Smart and smarter: Overcoming limitations of rule-based chatbots

Chatbots have come a long way since the first one was introduced in 1966. However, to become a truly helpful user support channel, they still have a long road ahead of them. Knowledge maps, user context and structured content can help make a difference.

Text by Alex Masycheff

Image: © Sebastien Decoret/123rf.com

The first chatbot

The first chatbot was built by Joseph Weizenbaum, an American computer scientist from the Massachusetts Institute of Technology, in 1966. The bot was called Eliza, in reference to Eliza Doolittle of Bernard Show's *Pygmalion*. To make Eliza understand the user's questions and give answers, the bot used a script that consisted of pattern-answer pairs. If a user's question matched one of the pre-defined patterns, Eliza would respond with the matching answer. One of the most famous scripts was called Doctor. It emulated a conversation with a psychotherapist. Ironically, while Weizenbaum wanted to show the superficial nature of interaction between computers and humans, quite a few people believed they were talking to a human when chatting with Eliza. Weizenbaum's own secretary asked him to go out of the room when she was talking to Eliza saying that she was having a private conversation!
Here's a famous example of a conversation with Eliza:

Human: Men are all alike.
Eliza: In what way?
Human: They're always bugging us about something or other.
Eliza: Can you think of a specific example?
Human: Well, my boyfriend made me come here.
Eliza: Your boyfriend made you come here?
Human: He says I'm depressed much of the time.
Eliza: I am sorry to hear that you are depressed.
Human: It's true. I am unhappy.
Eliza: Do you think coming here will help you not to be unhappy?
Human: I need some help, that much seems certain.
Eliza: What would it mean to you if you got some help?
Human: Perhaps I could learn to get along with my mother.
Eliza: Tell me more about your family.

At first sight, this conversation looks quite natural. However, if you take a closer look, you'll notice that all Eliza did was repeat the user's words and respond to them with a very limited set of phrases. This means that if a user's question didn't match any of the pre-defined patterns, Eliza wouldn't be able to answer. This made Eliza unable to scale because it would be virtually impossible to pre-define all possible questions and responses that the user might potentially provide.

Why a rule-based approach is not scalable

Most of the chatbots that exist today are still based on the same approach. Of course, they are much more flexible, but they still require question-answer pairs to be defined explicitly.
To make the situation even more complicated, think about the many ways to ask the same thing. For example, here are just a few variations (also known as utterances) of the question "How to print a document":

- What should I do to print a document?
- What does it take to print a document?
- How can I print out a file?
- What's the procedure for printing?
- I need to print a document. What steps should I take?

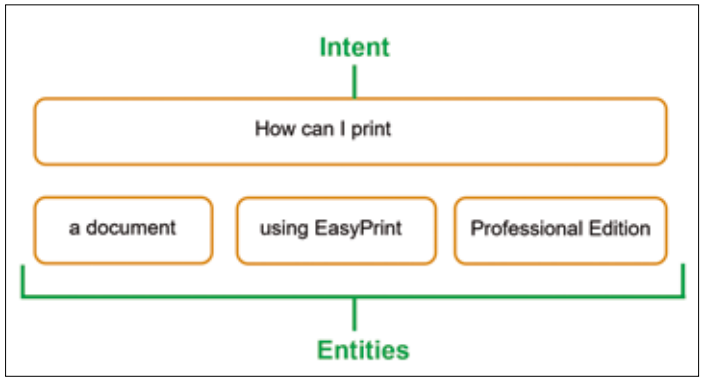

Figure 1: Intent and entities

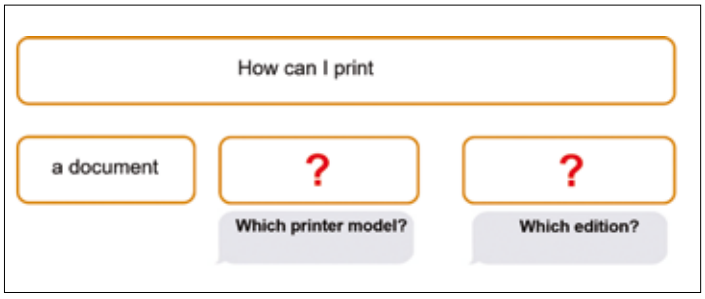

Figure 2: Some entities are not provided

You can probably come up with a dozen variations of one single question. Now think about how many other issues are usually covered in product documentation, and you'll realize the amount of effort required to identify not only the issues themselves, but also all possible utterances.
On top of this, there is a good chance that there are multiple products or different flavors of the same product, and the printing procedure is different for each of them. First, this means that question-answer pairs have to have multiple branches depending on the printer model or flavor. Second, to make the chatbot give a relevant answer, you have to somehow identify which information is missing in the user's request. For example, if the user asked "How can I print a document?" and the printing procedure is different for different models of the printer or for different types of documents, the chatbot should be able to ask what model or document the user is working with.
Identifying intents and entities is how this challenge is now solved. The "intent" is the goal that the user wants to achieve (see Figure 1). The "entity" is a parameter of the goal. In our example, the intent is to print a document. The entity is a document (the printing procedure of an image might be different). A separate entity might represent the printer's flavor, model, or edition.
If any of the required entities is not provided in the user's request, the chatbot has to ask the user about the value for this entity (see Figure 2).
When the amount of content is not big, and the complexity of variations is not significant, all possible intents and entities for each intent can probably be manually defined.

Otherwise, maintaining such a chatbot will require too much effort.
An additional problem is how to provide the user with comprehensive and useful information, even if the user didn't explicitly ask for this information.
A challenge of the information age is that we don't know what we don't know. The user doesn't necessarily know which question should be asked to achieve a goal. Therefore, if the user doesn't ask a question explicitly, the chatbot won't give an answer, although the information could be important and helpful for the user.
Below, you will find a solution that consists of three components:

- Knowledge map
- User's context
- Structured content

Knowledge map

A knowledge map describes the subjects of the domain and relationships between them.
Suppose a company produces three models of a printer: Basic, Pro, and All-In-One. The following is known about these printers:

- Basic and Pro are ink-jet printers. All-In-One is a laser printer.
- Basic and Pro are connected to the computer via a regular cable. All-In-One has both a cable and WiFi connectivity.
- The following issues might occur on all printers regardless of the model: the printer doesn't print, there is a paper jam, and the print is too light.
- If the printer doesn't print, it might be related to connectivity problems (for example, the cable is not plugged in or the Wi-Fi connection is lost).
- If the print is light, the cartridge has to be replaced, and we want to give the user the option of ordering a new cartridge right away.
- Troubleshooting procedures are different for different models, so information about the user's model is required to provide the user with the relevant troubleshooting procedure.

Figure 3 shows how a knowledge map for this domain can be constructed.

As you can see, the knowledge map defines the types, model, and connectivity options as well as possible issues that might occur. It also shows how each of these options relate to each other.
The knowledge map can be used to automatically generate questions about information missing in the user's request and to navigate the user through options.
Let's suppose the user typed: "My printer doesn't print." After locating this issue in the knowledge map, the chatbot crawls through the relationships that this issue has.
The chatbot identifies that this issue is related to all three models (see Figure 3). To disambiguate this situation, the chatbot can prompt the user to choose the model from the list of models defined in the knowledge map. Furthermore, if the user responded that the model is All-In-One, by navigating through the knowledge map, the chatbot can now find that the issue might be caused by connectivity, and All-In-One has both cable and Wi-Fi connectivity. Because the troubleshooting procedure for Wi-Fi and cable connectivity is different, the chatbot again needs to disambiguate the situation by asking which connectivity option defined in the knowledge map the user is using.
In this example, the chatbot generated questions automatically based on the structure of the knowledge map rather than on manually defined entities specified for each question.
Let's consider another situation – the user saying: "My ink-jet printer doesn't print". Based on the structure of the knowledge map (see Figure 3), the chatbot can identify that there are two ink-jet models: Basic and Pro. To disambiguate, the chatbot should prompt the user to choose from these two options without even suggesting All-In-One, because that printer uses laser.
The knowledge map can also be used to provide the user with relevant information even if the user didn't explicitly request this information. Suppose that the print is too light. It might be an indication of low ink in the cartridge, so it would make sense to offer the user to order a new one. This can be done by defining in the knowledge map a relationship between the issue and cartridge ordering information.

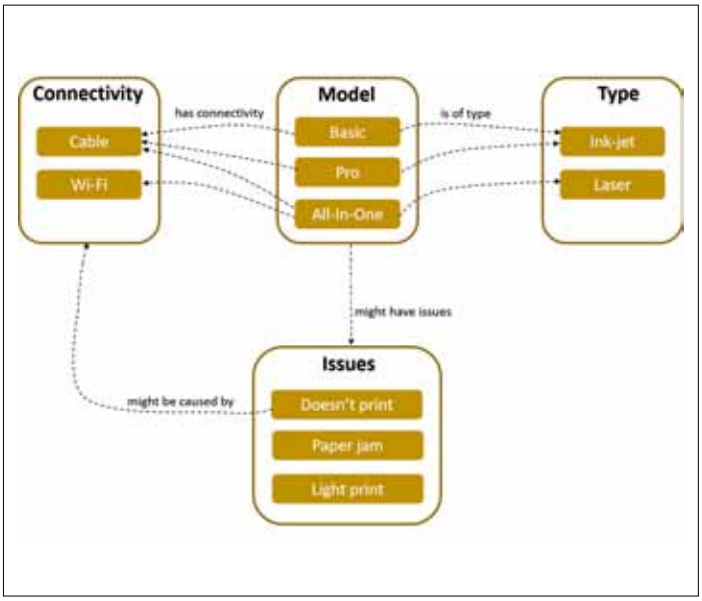

Figure 3: Knowledge map describing printers

Tools for building knowledge maps

Knowledge maps can be represented as ontologies in a knowledge representation format, such as Web Ontology Language (OWL) or Resource Description Framework (RDF). Because an XML presentation exists for both formats, this makes knowledge maps processable by machines.
There are various ways (and their combinations) to build a knowledge map:

- Manually, using ontology editors, such as Protégé (open source) or Fluent Editor (free for individual developers, open source projects, and academic research), or ontology APIs, such as Jena Ontology API or OWL API
- Automatically, using natural language processing and Artificial Intelligence engines
- By reusing existing ontologies

User's context

The information gathered about the goal the user wants to achieve, along with the goal's parameters (such as product model or connectivity type), is known as the user's context. The user's context includes various elements that may come in all kinds of variations.
For example, the user may experience problems with printing while using the All-In-One model connected to the computer via Wi-Fi.

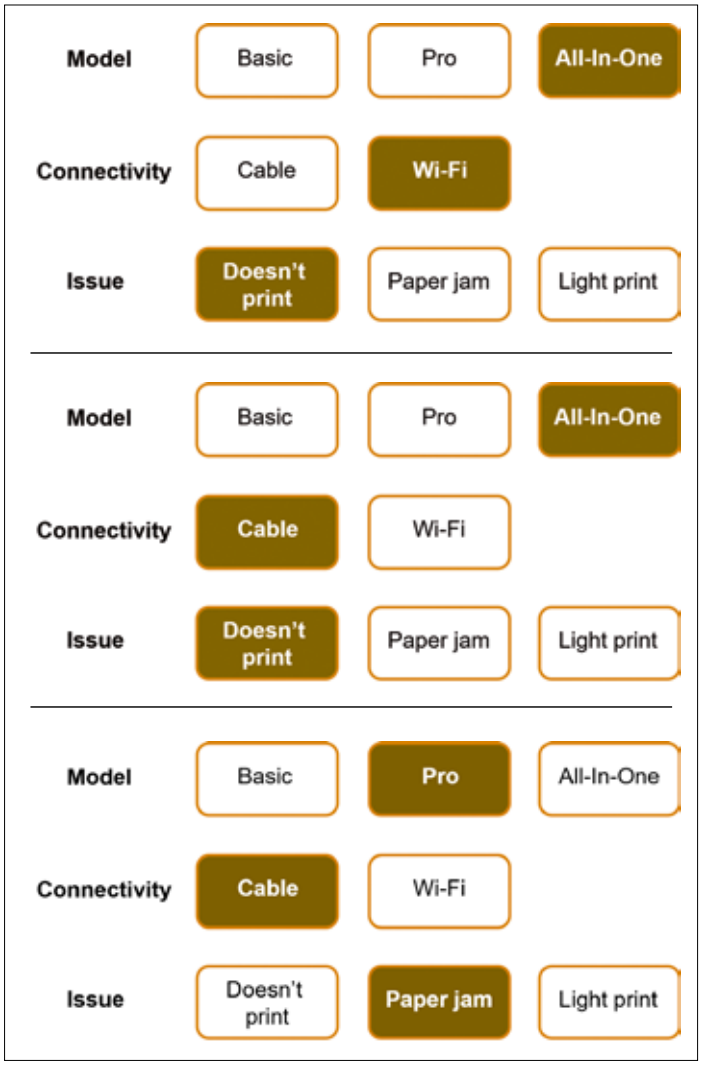

Figure 4: Different combinations of the user's context elements

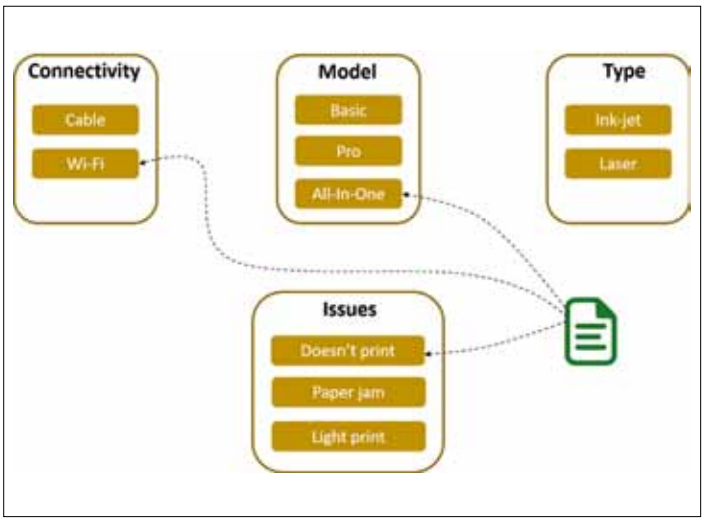

Figure 5: Content describing the issue when the All-In-One model with Wi-Fi connectivity doesn't print

Or the user's All-In-One printer may fail to print while being connected via a regular cable. Or the user may get paper jammed with the Pro model.
Figure 4 demonstrates different combinations of the user's context elements.

Structured content

As you can now see, the user's context is granular.
To find the content that precisely addresses the given combination of the user's context elements, this content itself has to be granular too. In addition, each content granule has to be semantically marked up to indicate the model, connectivity type, and issue it describes. This markup comes from the knowledge map. Semantic markup also makes content processable by machines. This is what structured content is all about (see Figure 5).
Because subjects in the knowledge map are linked to each other, by associating pieces of content with subjects in the knowledge map, we also connect these pieces to each other. This capability opens rich opportunities for navigating the user through the relevant content.

Conclusion

Rule-based chatbots are relatively easy and cheap to implement, but they can cover only the following simple scenarios:

- The amount of content is not big, and it's not growing, and thus, doesn't require much maintenance.
- The amount of content variations and their level of complexity is not significant.
- The scope of the domain is narrow.

Otherwise, both the initial configuration and ongoing maintenance of the chatbot will become very expensive and error-prone. Building a chatbot for complex scenarios requires building an information architecture to organize both content and knowledge. A combination of knowledge maps, user's context, and structured content can be a foundation for such architecture.

ABOUT THE AUTHOR

Alex Masycheff is CEO of Intuillion Ltd. that develops solutions for managing, automating, and delivering content. Alex has been in the content industry for over 20 years. He lead implementation of XML-based solutions in many companies, including Kodak, Siemens, Netgear, and EMC.

@ alex@intuillion.com
http://intuillion.com