An obvious candidate to answer that question would be conversational interfaces.

The research presented in this paper is based on the existing SAP Product documentation, with a focus on the HANA Server Installation and Update Guide. The main objective is to develop a prototype of a generic conversational interface, which can navigate through data, structured for a human's ease of access, independently from the industry it currently operated. The prototype strives to open the discussion by carrying the naïve belief, that we can create one solution to resolve all issues.

Overall, creating a chatbot, that would guide a user through the lifecycle management procedures of interest, provides a new style of user experience for the target group, performing those operations, and is feasible with the current technologies available. This suggests a way of utilizing human readable structured information that can result in general improvements of the business-customer relationship and a reduction of the effort needed for general support. By relation, these principles can be applied to all industries, especially for lightweight cases.

Keywords: chatbot, support, experience, conversational, interface, documentation.

CEUR-WS.org/Vol-2656/paper18.pdf

1 Introduction

With the rapid development of technology, the expectation for software interaction is evolving. Most products and services usually offer several user interfaces with their variations, typically: command line (CLI) and graphical user interface (GUI). Currently on the rise is the natural language interface (NLI). Also referred to as conversational interface, it can be spoken or written. These days the written form, or chatbots, are in high demand. Software consumers now expect such an interface to be a natural part of every product, especially in the cases where a service is being provided.

Copyright © 2020 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

Chatbot development however is hindered not by technical, but by ethical problems. Let us take as a reference a general-purpose chatbot that will work with all the available data on diseases and medication. A software solution like that will be perceived as an omniscient authority that can be trusted. However, we are constantly finding new diseases, new symptoms, and complications for old ones and in addition, not all combinations between illnesses and their side effects are documented. This could lead to two major issues: self-diagnosis and self-treatment. In the case of self-diagnosis, based on what the user defines as symptoms, or is prompted to consider, a decision will be made. As most people are not licensed physicians, recognizing the right symptoms is already a problem. On the other hand, if the diagnose is already available there is a risk that the wrong treatment is chosen, since there are factors like drug compatibility and personal and locational factors, to be considered. Not to mention, the overall lack of empathy such a solution would have. Regardless of the choice of words and phrasing, a user will always be aware that this is software. Similar opinions were also observed by a web study that asked physicians their opinion of the applications of chatbots in healthcare. [1]

This by no means implies that there is no application for chatbots in this field. If we were to remove the described ethical issues, we will see great benefits. One example is a chatbot, dedicated to cancer patients. It can answer all questions related to a patient's treatment plan, the expected side effects and overall progression of the illness during its phases. In this case, there is no self-diagnoses or self-treatment and the lack of empathy is capitalized upon, as there is no judgement towards what you ask. This simply proves that chatbots have their place, when moral ambiguities are removed and there is a clear purpose for their creation.

Usually the process for chatbot creation is as follows: define the problem area for your chatbot, gather the necessary data, clean and restructure it. This creates multiple structures and models for the same dataset and makes reusing and freely combining them harder than creating a new structure for every case. This is hardly necessary for each problem to be resolved. To prove that I will reverse the creation process and start by asking what data is already naturally available and structured in some form.

1.1 Problem area

We as humans like to organize data; technically speaking all data conforms to some structure. There is a clear expectation on how a book or manual would look like in comparison to an article or a text message. Therefore, it would be a matter of understanding the principle of the data structure. Since that is not the focus

presently, let us look at a structure that would be easy to explain to an algorithm. Namely, product documentation.

Product documentation is usually written in some markup language to ensure continuity and proper visualization. There are set rules on the meaning and proper usage of words in the context of a product and that ensures no ambiguity when looking at one specific documentation. That of course changes as between companies and authors that notion differs. The main issue with documentation is that in most cases it is been reorganized and reworked a few times. Presently we are left to work with a hybrid combination of all those versions. Therefore, it's key to know the historic evolution of the structure and reflect it properly. A minor issue to consider is that there are multiple authors and they have a personal preference to a markup syntax. This is mitigated by the requirement of a consistent look and can be added to the logic of an algorithm.

Now that the data is present, the questions is what problem can product documentation solve? There is always the possibility to create yet another search engine. Considering most documentations are properly indexed and searchable form the internet, we can consider other options. An interesting case is troubleshooting, as usually the documentation is read after a problem occurs. However, there is one page per solution for troubleshooting, unless you record every interaction with a product. A better data for that problem would be blogs and user forums that go into the details of the issue and its resolution. In the end, I settled for procedure guidance.

When talking about long running procedures, such as system or product updates, the product documentation could be 200-300 pages. That volume of data is hard to comprehend on one read, especially as each sentence has a warning for the future. So, the usual situation is:

- a user triggers a procedure and gets an error somewhere in the execution, the documentation is of no help due to the sheer volume;
- a support ticket is opened to resolve the issue;
- since support queues are at times overloaded the likely resolution is to read the documentation, as the proper execution is explained there;
- this answer takes time and brings no value, hitting the instant gratification problem;
- the customer experience is very poor as the procedure was not executed and the user was not able to help themselves.

To improve the customer experience, we need to look at the instant gratification problem. To quote Baron [2]:

"The industry numbers speak for themselves: 60% of respondents believe that one minute is too long to be on hold, 42% complain about the need to speak to different agents, and 78% terminate contracts because of bad support. This amounts to $1.6tr annual losses due to poor customer support in the US alone."

Hiring new people is not a scalable solution, especially to fast growing businesses, therefore the answer lies in automation. The positive aspect is that customer incidents follow the 80/20 rule. Meaning that 80 percent of the problems, customers face, are similar and simple enough for their resolution to be automated and for the other 20 percent a human involvement and a deeper look is needed. If we combine that with the need of instant support and the growing demand for personalized experience, we get conversational interfaces as a potential solution again.

The goal of this research would be to create a prototype that would serve as an evaluation of the already existing human-readable organized data. The premise would be to understand if there is a need for a sophisticated preprocessed knowledge database for lightweight cases.

2 Building a prototype chatbot

Once our goal has been defined and we have our initial research, it is time to start building a prototype. In order to have a productive chatbot, we want to build it in a way that can scale the three P's, which are three core ideas to keep in mind when developing a conversational interface. Namely personality, positioning, and proactivity. Abiding those allows creating an identity for the chatbot and setting the right expectations in its future user base, so that we minimize unhandled situations. To ensure that, we will use an existing bot platform or framework. After searching the field, the choice is SAP's Conversational AI.

2.1 Conversation flow

Let us imagine how the conversation would look like (Fig. 1). In blue is the user input, we would analyze to understand the user intention. The green blocks represent a response generated by the chatbot. It can be a simple reply in the case of "Greetings" or an inquiry for additional information, as is the case with "Ask for target version". The purple rhombs represent conditionals that need to be reflected in the algorithms we design. This can be part of the SAP Conversational AI platform or part of the server side, considering that in yellow are the calls to the server, it is obvious where to check those points.

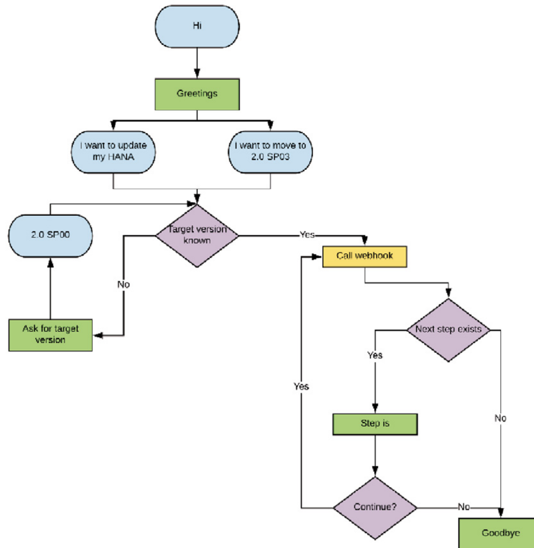

Fig. 1. Sample conversation flow for the HANA update guidance bot.

2.2 Data gathering

Let us acquire all the needed data. There is no public API to SAP's documentation, which leaves us with crawling for the needed information. If we are to write a crawler however, first we need to ensure that the website allows this kind of data extraction and has planned a sufficient server size to accommodate the load from an automated solution. All websites provide this information in their robots.txt file. This check can be integrated as part of the chatbot itself, however considering our objective, we will not leave the premise of SAP's help portal, and therefore a manual check is sufficient (Fig. 2).

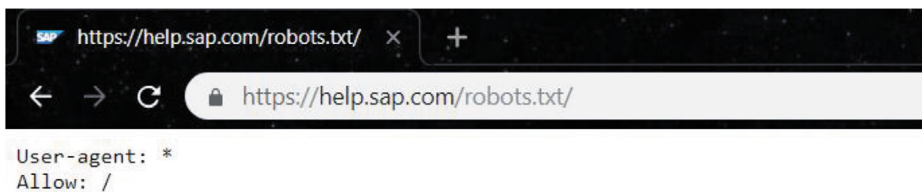

Fig. 2. SAP Help Portal – robots.txt.

We see there are no limitations for agents, and therefore no additional constraints are to be implemented for our solution.

SAP's documentation is based on SAPUI5, which is means that the documentation pages are JavaScript based and the actual content we are interested

in is loaded after the defined, in the page, JavaScript code is executed. However, in most cases, the documentation supports a version that does not require rendering and instead provides the html itself. The HANA Server Update documentation is one of them. Opening them side by side we see that while this solves the need of preloading and adding additional weight to the chatbot, the table of contents is now missing. Which makes us reliant on each page is content and links only. The main difference in the consumption of the documentation's pages is a slight change in the URLs themselves. Namely crawling "https://help.sap.com/doc/..." instead of "https://help.sap.com/viewer/..."

2.3 Recognizing user intent

To build the procedure steps from the documentation, first we need to recognize the intent of the user to perform the procedure. As described in the SAP Conversational AI documentation, we start by creating a custom entity recognizing a HANA version was mentioned in the sentence.

In compliance with the platform recommendation we've kept the entities simple and distinguishable, only adding one for the HANA version, keeping in mind that the version is actually comprised of two parts the HANA major version, that could be 1.0 or 2.0 and the SP level, so in the end a HANA version looks like "2.0 SP00". Separating those however would make it hard to handle the cases where the user provides their starting and target version in the same sentence. As they follow a numeric sequence, getting the higher version would be sufficient in the current implementation. In the case of separation, we risk mismatching the versions and SP levels. So instead of moving a HANA 1.0 SP12 to HANA 2.0 SP00, we could be trying to do a HANA 1.0 SP00 to HANA 2.0 SP12 migration, which in the example would fail, as the target version doesn't exist currently, however if that was not the case we would be providing completely wrong information from the beginning.

Afterwards we create a skill, called hana_update, that is triggered when the custom entity hana_version is present, as currently we want to focus solely on the update procedure - this is sufficient, however to include additional HANA procedures it would be easy to add a new entity, that helps recognize the intention of the user.

Currently the logic implemented is completely independent of the product. Besides the custom entity that we use to recognize the intent, we mention nothing about update or HANA; this is achieving the initial goal of perusing a general, unconstrained solution. However, we find ourselves in the situation that we heavily depend on the exact structure of the documentation, and even when we stay in the premise of the HANA update documentation, in between versions there are inconsistencies as missing links or adjustments in the

structure. To end, we rely on the assumption that all procedure steps are in the same subsection.

2.4 Finding the right documentation pages

Now that we have recognized that the user wants to update a HANA system in the platform, we can start building, in the backend, the tree of pages that describe the procedure. We start with the initial page for the update procedure. It's easy to notice that the target version, which is also containing the documentation of interest, is part of the URL:"https://help.sap.com/viewer/2c1988d620e04368aa4103bf26f17727/2.0.00/en-US/a428e6802a454f34bd3599782060c116.html", and changing only that part in the URL is sufficient to get the starting process page for our desired version. While for each version there are several related links, provided in the initial page, not all of them represent steps and not all of them are mandatory, so how do we handle their content?

We design an algorithm, reflecting a simple approach, to parse a page that does not contain procedure steps. We look on paragraph level and we look at all paragraphs in the html that have a tag <cite class="cite"> … </cite>. That translates into paragraphs that have a citation of a related link. Therefore, if a paragraph is describing a mandatory step, but it is not providing any additional information on how to execute it, the chatbot would not mention it, as it cannot provide additional information and guidance without shifting its focus to a search engine. Additionally, there are times where paragraphs have citations, but the information is not linked in the related links. To summarize we look at each paragraph that has an existing link in related links, tagged as a citation:

1. Look for conditional statements that imply this is an optional step dependent on the customer's specific setup. For example: "If you (need)…", "If you would like…", "In case you…"
2. Look of imperative statements that imply the step is mandatory. For example: "…required…", "…have to be fulfilled…", "…must be done…"
3. Other paragraphs usually have a "for more information…" style approach, so those consisting statements like that and all other, we consider optional steps. The condition of their execution being, the user's desire to do so.

The order of those conditions is of importance, as the agent needs to be able to handle statements of the kind: "…If you…it is required…"

This is our work on the initial page; we can now start with the next pages. Following the linked next pages, we notice that some steps of the procedures have multiple ways of being executed, and those ways are added one after the other. For steps, with similar naming, suggesting they describe the same step, like: "Download Components from SAP Service Marketplace Using the SAP HANA Studio" and "Download Components from SAP Support Portal Using

the Web User Interface", we can try and implement some logic to deduce they are the same step, for example a variation of TF-IDF and cosine similarity. One major issue with that however would be that steps like "Before updating" and "Updating" would get similar rates. Therefore, working with a combination of a step title and description can be beneficial.

Unlike the next pages where the link was always part of the same tag object in the same place, the procedure steps are placed in different parts of the pages, depending on the existence of prerequisites or paragraphs before and after. While the structure varies, in the observed pages of documentation, a rule of thumb is that the name Procedure is added in some tag and the steps themselves are organized as an organized list, or in other words the … tag.

2.5 Objects and data structures

The documentation already has a tree structure, translating this into computer logic is trivial; the question is how useful said tree would be. What we notice is that we rely on properly designing the tree, and representing each step on its rightful level, as optional steps positioning in the tree structure is not always clear. They may need to be executed at an exact time, therefore being represented as a level on their own, which makes skipping over them a case to be covered or they could be on the same level as a mandatory step. Additionally, according to our definition of optional, having multiple ways described, for the same step, makes all of those pages - optional steps, yet choosing one of them is mandatory. If we consider the user's perspective, it would be better to see all options and let an actual person judge their similarity instead of arbitrarily miss clearly linked steps.

As the functionalities are implemented, it becomes clear that in most cases representing the tree as a stack is quicker and easier to work with. To construct the aforementioned stack, each html file is represented as an object with a few properties. We have added the property name, which allows us to introduce a naming convention and distinguish between the origins of the object. Whether it is a procedure step, its deriving is not currently implemented, or a general step that points to a new documentation page derived from a related link or a next step. The property url, contains the URL used to access this page, with the addition of several checks it ensures that we don't create a cycle in the tree of steps, the properties question and flag are related to the previously explained algorithm, used to compile the necessary documentation pages. In the cases where we have an optional step, we generate the question needed to define its execution. For example, while parsing the paragraphs, if we come across a conditional statement, we transform it to a question that is then carried with the object. The existence of that question is the defining factor of its optional nature. In the case where the

paragraph has the "If you need… statement", that related link would be added with the question "Do you need. An important thing to note is that all questions are formed in the way that a positive user reply would indicate the execution of said step, while a negative one would skip it.

2.6 Response algorithm

The SAP Conversational AI platform can consume responses from the backend as long as they abide to the specified format [3]. The backend sends the message in a JSON format, specifying the type (e.g. text, picture, buttons, etc.), we use this to return the next step, that should be executed.
After the update intent has been recognized, the platform calls the webhook that constructs the stack of steps, by the target version available. Afterwards the backend returns to the chatbot itself and waits for confirmation that the user would like to be guided through the procedure.

There are two states recognized afterwards, the user gave a positive reply and the user gave a negative one. Both states have an entity defined and a skill. In addition, we introduce in the backend-maintained objects the properties procedureFlag and the procedureQuestion. Both are True/False flags with different purpose. The procedureFlag indicates whether the agent detects Procedure steps in the next page. It is based on that the decision to propose said breakdown is made. On the other hand, the procedureQuestion is equivalent to the questionFlag, as it shows that we already asked this question and should now process the reply.

All steps are currently kept in a stack, the simplest logic being that once we provide a step, we pop it from the stack. There are several cases when the user gives a positive answer and the chatbot needs to make the corresponding decision.

The first one is the user just executed step n and is now confirming they would like to continue by receiving information regarding step n+1, however that step could be an optional one or a mandatory one. As we previously outlined optional steps are distinguished by the existence of a question in their object description. Therefore, in this case we need to provide the next step if it is mandatory and pop the stack or ask the question of the optional step and leave the stack as it is. Since we leave, the stack as it is there is no way to differentiate this case from the second one, which is why we raise the questionFlag.

The second one is the user has executed step n and step n+1 is an optional one. The positive reply is in response to the question of the optional step, as we know from the convention applied when forming the question this indicates, the step will be executed, and we proceed as if it is mandatory. To deduce that the question has been asked we check the questionFlag, an optional step with a false flag needs to provide the user its question, while an optional step with a true flag would adopt behavior dependent on the user input.

The logic for a negative reply is similar; we start by checking if there are any steps left in the stack, as the last one could have been optional. Afterwards the user may be replying to the question of the optional step or choosing to stay on the current step and ask for additional help.

Every time we introduce procedure steps for execution, we merge a list of them with the current list of steps. To do that said steps need to be represented as HTMLObjects, yet we would like to differentiate and not just by name. As a compromise, we create the objects with the URL of their page and we add the actual procedure step in the title property, as anyways in the already implemented response algorithm we provide the user with the page titles and a link to see them in the official documentation.

2.7 Architecture

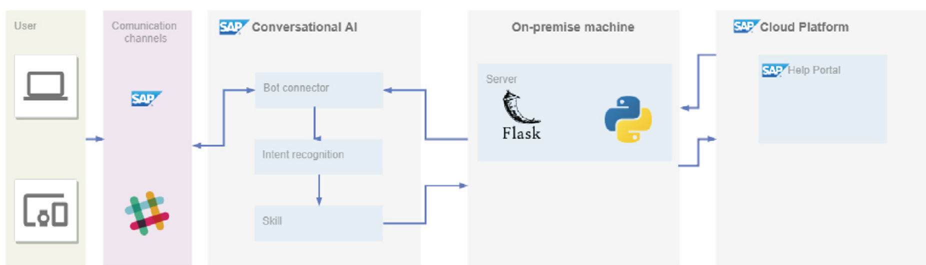

Fig. 3. Architecture of the HUG bot**.**

1. The user interacts using a communication channel of choice, currently the supported options are the SAP Conversational AI platform and Slack.
2. The request if forwarded to the bot connector in the platform.
3. The bot connector provides the message content to the intent recognition component.
4. Recognizing the intent triggers the skill connected to it.
5. The skill calls a webhook to the on premise server, running on python and flask
6. The server calls the SAP Help Portal, residing on the SAP Cloud Platform, to get the documentation.
7. Based on the documentation a response is formed and returned from the server to the bot connector.
8. The bot connector provides it to the communication channel, where the user can see it.

3 Conclusion

This research strived to explore the already existing structured data in the world around us, without changing it in any way and as this is the starting point for a bigger discussion, there is no additional labeling or context added. Objectively speaking, the process of cleaning and organizing knowledge is the most time and resource-consuming task when working on a problem space in a specific industry. In addition, while going for a one solution fits all approach sounds unrealistic, there could be principles that are relevant across different industries.

To find those principles we develop a conversational interface that uses the already existing structured documentation to guide a user through a standard technical maintenance process. To summarize our findings, generating a procedure from the existing documentation is dependent on the style and formatting of the documentation. As usually that is done by humans, there are expected inconsistencies, that only a machine can stumble upon, such as changing the naming convention in the source page tags. To go around that and consume the pages in their natural state, regardless of those inconsistencies, requires significant investment in text analysis, extending on the concepts, currently applied, for example when detecting action items in mail communication. What was proven with the HUG bot is that as long as the scope is kept narrow, we can rely on a rule-based approach, reinforcing again the current positioning of conversational agents, as described in the introduction. Comparing the effort needed to define the specific scope rules and implement them and the effort needed to invent new algorithms handling proficiently the recognition of actionable steps, with their optionality in mind and preserving their order and priority, it is clear which would be the quicker win from a business perspective.

Creating the user interface was simplified by the existence of a framework as SAP Conversational AI; however, it is obvious that chatbot creation, even when aided by such tooling introduces a new way of thinking when designing software. The entity, intent, skill trio is not specific to the SAP solution, instead it is the current industry standard for platforms providing such capabilities. Therefore, mapping the expectations of the interface and the capabilities of the procedure generating functionality needs to be reflected in the defined format a procedure would be transferred from one side to the other.

In conclusion, creating a chatbot, that would guide a user through the lifecycle management procedures of interest, provides a new style of user experience for the target group, performing those operations, and is feasible with the current technologies available. However, to ensure that the solution is production-ready and suited to the business needs, it needs to be clearly positioned with the right context provided, in the sense of user background, solution capabilities and brand digital identity. To do so it needs to utilize the capabilities of conversational agents,

while keeping an open mind, concerning automation of the business logic. As now, the effort for handing this over to a machine is still higher, when combined with its error rate, than having a person manually define this, especially in cases where this business logic is executed only once, e.g. the HUG bot's procedure steps generation.

To go back to where we started, using such a solution would prove to be an ethical issue. There is always the risk of problems and inconsistencies between versions of documentation as it is written by humans. Introducing a procedure guidance chatbot would improve the customer experience greatly as the fast response from an authority would solve the instant gratification problem and allow for a more latent customer, since the most common issues can be described there. However, documentation is prone to human errors and the chatbot would not be able to recognize those and instead of helpful, will prove harmful. In addition, while this is a risk to be calculated on a case-by-case basis, it is undisputed that there was no need for a new data model, another restructure or re-tagging of the documentation to resolve this lightweight case.

References

1. Palanica A, Flaschner P, Thommandram A, Li M, Fossat Y, Physicians' Perceptions of Chatbots in Health Care: Cross-Sectional Web-Based Survey, J Med Internet Res 2019;21(4):e12887, DOI: 10.2196/12887, PMID: 30950796, PMCID: 6473203
2. Baron, J., 2018. Why You Need To Automate Your Customer Support With Chatbots, https://cai.tools.sap/blog/customer-support/, last accessed 2020/04/21.
3. SAP Conversational AI, 2018. Official documentation, https://sapconversationalai.github.io/docs/concepts/intent, last accessed 2020/04/21.