

An Architecture-Independent Data Model for Managing Information Generated by Human-Chatbot Interactions

Massimiliano Luca¹^a, Alberto Montresor¹^b, Carlo Caprini² and Daniele Miorandi²^c

¹*Department of Information Engineering and Computer Science, University of Trento, Via Sommarive 5, Trento, Italy*

²*U-Hopper S.r.l., Via R. da Sanseverino, 95, 38122, Trento, Italy*

Keywords: Chatbot, Data modeling, Data representation, Knowledge representation

Abstract: This paper introduces a data model for representing humans-chatbots interactions. Despite there are many models that allow representing the usage and the behaviour of bots, a service that can store information from any conversational agent regardless the architecture is still missing. With this work, we introduce a general-purpose data model to store both messages and logs. To succeed, we raise the level of abstraction of the other analyzed models and we focused on the core logic of chatbots: conversations and interactions with the users

1 INTRODUCTION

In the last few years, the development of new technologies such as Artificial Intelligence (AI) led to a drastic change in the way people interact with services and companies. In particular, chatbots have become more and more popular, and they are redefining interactions in many different fields. According to Market Research Future, the usage of chatbots will increase up to 37% between 2016 and 2023 (Future, 2019). The first chatbot prototype, ELIZA, was developed in 1966 by Joseph Weizenbaum (Weizenbaum, 1966), with the goal of emulating a psychotherapist. In Weizenbaum's project, the bot creates the answers by looking for specific patterns in the user request. Now, bots create their responses using Machine Learning (ML) and Natural Language Processing (NLP), leading to a significantly higher quality of conversations. For this reason and due to the obsession of the companies for customers, bots are now everywhere. This recent technology is now used for customer service and marketing (Chakrabarti and Luger, 2015), (Verhagen et al., 2014), (Gnewuch et al., 2017), healthcare (Kethuneni et al., 2009), (Brixey et al., 2017), lifestyle (Fadhil and Gabrielli, 2017), education (Heller et al., 2005), (Kerlyl et al., 2007), tourism (Nica et al., 2018) and so on.

In the last few years, the number of platforms


for creating conversational agents increased as well. Diederich, Brendel and Kolbe counted 51 different services within this scope (Diederich et al., 2019). Some of them can be used to analyze the usage and the performance of a chatbot hosted on a third-party platform, while others provide all the tools needed to create a bot (e.g., Dialogflow or IBM Watson Assistant). The problem is that, at the moment, there is not a common data model for representing the interactions between users and chatbots. In other terms, each platform represents messages and conversations in different ways.


After an analysis of the existing models, in this work we define a general-purpose and platform-independent data model to represent the interactions between users and chatbots. The final goal is to have a representation that developers and researchers can use to integrate data coming from different platforms.


In Section 2, we describe how other services faced this issue and we analyze the proposed models. In Section 3, we discuss our solution by focusing on the messages (Section 3.1) and logs (Section 3.2). In Section 4, we focus on how three different chatbots with different architectures used our proposed data model to store the interactions generated by their bots.

2 RELATED WORK

The spread of platforms that allow developers to analyze the behavior of a chatbot correlates with the

^a <https://orcid.org/0000-0001-6964-9877>

^b <https://orcid.org/0000-0001-5820-8216>

^c <https://orcid.org/0000-0002-3089-977X>

numbers of data models proposed. The platforms listed in (Diederich et al., 2019) usually do not provide the data model used to store the interactions. This is mainly because these platforms have their core business in letting companies develop and deploy a bot. Other platforms, instead, are focused on chatbot analytics and usually share a way to integrate a bot in their systems. We used such platforms to create a general model and to compare the expressiveness of the representations. In particular, we selected five analytics platforms: Botanalytics, Botlytics, Botmetrics Dashbot and Chatbase. For each of them, we discuss their proposed models. There are also some services that cannot be taken into account because their models are too tailored for specific chatbots (e.g. Facebook Analytics for Messenger Bots).

*Botanalytics*¹ is a platform that allows analyzing chatbot data to get insights regarding the engagement of the users. It supports 19 different platforms among the most common ones such as Facebook Messenger, Google RBM, Slack, Telegram and others. Moreover, it supports generic chatbots, webchats, SMS and emails. Regarding the well-known supported platforms, for each of them, Botanalytics uses a specific data model. As stated in Section 1, the goal of this paper is to provide a generic, platform-independent data model that can fit with any service.

Therefore, the most interesting part offered by Botanalytics are the models used for generic chatbots and web chats. The two models are identical and consist of three fields. `is_sender_bot` is a boolean value that is true if the message is sent by the bot, false otherwise. `user` is an object to identify the user in the system while `message` is an object and can be used to store the content of the message. We believe that in such a model, there are some significant limitations. First of all, some basic concepts are missing. For example, there is no way to identify a conversation and it is impossible to understand whether a message adds values or a context to a message received before.

Moreover, there is no way to determine the order of the messages (e.g., there is not a timestamp or a field to understand which was the previous message). Last but not least, all the modern chatbots use ML and NLP to extract entities, intent, domain and sometimes also the language of a message. In the model proposed by Botanalytics, it is not possible to store such information.

*Botlytics*² partially solve the limits of Botanalytics. It allows to save data coming from any platform and it does not have tailored models for each of them. It only has a general representation for the

messages composed by six different fields. `text` is the content of a message and `kind` is similar to the `is_sender_bot` shown before. It can be filled with "incoming" or "outgoing" to understand whether the message is received or sent. Botlytics solves the major problem of the conversation by using the field `conversation_identifier`, a unique string that organizes messages into conversations. With this field, they partially solved the problem of the order of the messages. The field `sender_identifier` is used to identify the sender of the message while `platform` represents the name of the platform that the message was sent on. An interesting feature introduced by Botlytics is the design of the payload of a message. It is defined as the payload from complex messages that include than just text. In other terms, it is common nowadays that bots answer to a question with complex objects such images, carousels, maps or a set of buttons representing suggested actions. Therefore, it is important to model such responses. A critical point of Botlytics concerns the issue with the meta-information extracted with ML and NLP that cannot be modelled.

*Botmetrics*³ is based on six fields: `text`, `message_type`, `user_id`, `platform` similar to those we saw in the other platforms; it adds `create_at` and `metadata`. The former gives the possibility to represent the timestamp when the message was sent or received, while the latter can be used to store additional information. By using `metadata`, domain-specific information can be easily associated with a message. A `metadata` field is defined also in the model proposed in Section 3; in Section 4.1, we show how this field can be used to store such additional information. Unfortunately, neither information about ML and NLP nor data related to conversations are included in Botmetrics.

The data model proposed by *Dashbot*⁴ differs from the others. Apart from the `text` and the `user_id`, the other standard fields are missing (e.g., the id of the type of the message and the platform). It contains a field `intent` that partially solve the NLP and ML issue discussed above; unfortunately, fields for the domain, the entities and the language of the message are still missing. Optionally, the developer can store a list of `images` and a list of `buttons` where an image consists of a URL, and a button consists of an identifier, a label and a text. It is also possible to use `postback`: an object that contain the identifier of the button clicked. It is an interesting way of designing an eventual, complex message. As mentioned, it is common to have carousels as responses and the one

¹<https://botanalytics.co>

²<https://botlytics.co>

³<https://bot-metrics.com>

⁴<https://www.dashbot.io>

Table 1: Summary of the features supported by the platforms analyzed in Section 2

	Our solut.	Bot Analytics	Botlytics	Dashbot	Botmetrics	Chatbase
Message type	X	X	X		X	X
Platform	X		X		X	X
User identifier	X	X	X	X	X	X
Conversation identifier	X		X			X
Timestamp	X				X	X
Content - text	X	X	X	X	X	X
Other contents	X		Partially	Partially		
Domain	X					
Intent	X			X		X
Entities	X					
Language	X					
Bot Version	X					X
Domain Specific Data	X				X	
Platform Specific Data				X		
User Profile				X		
Message Handled						X

proposed in Dashbot can be a way for representing them. What is interesting is that there are two other fields, `platformJson` and `platformUserJson`, that can be used to store platform- and user-specific information. Regarding the model we propose, we decided to model users and messages as two completely separated entities. Thus, the user profile and user-related information should be saved and handled apart from the messages.

Another data model that we analyzed is proposed by *Chatbase*⁵. In their model, `type`, `user_id`, `time_stamp`, `platform`, `message`, `intent`, `not_handled`, `version`, `session_id` are defined. What is interesting is that they use both the timestamp and the identifier of the session for handling messages. Concerning the other services, they have a `not_handled` variable to understand whether the chatbot handled a message or not. What is interesting is that they only model the raw message of the body regardless of the type of the original message, and they partially collect the information coming from NLP and ML modules.

Other platforms and frameworks such as *LinTO AI*⁶ and *Xakit*⁷ but due to their complex pipeline, we found it difficult to map the in Table 1.

⁵<https://chatbase.com>

⁶<https://linto.ai>

⁷<https://xatkit.com>

3 PROPOSED SOLUTION

In this section, an overview of our proposed data models proposed is given. In particular, we introduce two different models: one for messages and one for logs. The general idea is to use the message data model to store information about the conversations between users and chatbots, while the log data model should be used to handle the interactions.

3.1 Messages

The data model proposed for the messages is summarized in Figure 1. It shows three different kinds of messages: *requests*, *responses* and *notifications*. All types of messages share the following attributes:

- `messageId`: a mandatory field that uniquely represents a message entity into the system. In other terms, it is not possible to have two messages with the same `messageId`.

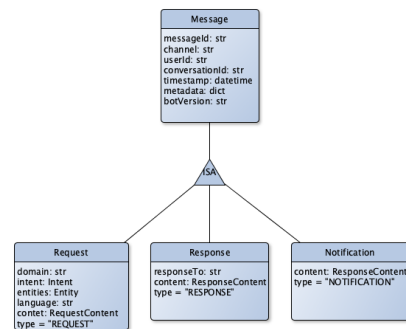


Figure 1: The fields of the three different types of messages

- `channel`: a mandatory string that represents the channel in which the conversation is taking place. For example, a channel can be 'MESSENGER' or 'TELEGRAM'.
- `userId`: a mandatory string that is used to identify a user. The user identifier can be handled at three different levels. The `userId` can be either provided by the platform (e.g., Messenger can provide the id of the user you are chatting with), or be generated locally. Note that if the `userId` is provided by the platform on which the chatbot is hosted, users can be tracked on multiple bots, a feature not available otherwise.
- `conversationId`: an optional field that can be used to identify a conversation. A conversation is given by a non-empty set of messages. Most of the platforms that host bots automatically provide an id for the conversation.
- `timestamp`: it represents the instance in which the message has been generated. Note that it can differ from the time when the message reach the logging system. In general, it is interesting to save the moment in which the interaction between the user and the bot happened. The ISO Standard 8601 is suggested here.
- `botVersion`: an optional field useful to distinguish different versions of the bot generating the message (e.g., in A/B testing).
- `metadata`: a field that store any extra information not modelled by the other fields. For instance, in the Memorable Experience project (Section 4), this field is used to associate with a message the id of the hotel involved in the conversation, the name of the hotel and the number of nights the user stayed in the hospitality structure.

All the other attributes and the content depends on the type of message. The value of the field `type` can be `REQUEST`, `RESPONSE` or `NOTIFICATION`.

3.1.1 Request Messages

A request message is a message that, in the interaction flow, goes from the user to the chatbot. In general, in a request message, a user may ask the chatbot to perform an action, can send a comment or start a conversation. The user may use different means to interact with a chatbot, such as natural language (using voice or text) or by using predefined actions (buttons). To model these needs, we designed the field `content` of a request message in two different ways: `text` contents are the messages that involve natural language while `action` content represents the interaction through but-

tons. The data models for text messages and action messages are shown in Figure 2.

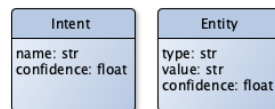


Figure 2: On the left, a representation for intents. On the right, a representation for an entity. In a single message, it is possible to have than one entity.

In modern chatbots, there is also the possibility to interact using non-verbal communications. We identified two main cases: sharing location data (an example is shown in Section 4) and sharing files. To fulfill these needs, we designed two additional types of content. Regarding the location, we simply designed a data model that support longitude and latitude; for the attachment, we defined a model that contains the URI of the file and an optional alternative text. The data models proposed can be seen in Figure 3.

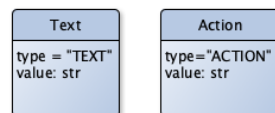


Figure 3: On the left, the characterization of a textual request. On the right, the description of an action-based request.

Our model fully supports the meta-information generated by NLP and ML modules. In fact, as shown in Figure 1, it is possible to associate a `domain`, an `intent` and a list of `entities` to each request message.

A *domain* can be defined as a way to organize intents and entities in groups. Usually, a domain is a general term representing the topic of a conversation (e.g., travel, work). It is common to have bots with a single domain, but it is possible to develop bots with multiple domains. Therefore, we decided to track the domain of the conversation and to represent it as a string. The usage of this field may become helpful also in terms of computing analytics. For instance, a person may be interested in analyzing only the conversations within a certain domain.

For similar reasons, we decided to model the *intent*, a string representing the goal that the user wants to achieve. It is specific than the domain, and is usually associated with a confidence score – a metric of accuracy that varies between 0 and 1, representing the quality of the detection of the intent.

Finally, the NLP and ML modules provide also information about the *language* of the messages and

a set of *entities*. An entity represents something the users are likely to talk about and which has meaning to the business. For example, if a user is looking for a flight, a meaningful part of the text is the destination. Once extracted, an entity is associated with a confidence score and with a type. The confidence works like the one introduced for the intent, while the type is meta-information associated with the part of the extracted text. Using the same example, suppose that Boston is detected as the destination. The string "Boston" may be associated with the type `geo_city` and, eventually, to a confidence score. From a single message, it is possible to extract than one entity. The data models for entities and intents are depicted in Figure 4.

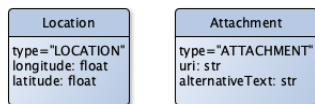


Figure 4: The characterization of two additional requests: location on the left and attachment on the right.

3.1.2 Response Messages and Notifications

Response messages represent the answer of the bot to the user. Thus, there is no need to take into account metadata generated by ML and NLP modules. An important field to add is `responseTo`. Many bots tend to send multiple messages to answer a specific question. An example can be seen in Figure 8 where once the user shared the location, the bot answered with three messages: two text messages and a carousel. All three messages are part of the same conversation, and each of them should be represented as a single message with a unique identifier. In this sense, having a `conversationId` and a `messageId` is not enough to catch the fact that the three messages are trying to answer to a single message. This is why we designed `responseTo`.

Another point that emerges from Figure 1 is that the types of content associated to response messages differ from the ones associated to requests. Figure 5 shows the possible types of contents for responses. While *text*, *attachment* and *location* responses work similarly to the ones of request messages, the are two main novelties: *multi-action* and *carousel* responses. Both of them can be seen in Figure 8. In particular, a multi-action content consists in a set of buttons that can be used to suggest actions to take to the users, while carousel responses are set of cards that are used to share a set of items with users.

A peculiarity of the buttons is that it is uncommon to have messages containing only buttons (multi-

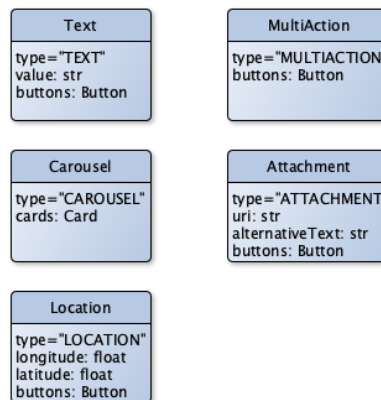


Figure 5: The models for the possible contents of response messages and notifications

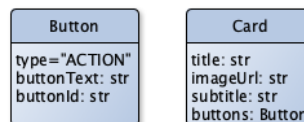


Figure 6: The description of buttons (on the left) and cards (on the right).

action responses). The designers tend to attach buttons to standard messages. For instance, the buttons showed in Figure 8 are not part of a text message; they are attached to a text message. To fulfil this need, we decided to add the field `buttons` to each possible response and each card. Cards and buttons are complex object that are represented in Figure 6.

A button is characterized by `buttonText` and `buttonId`. While the first one corresponds to the text that the users are going to find in the chat, the latter is used to identify the buttons in the system. For instance, in the example in Figure 8, the text of the button is "find a bike" while an example of identifier is "button_1". Cards are represented using `title`, `subtitle`, `imageUrl` and `buttons`. By mapping the features with the example of Figure 8, we find that the title is the highlighted line ("Tiistinkallio") while the subtitle is the other lines of text ("39 available bikes"). The buttons are "Open map" and "Add to favourite". Having an image on each card is highly suggested but not mandatory. Whenever an image is present, the field `imageUrl` can be used to link a resource. The only difference between a response message and a notification is that in the first one the communication flow starts from the user, while in the latter, the chatbot send the first message. For this reason, the only difference between them is the `responseTo` field that is not present in the notifications. Regarding the contents for a notification, the possibilities are the same

offered for response messages (Figure 5).

3.2 Logs

When we decided to model the interactions between bots and humans, we discovered that logging messages was not enough. In particular, by modelling the content of the message and the other attributes described in Section 3.1, the interaction between the user and the message is missing. To store actions such as click on buttons; scroll over a carousel and others, we propose a data model for logs. The model proposed is shown in Figure 7.

What follows is the detailed description of each field:

- `logId`: a string that uniquely represents the log inside the system. In other terms, it is not possible to have two logs with the same identifier.
- `component`: the name of the object that generated the log.
- `authority`: the namespace of the component that generated the log. A namespace can be defined as a set of symbols that are used to organize objects of various kinds, so that these objects may be referred to by name.
- `severity`: a field that takes values such as `ERROR`, `WARNING`, `INFO` and `DEBUG`.
- `logContent`: a string that represents the message of the log. This string can be arbitrarily long and should be significant enough to let people understand the situation and, possibly, the meaning of the log. An example of the possible malformed log is “ERR001” while an example of a simpler log is “ERR001: Wrong password.”;
- `timestamp`: the instant in which the message has been generated. Note that it may differ concerning the moment in which the message reach the logging system. In general, it is interesting to save the moment in which the interaction between the user and the bot happened. We suggest to use the standard ISO 8601;
- `botVersion`: it contains the version of the bot that generated the log;
- `metadata`: it is a field that represents any extra information not modelled by the other fields.

4 USE CASES

In this section of the paper, we show how different architecture can be mapped to the model proposed

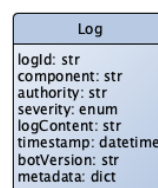


Figure 7: Data models of possible contents of a request message

4.1 Memorable Experience

The customization of the experience is fundamental in today’s world and is a crucial factor in becoming competitive on the market, even in the tourism sector. The idea of this project sponsored by FESR fundings is to create a chatbot providing a fully-customized experience for tourists.

This chatbot is hosted on Facebook Messenger, and the business logic has been developed from scratch from a private company. For this reason, it represents a challenge for the model proposed. Moreover, in this project emerged the need to associate some domain-specific information to each message such as the name of the hospitality structure and the length of the stay. What follows is an example of how they use our data model for storing messages.

```
{
  "messageId": "M31938129",
  "channel": "Facebook",
  "userId": "U545940",
  "conversationId": "C09532",
  "timestamp": "2019-10-10:12.45.23T",
  "botVersion": "1.0A",
  "metadata": {
    "hotelName": "Hotel Maria",
    "numberNights": 4
  },
  "domain": "general"
  "intent": {
    "name": "greeting",
    "confidence": 0.94
  },
  "content": {
    "type": "TEXT",
    "value": "Hi"
  }
  "type": "REQUEST"
}
```

Our model succeeds in fitting all the needs that emerged from this project. Moreover, our model successfully store data coming from a complex pipeline.

The companies involved in this project decided to use also the logs for modeling the interaction between the users and some items of the conversations. In particular, they decided to count the number of taps per

user per card. To succeed, they used the data model proposed in Section 3.2 as follow.

```

"logId" : "LOG312312313",
"project": "memex",
"component": "dim.memex",
"authority": "detection.click",
"severity": "INFO",
"logContent": "button pressed",
"timestamp": "2019-07-04T08:33:48T"
"botVersion": "1.0",
"metadata":{
  "buttonId": "BTN03021312",
  "messageId": "MSG0312312",
  "userId": "USR321321",
  "link": "http://w3u.it"
}

```

4.2 BlueAlpaca

It is a chatbot developed in the context of the SynchroCity project funded by the European Union. It uses our data model to store messages. Differently from Memorable Experience, it is hosted on Facebook Messenger, and the business logic is handled with wit.ai, a service by Facebook to create chatbots. We believe it is a relevant case study to check the validity of our data models for two main reasons. First of all, as shown in Figure 8, this bot uses a complex type of responses such as cards and buttons. Moreover, in their stack, there is a well-known and widely-used service to extract knowledge from the messages. Considering the carousel response, what follows is

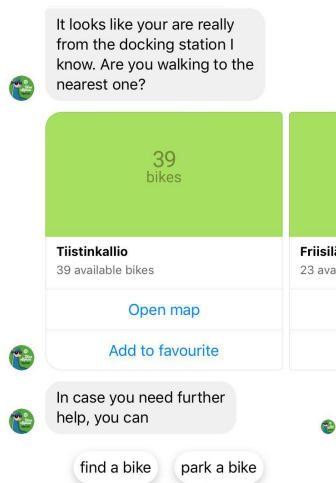


Figure 8: Data models of possible contents of a request message

the mapping with our data model of the content of the message.

```

"type": "CAROUSEL",
"cards": [
  {
    "title": "Tiistinkallio",
    "imageUrl": "images/image.png",
    "subtitle": "39 available bikes",
    "buttons": [...]
  }, ...
]

```

Concerning the other models, our one is the only one that fully supports complex types of responses. Moreover, with this project, we found out that our model can also be used if services such as wit.ai are used to compute ML and NLP metrics.

4.3 PathFinder

To further validate our model, we took into account PathFinder: an Action on Google service that accepts both text and voice-based requests. The bot is managed using DialogFlow, and its goal is to provide directions for any transports using multiple sources of data. For instance, PathFinder can provide directions for bicycle users in areas (e.g., Italy) where some map providers do not support such kind of directions. Our model was used to store messages in a structured and organized way, and it fulfils all the needs of the developers that successfully mapped the information coming from Google's DialogFlow with the model proposed.

5 CONCLUSIONS

After an analysis of other models, we described a generic and architecture-independent model that, as proven in Section 4, can be used with bots hosted on several platforms, based on a large range of services. We found that the most complete model is the one offered by Chatbase. However, we found some limitations. In particular, it partially covers the needs of NLP and ML modules. Chatbase allows to represent the intent, but entities, domains and language should also be considered. Another big limitation of its model is that the only content type supported is text. Whenever a message contains more than just text, it must be converted into a string and only after this processing step, the message can be stored. By looking at the case studies analyzed in Section 4, we decided that modelling the content as text is not enough. Our data model is the only one that supports a variety of different content types and only Botlytics and Dashbot partially support other types of messages. In particular, on Botlytics, a field called payload can be used to represents all the content of

a message which is not text. Concerning Dashbot, it gives the possibility to store a text, a list of images and a list of buttons. In our opinion, this model presents some limitations. First of all, it is not possible to represent complex responses such as location messages. Moreover, the meta-information representing the type of message is missing. In other terms, if a message contains an image and a button, it is not possible to establish whether it is a simple image or a card of a carousel. Thus, we believe that having a unified model for a type of message such as a card can improve the quality of the data stored. Finally, having a unified data model means that data from different platforms can be saved in the same way, and thus, data can be analyzed more efficiently. As described in Section 1, it is and common for companies to publish their bot and offer their services on more than one platform. Thus, we believe it is fundamental to have a unique representation of data.

Another limit that we found is the possibility of associate domain-specific information to the messages. For instance, we showed that by using `metadata`, in Memorable Experience, the developers could associate the name of the hospitality structure to the message. As showed in Table 1, apart from Botmetrics this kind of information cannot be stored with the analyzed models. A similar approach is provided by Dashbot. They allow to store platform and user-related data in two specific fields. We decided to do not store user data in the messages because we believe that the profile of a user should be managed with a completely different data model and we think that platform-specific information should be modelled using the `metadata` field.

With the introduction of this model for representing interactions between human and chatbots, we hope to contribute to the analysis and the management of data coming from bots hosted on different platforms and based on different services.

ACKNOWLEDGEMENTS

The project is carried out within the ERDF Regional Operational Programme 2014-2020 of the Autonomous Province of Trento, and it is co-financed by the European Union through the European Fund for Regional Development, for the Italian state and the Province of Trento.

REFERENCES

- Brixey, J., Hoegen, R., Lan, W., Rusow, J., Singla, K., Yin, X., Artstein, R., and Leuski, A. (2017). SHIHbot: A Facebook chatbot for sexual health information on HIV/AIDS. In *Proceedings of the 18th Annual SIG-Dial Meeting on Discourse and Dialogue*, pages 370–373, Saarbrücken, Germany. Association for Computational Linguistics.
- Chakrabarti, C. and Luger, G. F. (2015). Artificial conversations for customer service chatter bots: Architecture, algorithms, and evaluation metrics. *Expert Systems with Applications*, 42(20):6878 – 6897.
- Diederich, S., Brendel, A. B., and Kolbe, L. M. (2019). Towards a taxonomy of platforms for conversational agent design. In *In Proc. of the 14th Int. Tagung Wirtschaftsinformatik - Human Practice. Digital Ecologies. Our Future.*, pages 1100–1114.
- Fadhil, A. and Gabrielli, S. (2017). Addressing challenges in promoting healthy lifestyles: The al-chatbot approach. In *Proceedings of the 11th EAI International Conference on Pervasive Computing Technologies for Healthcare, PervasiveHealth '17*, pages 261–265, New York, NY, USA. ACM.
- Future, M. R. (2019). Chatbots market research report – global forecast 2023.
- Gnewuch, U., Morana, S., and Maedche, A. (2017). Towards designing cooperative and social conversational agents for customer service. In *Proceedings of the 38th International Conference on Information Systems (ICIS), Seoul, ROK, December 10-13, 2017. Research-in-Progress Papers*. AISeL, Seoul, ROK.
- Heller, B., Proctor, M., Mah, D., Jewell, L., and Cheung, B. (2005). Freudbot: An investigation of chatbot technology in distance education. In Kommers, P. and Richards, G., editors, *Proceedings of EdMedia + Innovate Learning 2005*, pages 3913–3918, Montreal, Canada. Association for the Advancement of Computing in Education (AACE).
- Kerlyl, A., Hall, P., and Bull, S. (2007). Bringing chatbots into education: Towards natural language negotiation of open learner models. In Ellis, R., Allen, T., and Tuson, A., editors, *Applications and Innovations in Intelligent Systems XIV*, pages 179–192, London. Springer London.
- Kethuneni, S., August, S. E., and Vales, J. I. (2009). Personal health care assistant/companion in virtual world. In *Proc. of the 2009 AAAI Fall Symposium*. AAAI.
- Nica, I., Tazl, O. A., and Wotawa, F. (2018). Chatbot-based tourist recommendations using model-based reasoning. In *ConfWS*.
- Verhagen, T., van Nes, J., Feldberg, F., and van Dolen, W. (2014). Virtual Customer Service Agents: Using Social Presence and Personalization to Shape Online Service Encounters*. *Journal of Computer-Mediated Communication*, 19(3):529–545.
- Weizenbaum, J. (1966). Eliza;a computer program for the study of natural language communication between man and machine. *Commun. ACM*, 9(1):36–45.