Conversational Data Exploration

Nicola Castaldo, Florian Daniel[0000−0003−3004−8702], Maristella Matera[0000−0003−0552−8624], and Vittorio Zaccaria[0000−0001−5685−9795]

Politecnico di Milano - Dipartimento di Elettronica Informazione e Bioingegneria
P.zza Leonardo da Vinci, 32 - 20133 Milano - Italy
nicola.castaldo@mail.polimi.it,
{florian.daniel,maristella.matera,vittorio.zaccaria}@polimi.it

Abstract This paper presents a framework for the design of *chatbots for data exploration*. With respect to conversational virtual assistants (such as Amazon Alexa or Apple Siri), this class of chatbots exploits structured input to retrieve data from known data sources. The approach is based on a conceptual representation of the available data sources, and on a set of modeling abstractions that allow designers to characterize the role that key data elements play in the user requests to be handled. Starting from the resulting specifications, the framework then generates a conversation for exploring the content exposed by the considered data sources.

Keywords: Chatbots for Data Exploration · Chatbot Design · Conversational UIs.

1 Introduction

Chatbots are growing fast in number and pervade in a broad range of activities. Their natural language paradigm simplifies the interaction with applications to the point that experts consider chatbots one of the most promising technologies to transform instant messaging systems into software delivery platforms [2,3]. The major messaging platforms have thus opened their APIs to third-party developers, to expose high-level services (e.g., messaging, payments, bot directory) and User Interface (UI) elements (e.g., buttons and icons), to facilitate and promote the development of innovative services based on conversational UIs [4].

Despite the huge emphasis on these new applications, it is still not clear what implications their rapid uptake will have on the design of interactive systems for data access and exploration. The applications proposed so far mainly support the retrieval of very specific data from online services. It is still unclear how this paradigm can be applied for the interaction with large bodies of information and machine agents. A critical challenge lies in understanding how to make the development of bots for data exploration scalable and sustainable, in terms of both software infrastructures and design models and methodologies [1,5].

In this paper, we present a *data-driven design paradigm* for building conversational interfaces for data exploration. In line with traditional Web Engineering methodologies, we exploit properties of data models and propose schema annotations to enable the generation of conversation paths for the exploration of a

database content. After clarifying the main requirements characterizing chatbots for data exploration, we introduce *i)* a set of *conversational annotations* that characterize the role that data elements play in the exploration of a database content, and *ii)* a design process and an enabling architecture that exploit the conversational annotations to generate a dialogue for exploring the database. We then conclude the paper and outline our future work.

2 Chatbots for Data Exploration

With respect to conversational virtual assistants (e.g., Amazon Alexa or Apple Siri), *chatbots for data exploration* use conversations to let the users move across different data items within a body of organized content. In order to exemplify the main characteristics of this class of chatbots, let us consider a simple yet expressive conversation for exploring an example database storing data on orders and customers (see Figure 1). The respective database schema is reported in Figure 2.

Current approaches would already support the development of such a chatbot. However, some of them would require using conventional programming languages to write rule-based scripts for handling the dialog. For example, Microsoft's Bot Framework[1] requires using the .NET family of languages, while IBM's Watson Conversation Service comes with SDKs for Node, Java, Python, .NET. In order to simplify the development, some *dedicated platforms* (e.g., Dialogflow[2] and Motion AI[3]) offer visual, flowchart-based development paradigms also equipped with advanced AI and NLP support that only needs to be properly configured. Other frameworks propose pro-

Chatbot **User**

what do you have to show me?

Hi there explorator!
Here is a few examples of what you can ask me:
- Find **orders** that contained product *Fiat Punto*
- Find **customers** that are located in *Milan*
- Find **payments** done by *American Souvenirs Co.*

find **customers** with contact *John*

Let me check...
Et voila! I found 8 values:
[*Car Models - B.Goode John*]
[*JJ Store - Browns John*]
[*American Souvenirs Co. - Wayne John*]
[SHOW MORE]
Select the one you are interested in or click the last button to see more.

* user clicks on [*Car Models - B.Goode John*] *

Here is what I know about this **customer**:
- *customerName: Car Models*
- *contactLastName: B. Goode*
- *contactFirstName: John*
...

If you want more information, I can tell you:
- [*orders made*]
- [*payments made*]

show me the past selections, please

Here you can see the history of the conversation! You can resume the context of this conversation to a point in the past by clicking on the corresponding button.
- [FORGET ALL]
- [element of type **customer** with contact *John*]
(currently) selected **customer**: *Car models - B.Goode John*

Figure 1. Exemplary conversational data exploration by a user assisted by the envisioned data exploration bot.

[1] `https://docs.botframework.com/en-us/`
[2] `https://dialogflow.com/`
[3] `https://www.motion.ai/`

prietary, textual languages for the design of conversation paths. For example, PandoraBots[4] proposes AIML (Artificial Intelligent MarkUp Language), an XML-based language for tagging conversation elements, such as parameters in the user requests, query patterns and answer templates. However, all these approaches still require the explicit configuration of predetermined utterances, intent interpretation and actionable answers to be assembled in the conversation. In contrast, our work has the ultimate goal of generating conversational paths for data exploration starting from the schema of the data source and taking advantages of data properties that can be derived from the schema itself.

2.1 Requirements

To achieve our goal, there are a number of requirements to be fulfilled by the environment for chatbot generation and execution, for example, just to mention the most relevant ones, the capability to:

1. Connect to the database to obtain the schema;
2. Support generic data exploration vocabulary and actions;
3. Extract/learn database-specific vocabulary from schema and instances;
4. Extract/learn database-specific actions from the schema, for example navigating relationships can be seen as data access actions;
5. Allow the user to manipulate query results, e.g., further filtering results based on some attributes.

Moreover, the chatbot has to dynamically identify intents and entities from the user utterance, translate the interpreted elements into queries and connect to the database for query execution. Other important aspects then refer to proper visualizations of query results and the management of context memory.

The approach that we present in this paper goes exactly into this direction. In the following sections we will discuss some preliminary results in the definition of a design methodology that supports the generation of a conversation starting from a database schema. It is possible to think of scenarios where the conversation could come out directly from the analysis of properties of the database schema. However, in order to achieve conversations that can be effective for the final users (e.g., free of ambiguities in the possible suggested paths and able to use an adequate vocabulary), we will show how a *designer*, who knows how the database is organized, can annotate the schema to model how some data elements can guide the definition of the conversation.

3 Model

To translate an input phrase into a specific query on the database, we propose the definition of a mapping between what can be understood by the chatbot (intents and entities) and the elements of the database (relations, attributes,

[4] `https://www.pandorabots.com/`

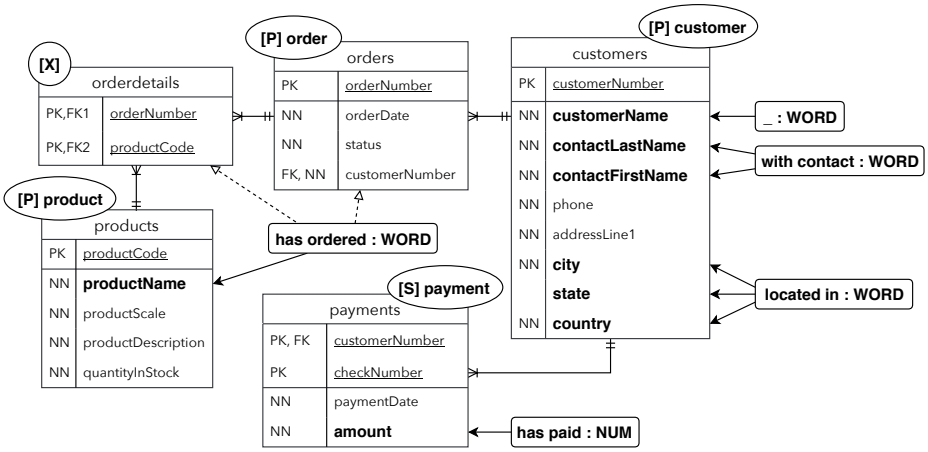

Figure 2. Database schema and annotations for *table types* and *conversational attributes*.

relationships). This mapping is defined by a *designer*, who knows the structure of the database and is in charge of modeling the dialogue flow for the interaction with the database by the final user. In the following we will illustrate the main ingredients of our conceptual modeling approach; to make the description easier to follow, we will refer to an example database shown in Figure 2.

Conversational Entities based on Table Types. The first step is to characterize the database tables as *Primary* (tag `[P]`), *Secondary* (tag `[S]`), or *Crossable Relationship* (tag `[X]`), to express the role that data entities play in the exploration of data. The distinction between the first two types is that while values belonging to a Primary table may represent useful information without the need of a context, Secondary tables contain data strongly related to other entities, which become clear only when that specific relationship is considered. For example, tagging as Secondary the table `payments` means that the access to its values depends on another table, `customers`. In other words, with this characterization the designer defines that instances of `payment` can be retrieved only by passing first through instances of `customers` – for example because payments data would not be meaningful otherwise. Labeling tables as Primary, in contrast, relaxes this dependence constraint and enables direct queries on tables. Finally, Crossable Relationship characterization is dedicated to bridge tables, i.e., the ones that represent many-to-many relationships between entities. Their rows may have a meaning only when a join operation across them is executed; thus no direct or deferred search is allowed on them.

Some tables might have names not suitable in the context of a conversation. Thus, as reported in Figure 2, each primary and secondary table has to be labeled with a keyword, i.e., the name that will represent it during the conversation. For example, in the request "find **customer** with contact *John*", the chatbot

understands that **customer** is the placeholder for the table `customers`. Multiple words can be associated to a table.

Conversational Attributes. Defining the role of tables and their representative names enables the system to translate phrases like "find **customers**", which aim to select all the instances of the corresponding table. Other elements are needed to interpret conditions for filtering subsets of instances, for example queries like "find **customers** that are located in *Milan*". In order to support this granularity in the results, the designer can define a set of *conversational attributes* for each data element, as shown in Figure 2. This annotation procedure considers what are the attributes that will be used to filter the results and how the final user will refer to them, i.e., through which expressions. For example, in Figure 2, the attributes `city`, `state` and `country` are labeled with the expression **located in**: this gives the chatbot the capability to understand the input phrase "find **customer** located in *Milan*", and process it as a query that selects the customers having `city = Milan`, or `state = Milan`, or `country = Milan`. These conversational attributes may belong to other tables, rather than the one directly addressed by the user request. This is for example the case of the conversational attribute `has ordered`: even if it is defined for the field `productName` of table `products`, it can be used also in queries like "find **customer** that has ordered *Fiat Punto*". The table `customers` is indeed reachable through the tables `orders` and `orderdetails`. This is graphically represented in Figure 2 by means of dotted arrows.

In some cases the user may also ask: "find **customer** *Car Models*", without specifying in the utterance any conversational attribute, e.g., without specifying that `Car Models` in the previous query is a value for the customer name. The system will anyway search for related instances in the table `customer` by the attribute `customerName` that, as represented in Figure 2, is tagged as a conversational attribute for the table. However, as indicated by the graphic symbol "_", the user does not need to specify any additional expression.

In order to help the chatbot process and interpret correctly the utterance, for each conversational attribute it is important to specify its *conversational type*:

- `WORD`: any information without a particular structure or syntax;
- `NUM`: numerical values;
- `DATE`: datetime/date values;
- `Custom ENUM`: entities that can assume enumerable values.

The last one can be useful when an attribute can assume only a fixed set of values. An example could be the attribute `businessName` for the table `customers` with values `Public Administration` and `Private Company`.

Conversational Relationships. In the conversation example in Figure 1, after the user has successfully selected a customer, its data and a few buttons are displayed. These last represent the relationships that can be navigated starting from the instance being examined. These relationships have to be specified in the

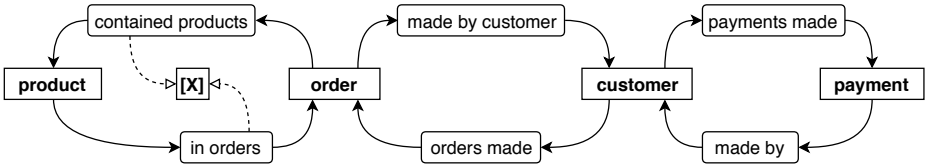

Figure 3. Annotations to specify conversational relationships.

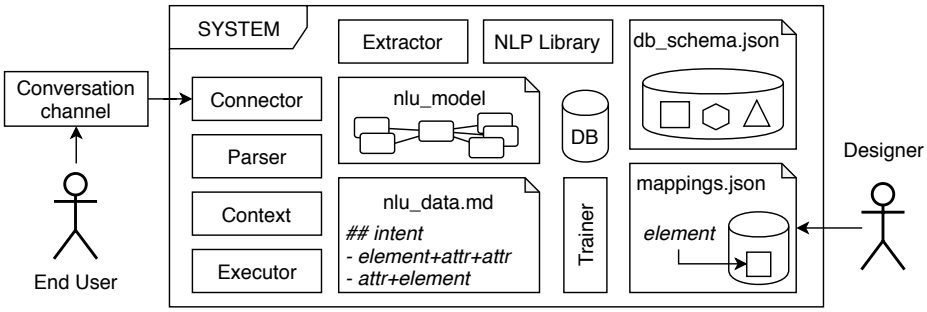

Figure 4. Functional architecture for conversation generation and execution.

annotated database schema (see labeled arcs in Figure 3) so that user utterances can also imply navigation in the database through join operations. The arcs in Figure 3 are not bidirectional, since during conversation the user may refer differently to relationships depending on the direction in which they are traversed. Note that the relation between `product` and `order` needs to be supported by a crossable relationship table, being it a many-to-many relationship.

4 Design Process and System Architecture

The chatbot design process consists of both automatic and manual phases:

1. *Parsing of the database schema* (automatic): this is needed to interpret the database schema and generate a simplified version enabling the annotation activity;
2. *Schema annotation* (manual): the designer produces a new schema with all the annotations needed to generate the conversation;
3. *Generation of the training phrases* (automatic): this consists of the training of the Natural Language Understanding (NLU) model.

Figure 4 shows the organization of the current system prototype that supports the previous phases. In this version the system works with relational databases and SQL schemas; we are however working to generalize the approach to other data models.

The first artifact is the JSON document, `db_schema.json`, that the system generates automatically by means of the *Parser* module. This is a Python script

that extracts from a SQL schema the main elements of the database and generates a simplified version of the schema in JSON format. The designer thus defines the conversational model through the annotations described in the previous section; the result of this activity is the JSON document `mappings.json`.

The automatic generation of the training phrases and the training of the model then take advantage of a *Natural Language Processing (NLP) library*, that in the current prototype is RASA NLU[5]. This component interprets sentences by performing intent classification and entity extraction. For example, in the utterance "find **customer** with contact *John*", the intent is *"to find an instance of an entity"* , and the identified conversation entities relate to the table word **customer**, the conversational attribute with contact, and the keyword *John*. This interpretation is possible because the system has previously generated the document `nlu_data.md`, that is a MarkDown file containing a list of sentences for each intent, in which the entities are labeled accordingly. This generation step is followed by a training where the `nlu_data.md` document is used by the NLP Library to create the `nlu_model` enabling intent classification and entity extraction. Both these steps are performed by the Python module *Trainer*.

Once the model is defined, the communication with the user can take place. The module *Connector* is responsible for receiving utterances and delivering answers through different channels[6]. At runtime, the *Extractor* module uses the NLP Library and the model previously created to classify the correct intents and extract the entities. The *Executor* module then executes the resulting queries on the database content. This last component represents the core of the chatbot, since it manages the communication with the database and with the *Context* module and, based on the results of a query or on the current status of the conversation, identifies what and how to answer to the user.

To track the status of the conversation, the *Context* module registers every action performed during conversation, as well as the instances retrieved from the database to compose the answers. For sake of brevity, and because the focus of this paper is on modeling primitives for conversational elements, we will not describe here the logics underlying this module. It is however worth mentioning that managing the context is not just maintaining the list of the previous utterances. Other aspects need to be considered, for example whether the entity of the current utterance has been already shown in previous requests and answers and so these old messages can be now discarded.

5 Conclusion

This paper has illustrated some preliminary results in the definition of a framework for the development of chatbots for data exploration. As observed in [5], conversation-based interaction may simplistically lead to believe that chatbot development mainly implies writing rule-based scripts governing the dialogue. At the other extreme, NLP- and AI-sophisticated techniques may appear as the

[5] `https://rasa.com/docs/nlu/`

[6] The current prototype implementation exploits Telegram.

only viable solutions. While these approaches may be valid in several scenarios, our position is that model-based methodologies, which are typical of Web Engineering, could offer several advantages to the chatbot world, especially when conversational interfaces are meant to support the access to extensive collections of data. With this paper we aim to give a first contribution in this direction. In particular, we highlight that chatbot development may require focusing on data modeling concerns that can guide the generation of conversations for data exploration. Some aspects, however, still remain open and our future work will focus on them.

A challenge is understanding how the conversational paradigm can integrate – or even replace – the traditional, visual paradigm for data exploration. This is not only a problem of data presentation, but also of adequate data analysis techniques, such as summarization, interactive drill-down and roll-up data exploration capabilities or the automatic generation of graphical charts. Our future work will study how these aspects impact the different layers (data, application logic, presentation). It will also try to understand whether the current approach can be adapted on top of different models (e.g., graph data models), also covering the integration of distributed data sources. Since we expect that the effort needed to manually define annotations would grow with very complex databased schemas, we will devise mechanisms to automatically identify notable elements in a database schema, and recommend their annotation within a usable visual environment.

Assessing the validity of the approach with respect to the experience of the final users is another challenge. We already planned some user studies with the aim of evaluating to what extent the generated conversations support user needs in data exploration. Gathering data about the user performance and satisfaction will help us verify the expressiveness of the modeling approach with respect to the elements to be provided for the conversations to be effective. Thus the user studies will have a double value: to improve the user experience of the generated applications as well as to validate the effectiveness of the modeling method.

References

1. Akcora, D.E., Belli, A., et al.: Conversational support for education. In: Proc. of Artificial Intelligence in Education, AIED 2018, London, UK, June 27-30, 2018, Part II. LNCS, vol. 10948, pp. 14–19. Springer (2018).
2. Grech, M.: The current state of chatbots in 2017. GetVoIP.com (April 2017), `https://getvoip.com/blog/2017/04/21/the-current-state-of-chatbots-in-2017/`
3. Inbenta Technologies Inc.: The ultimate guide to chatbots for businesses. Tech. rep., www.inbenta.com (2016)
4. Klopfenstein, L.C., Delpriori, S., Malatini, S., Bogliolo, A.: The rise of bots: A survey of conversational interfaces, patterns, and paradigms. In: Proc. of the 2017 Conference on Designing Interactive Systems, DIS '17, Edinburgh, United Kingdom, June 10-14, 2017. pp. 555–565. ACM (2017).
5. Pereira, J., Díaz, O.: Chatbot dimensions that matter: Lessons from the trenches. In: Proc. of Int. Conf. on Web Engineering, ICWE 2018, Cáceres, Spain, June 5-8, 2018. LNCS, vol. 10845, pp. 129–135. Springer (2018).