A Model-based Chatbot Generation Approach to Converse with Open Data Sources*

Hamza Ed-douibi[1][0000−0003−4342−4818], Javier Luis Cánovas Izquierdo[1][0000−0002−2326−1700], Gwendal Daniel[1][0000−0003−0692−0628], and Jordi Cabot[1,2][0000−0003−2418−2489]

[1] UOC. Barcelona, Spain
{hed-douibi,jcanovasi,gdaniel}@uoc.edu
[2] ICREA. Barcelona, Spain
jordi.cabot@icrea.cat

Abstract. The Open Data movement promotes the free distribution of data. More and more companies and governmental organizations are making their data available online following the Open Data philosophy, resulting in a growing market of technologies and services to help publish and consume data. One of the emergent ways to publish such data is via Web APIs, which offer a powerful means to reuse this data and integrate it with other services. Socrata, CKAN or OData are examples of popular specifications for publishing data via Web APIs. Nevertheless, querying and integrating these Web APIs is time-consuming and requires technical skills that limit the benefits of Open Data movement for the regular citizen. In other contexts, chatbot applications are being increasingly adopted as a direct communication channel between companies and end-users. We believe the same could be true for Open Data as a way to bridge the gap between citizens and Open Data sources. This paper describes an approach to automatically derive full-fledged chatbots from API-based Open Data sources. Our process relies on a model-based intermediate representation (via UML class diagrams and profiles) to facilitate the customization of the chatbot to be generated.

Keywords: Open Data · UML · Chatbots · API · OpenAPI.

1 Introduction

Open Data has emerged as a movement that promotes the free distribution of data for everyone to consume and republish. Governmental organizations are one of the significant sources of Open Data resources. They make their data publicly available online to provide more transparency and enable the general public to monitor and control the action of government bodies. For instance, the Spanish Open Data portalregisters more than 20,000 resources while the European portal, which harvests the metadata of Public Sector Information available on public data portals across European countries, links to over 1 million already.

* Work supported by the Spanish government (TIN2016-75944-R project)

On the one hand, Open Data promotes public awareness and aims at boosting citizen participation but, still, regular citizens hardly benefit from them as consuming Open Data requires non-trivial technical skills. Indeed, more and more Open Data sources are released as "web-friendly" artifacts (e.g., Linked-Data, APIs or NoSQL databases) that facilitate their consumption by external software applications and not directly by end-users. In particular, some specific technologies to support the publication of Open Data in the Web have been widely adopted in the last years, namely: SOCRATA[3], CKAN[4] and ODATA[5]. Other organizations also rely on OPENAPI[6], an initiative to formally describe general-purpose REST APIs, to document their Open Data APIs. While all these Web APIs "standards" offer a powerful means for writing complex data queries, they require advanced technical knowledge that hampers their actual use by non-technical people.

On the other hand, chatbots are intelligent conversational agents typically embedded in websites and instant messaging platforms. Users can ask questions or send requests to the chatbot using natural language, with no need to learn any technical knowledge/language. Chatbots have proven useful in various contexts to automate tasks and improve the user experience, such as automated customer services [22], education [13], e-commerce and, basically, every single domain involving any type of user interaction, including technical domains such as database queries[1]. Thus, we believe chatbots are the ideal interface to access and query Open Data sources, thus allowing citizens to access the government/company data they need directly. Citizens would ask the questions in their own language, and the chatbot would be the one in charge of translating that question into the corresponding API request/s.

In this sense, we propose a model-based approach to generate chatbots tailored to the Open Data API technologies mentioned above. As input of our process, an API definition is analyzed and imported as a UML schema annotated with UML profiles, which address specific domain information for chatbot configuration and Web API query generation. This API model is then used to generate the corresponding chatbot to access and query the Open Data source. Via the chatbot, users can ask direct queries or follow one of the guided query paths that facilitate the Open Data exploration. To validate our approach, we provide a proof-of-concept Eclipse plugin that fully supports SOCRATA and allows the integration of other Open Data specifications (i.e., ODATA, CKAN) as well as generic Web APIs (via OPENAPI specification).

Note that we focus on chatbots to help citizens exploit and dialogue with the Open Data resource they are interested in, not on chatbots aimed to help citizens find the best candidate/s data source/s based on their search interests [14], which is also useful but can be easily replaced with a proper keyword-based search interface. This is not the case for the approach we propose here as, even when

[3] `https://dev.socrata.com/`
[4] `https://ckan.org/`
[5] `https://www.odata.org/`
[6] `https://www.openapis.org/`

citizens know which data sources to query, they typically lack the technical skills to do it on their own and therefore will benefit from our chatbot to act as an "interpreter" between them and the underlying API technology.

The rest of the paper is organized as follows. Section 2 introduces the background of our work. Section 3 briefly describes our approach while sections 4 and 5 describe its main phases, namely, Open Data Import and Bot Generation, respectively. Section 6 described the tool support and Section 7 presents the related work. Finally, Section 8 ends the paper and presents the future work.

2 Background

2.1 Open Data

The Open Data movement aims to make data free to use, reuse, and redistribute by anyone. In the last years, Open Data portals have evolved from offering data in text formats only (e.g., CSV, XML) towards web-based formats, such as LinkedData [2] and Web APIs, that facilitate the reuse and integration of Open Data sources by external Web applications. In this subsection, we briefly describe the most common Web API technologies for Open Data, based on their popularity in governmental Open Data portals.

Socrata. Promoted by Tyler Technologies, the Socrata data platform provides an integrated solution to create and publish Open Data catalogs. Socrata supports predefined web-based visualizations of the data, the exporting of datasets in text formats and data queries via its own API that provides rich query functionalities through a SQL-like language called SoQL. Socrata has been adopted by several governments around the world (e.g., Chicago[7] or Catalonia[8]).

CKAN. Created by the Open Knowledge Foundation, CKAN is an Open Source solution for creating Open Data portals and publishing datasets in them. As an example, the European Data Portal relies on CKAN. Similar to Socrata, CKAN allows viewing the data in Web pages, downloading it, and querying it using a Web API. The CKAN DataStore API can be used for reading, searching, and filtering data in a classical Web style using query parameters or by writing SQL statements directly in the URL.

OData. Initially created by Microsoft, OData is a protocol for creating data-oriented REST APIs with query and update capabilities. OData is now also an OASIS standard. It is especially adapted to expose and access information from a variety of data sources such as relational databases, file systems, and content management systems. OData allows creating resources that are defined according to a data model and can be queried by Web clients using a URL-based query language in a SQL-like style. Many service providers adopted and integrated OData in their solutions (e.g., SAP or IBM WebSphere).

[7] `https://data.cityofchicago.org`
[8] `http://governobert.gencat.cat/en/dades_obertes/index.html`

OpenAPI. Evolving from Swagger, the OpenAPI specification has become the *de facto* standard to describe REST APIs. Though not specific for Open Data, OpenAPI is commonly used to specify all kinds of Web APIs, including Open Data ones (e.g., Deutsche Bahn[9]).

In our approach, we target Open Data Web APIs described by any of the previous solutions. We rely on model-driven techniques to cope with the variety of data schema and operation representations, as described in the next sections.

2.2 Chatbots

Chatbots are conversational interfaces able to employ Natural Language Processing (NLP) techniques to "understand" user requests and reply accordingly, either by providing a textual answer and/or executing additional external/internal services as part of the fulfillment of the request.

NLP covers a broad range of techniques that may combine parsing, pattern matching strategies and/or Machine Learning (ML) to represent the chatbot knowledge base. The latter is the dominant one at the moment thanks to the popularization of libraries and Cloud-based services like DialogFlow or IBM Watson Assistant, which rely on neural networks to match user intents.

However, chatbot applications are much more than raw language processing components [17]. Indeed, the conversational component of the application is usually the front-end of a larger system that involves data storage and service integration and execution as part of the chatbot reaction to the user intent. Thus, we define a chatbot as an application embedding a *recognition engine* to extract *intentions* from user inputs, and an *execution component* performing complex event processing represented as a set of *actions*.

Intentions are named entities that can be matched by the recognition engine. They are defined through a set of *training sentences*, which are input examples used by the recognition engine's ML/NLP framework to derive a number of potential ways the user could use to express the intention. Matched intentions usually carry *contextual information* computed by additional extraction rules (e.g., a typed attribute such as a city name, a date, etc.) available to the underlying application. In our approach, *Actions* are used to represent simple responses such as sending a message back to the user; as well as advanced features required by complex chatbots, like database querying or external service calling (e.g., API queries in this paper). As we will see, in this paper these actions will involve querying the API in charge of providing the Open Data information requested by the user. Finally, we define a *conversation path* as a particular sequence of received user *intentions* and associated *actions* (including non-messaging actions) that can be executed by the chatbot application.

[9] `https://developer.deutschebahn.com/store`

3 Overview

In this section, we present an overview of our proposal, depicted in Figure 1. Our proposal is split into two main phases, *Open Data Import* and *Bot Generation*.

During the import phase, an Open Data API model is injected (see OpenData *injector*) and refined (see *Model refinement*). The injector supports several input formats (i.e., Socrata, CKAN, OData and OpenAPI) and the result is a unified model representation of the API information (i.e., operations, parameters and data schemas).

Without loss of generality, this inferred API model is expressed as a UML class diagram to represent the API information plus two additional UML profiles. The first one, the Open Data profile, is used to keep track of technical information on the input source (e.g., to be used later on by the Bot to know which API endpoint to call and how). The second one is the Bot profile, proposed to annotate the model with bot-specific configuration options (e.g., synonyms or visibility filters) allowing for a more flexible chatbot generation. Once the injector finishes, the *Bot Designer* refines the obtained model using this second profile. During this step, elements of the API can be hidden, their type can be tuned, or synonyms can be provided (so that the chatbot knows better how to match requests to data elements).

The generation phase is in charge of creating the chatbot definition (see *Bot Generation*). This phase involves specifying both the bot intentions and its response actions. In our scenario, responses involve calling the right Open Data API operation/s, processing the answer, and presenting it back to the user.

As bot platform we use Xatkit [7], a flexible multi-platform (chat)bot development framework, though our proposal is generic enough to be adapted to work with other available chatbot frameworks. Xatkit comprises two main Domain-Specific Languages (DSLs) to define bots: Intent DSL, which defines the user inputs through training sentences, and context parameter extraction rule (see *Intents*); and Execution DSL, in charge of expressing how the bot should respond to the matched intents (see *Execution*). If preferred, Xatkit can also work with an internal Java-based DSL, that has the same semantics of the two (external) DSLs mentioned before but offering an alternative syntax (based on a chatbot fluent API), easier to adopt for programmers with Java knowledge.

Xatkit comes with a runtime to interpret and execute the bots' definitions. The execution engine includes several connectors to interact with external platforms (e.g., Slack or Github). In the context of this work, we implemented a new runtime in Xatkit to enable the communication with Web APIs.

The next sections describe each of these components in more detail. We will use the following running example to illustrate them. The example is based on an API provided by the Transparency Portal of Catalonia. In particular, the API that gives access to pollution data gathered by the surveillance network deployed within Catalonia. The data registers the air quality in Catalonia from 1991 until now, and it is updated daily. Besides the concentration of pollutants in the air, it is also possible to query the location and type of the measurement stations.

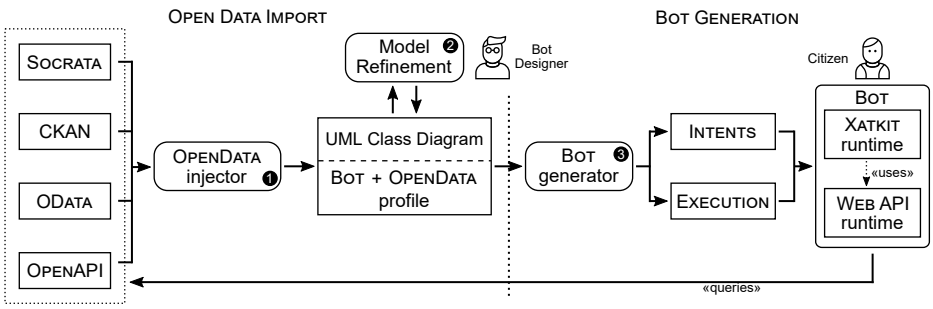


Fig. 1: Overview of our approach.

4 Importing Open Data APIs as Models

The import phase starts by analyzing the Open Data API description to inject a UML model representing its concepts, properties, and operations. This model is later refined by the bot designer. Next sections describe the main elements of this process. We will introduce first the modeling support required to represent Open Data APIs, then the injection step and finally the main tasks to tackle in the refinement step.

4.1 Modeling Open Data APIs

To model Open Data APIs, we propose employing UML class diagrams plus two UML profiles required to optimize and customize the bot generation.

Core Open Data representation as a UML Class Diagram Concepts, properties and operations of Open Data APIs are represented using standard elements of UML structural models (classes, properties and operations, respectively). Figure 2 shows an excerpt of the UML model for the running example[10]. As can be seen, the model includes the core concept of the API, called *AirQualityData*; plus two more classes to represent data structures (i.e., *Address* and *Location*). Note that the some elements include stereotypes that we will present later.

It is worth noting that most Open Data APIs focus around a single core data element composed of a rich set of properties which can be split (i.e., "normalized") into separate UML classes following good design practices, also facilitating the understanding of the model. This is what we have done for the UML diagram shown in Figure 2.

The Bot profile To be able to generate more complete bots, in particular, to expand on aspects important for the quality of the conversation, the Bot profile adds a set of stereotypes for UML model elements that cover (1) what

[10] Full model available at `http://hdl.handle.net/20.500.12004/1/C/ICWE/2021/232`

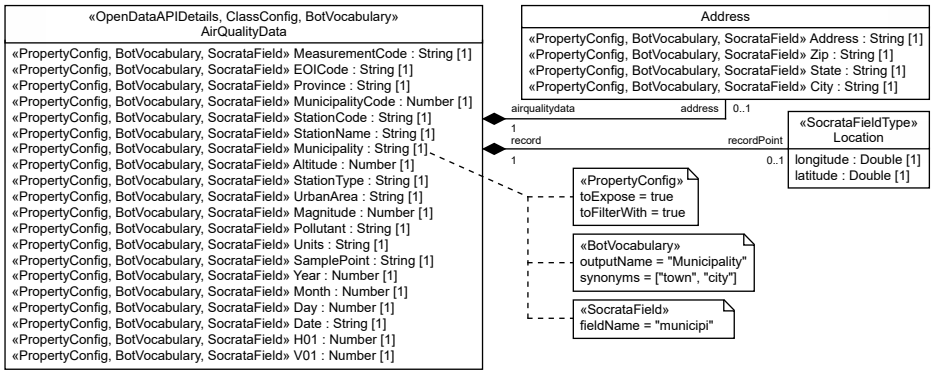


Fig. 2: UML model for the running example (our editor can show/hide the stereotypes to show a simplified representation of the diagram).

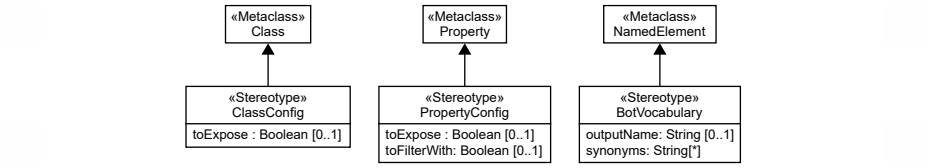


Fig. 3: Bot profile.

data the chatbot should expose, (2) how to refer to model elements (instead of the some obscure internal API identifiers), and (3) synonyms for model elements that citizens may employ when attempting to alternatively name the concept as part of a sentence.

Figure 3 shows the specification of the Bot profile. It comprises three stereotypes, namely, *ClassConfig*, *PropertyConfig* and *BotVocabulary*, extending the *Class*, *Property* and *NamedElement* UML metaclasses, respectively. The *ClassConfig* stereotype includes the *toExpose* property, in charge of defining if the annotated Class element has to be made visible to end-users via the chatbot. The *PropertyConfig* stereotype also includes the *toExpose* property, with the same purpose; plus the *toFilterWith* property, which indicates if the corresponding annotated property can be used to filter results as part of a conversation iteration. For instance, in our running example, pollution data could be filtered via date. Finally, the *BotVocabulary* stereotype can annotate almost any UML model element and allows specifying a more "readable" name to be used when printing concept information and a set of synonyms for the element.

In Figure 2 we see the Bot profile applied on the running example. Note, for instance, how we define that *town* and *city* could be used as synonyms of *Municipality* and that this attribute can be used to filter *AirQuality* results.

The OpenData profile While the previous profile is more oriented towards improving the communication between the chatbot and the user, this OpenData profile is specially aimed at defining the technical details the chatbot needs to

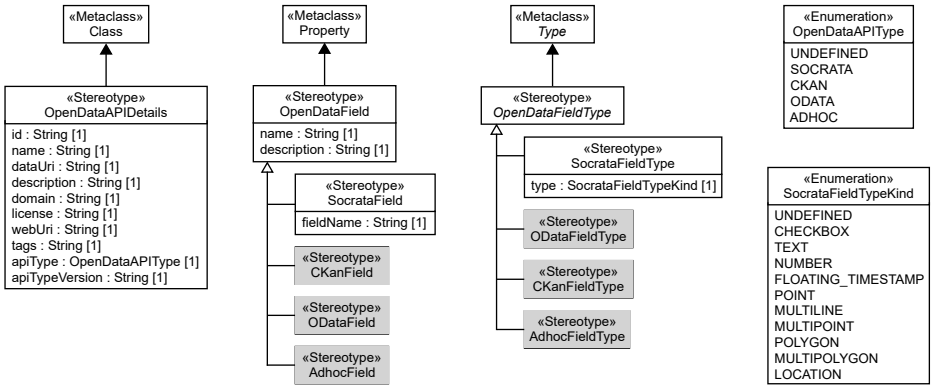


Fig. 4: OpenData profile.

know in order to communicate with the Open Data API backend. The profile defines a set of stereotypes that cover how to access the information of the model elements via the Web API. The access method depends on the specification followed by the Open Data API, which can be Socrata, CKAN, OData or OpenAPI.

Figure 4 shows the OpenData profile. As can be seen, we have defined three stereotypes, namely, *OpenDataAPIDetails*, *OpenDataField* and *OpenDataFieldType*, which extend *Class*, *Property* and *Type* UML metaclasses, respectively. The *OpenDataAPIDetails* stereotype includes a set of properties to enable the API query of the annotated UML Class. For instance, it includes the *domain* and *webUri* to specify the host and route parameters to build the query. It also includes the *APIType* property, which sets the kind of Open Data API (see values of the *OpenDataAPIType* enumeration). The *OpenDataField* stereotype annotates properties with additional information depending on the type of Open Data API used. For instance, the *SocrataField* stereotype indicates the name of the field (see *fieldName*) that has to be queried to retrieve the annotated property. Finally, the *OpenDataFieldType* stereotype includes additional information regarding the types of the properties used by the Open Data APIs.

Figure 4 also includes stereotypes prefixed with *CKAN*, *OData* and *Adhoc* (in grey) to cover the information required for CKAN, OData and OpenAPI specifications. We do not fully detail them due to the lack of space but they are available online[11]. Besides, the *Adhoc* annotations also use the OpenAPI profile [8].

As an example, this profile is also used to annotate Figure 2. While the profile is rather exhaustive and comprises plenty of detailed, technical information, note that it is automatically applied during the injection process.

4.2 Injection of Open Data Models

Injectors collect specific data items from the API descriptions in order to generate a model representation of the API. In a nutshell, regardless of the API

[11] `http://hdl.handle.net/20.500.12004/1/C/ICWE/2021/411`

specification used, the injector always collects information about the API metadata, its concepts and properties. This information is used to generate a UML model annotated with the OpenData profile. Additionally, injectors also initialize the annotations corresponding to the Bot profile with default values which will later be tuned during the refinement step (see next subsection).

In our running example, the injector takes as input the Socrata description of the data source[12] to create the UML model classes and stereotypes. To complement the definition of the data fields and their types, the injector also calls the Views API[13], an API provided by Socrata to retrieve metainformation about the data fields of datasets.

4.3 Refinement of Open Data Models

Once the injection process creates a UML schema annotated with stereotypes, the bot designer can revise and complete it to generate a more effective chatbot. The main refinement tasks cover: (a) providing default names and synonyms for model elements, which enriches the way the chatbot (and the user) can refer to such elements; and (b) set the visibility of data elements, thus enabling the designer to hide some elements of the API in the conversation.

During the refinement step, the bot designer can also revise the OpenData profile values if the API description is not fully aligned with the actual API behavior, as sometimes the specification (input of the process) unfortunately is not completely up-to-date with the API implementation deployed (e.g., type mismatches).

5 Generating the Bot

The generation process takes the annotated model as input and derives the corresponding chatbot implementation. As our proposal relies on Xatkit, this process generates the main artifacts required by such platform, specifically: (1) *intents* definition, which describes the user intentions using training sentences (e.g., the intention to retrieve a specific data point from the data source, or to filter the results), contextual information extraction, and matching conditions; and (2) *execution* definition, which specifies the chatbot behavior as a set of bindings between user intentions and response actions (e.g., displaying a message to answer a question, or calling an API endpoint to retrieve data). A similar approach could be followed when targeting other chatbot platforms as they all require similar types of input artefact definitions in order to run bots.

The main challenge when generating the chatbot implementation is to provide effective support to drive the conversation. To this aim, it is crucial to identify both the topic/s of the conversation and the aim of the chatbot, which will enable the definition of the conversation path. In our scenario, the topic/s

[12] `https://analisi.transparenciacatalunya.cat/api/views/metadata/v1/tasf-thgu.json`

[13] `https://analisi.transparenciacatalunya.cat/api/views.json?id=tasf-thgu`

is set by the API domain model (i.e., the vocabulary information embedded in the UML model and the Bot profile annotations) while the aim is to query the API endpoints (relying on the information provided by the OpenData profile).

Our approach supports two conversation modes, which are implemented in the *intents* file. Table 1 lists the main intents generated for the conversation, which we will present while describing the conversation modes. For each intent, we also generate the corresponding set of training sentences, following a predefined set of patterns that are instantiated based on the conversation context and the API vocabulary. Beyond these specific intents, Xatkit can also add by default a number of other generic intents, e.g. to provide chit-chat conversation support or help-related intents.

Direct queries The most basic communication in a chatbot is when the user directly asks what is needed (e.g., *What was the pollution yesterday?*). To support this kind of query, we generate intents for each class and attribute in the model[14] enabling users to ask for that specific information. Moreover, we also generate filtering intents that help users choose a certain property as filter to cope with queries returning too many data. Table 1, row 1, shows an example of this type of direct intent generated and a possible user utterance (i.e., concrete user input query) corresponding to this intent kind.

Guided queries We call *guided queries* those interactions where there is an exchange of questions/requests between the chatbot and the user, simulating a more natural Open Data exploration approach. Their implementation require a clear definition of the possible dialog paths driving the conversation. Table 1, rows 2-6, shows the intents generated for guided conversations, which are applied in order (starting with *GuidedSearch* and then adding filters using the rest of the intents). Figure 5a aims to summarize the possible conversation paths and the application order of the intents. The shown paths start once the user asks for a specific concept made available by the API. If the property can be filtered, the path gives the user the option to apply such a filter. This step repeats while there are other filtering options. Figure 5b shows an example of guided query for our running example.

As input assistance, both direct and guided modes include buttons as shortcuts in the conversation interface (see Figure 5b). Once the chatbot collects the request (with the possible filters) from the user, the next step is to query the involved Open Data Web API, which relies on the information provided in the OpenData profile. The implementation of this step is specified in the *execution* file, where the steps to query, filter and retrieve the information from the API are generated.

The final step in every query performed by the chatbot involves presenting and post-processing the results. In the presentation step, the user indicates the fields to show. Table 1, rows 7–8, shows the intents for setting the fields

[14] Note that this scales well as we do not actually create completely separate intents for each possible combination but use intent templates that can be instantiated at run-time over the list of elements in the model.

Table 1: Main intents generated.

Mode	Intent	Description	Example Sentence
Direct	*DirectSearch*	Shows elements given a filter	`show me all the air quality data with municipality equals to "Barcelona"`
Guided	*GuidedSearch*	Shows elements in conversation	`show me the list of air quality data`
Guided	*AddFilter*	Chooses an attribute to filter	`date`
Guided	*ChooseOperator*	Chooses an operator	`equals`
Guided	*ProvideValue*	Sets a value	`yesterday`
Guided	*EndFilter*	Ends filter for results	`I don't want to add filters`
Both	*SelectField*	Select fields for results	`municipality`
Both	*ShowResult*	Ends field selection for results	`I don't want to add fields`
Both	*AddPostFilter*	Adds a filter in results	`add filter magnitude less than "14"`
Both	*SortOrderBy*	Sorts/Orders the result	`sort by name ASC` `order by date ASC`
Both	*NextPage*	Shows the next page of results	`show me next page`
Both	*AddPostFunction*	Calls function on results	`calculate FUN ATT`

to present. Figure 5c shows an example of the result for our running example showing the fields *Municipality* and *Magnitude*. In the post-processing step, the user can apply additional filters, sort the results and paginate them. Table 1, rows 9–12, shows the intents for post-processing the results. Finally, note that our approach also incorporates aggregation functions (e.g., calculate the average, minimum o maximum) as post-processing operators. Xatkit integrates built-in pagination support to facilitate the navigation of large result sets.

6 Tool support

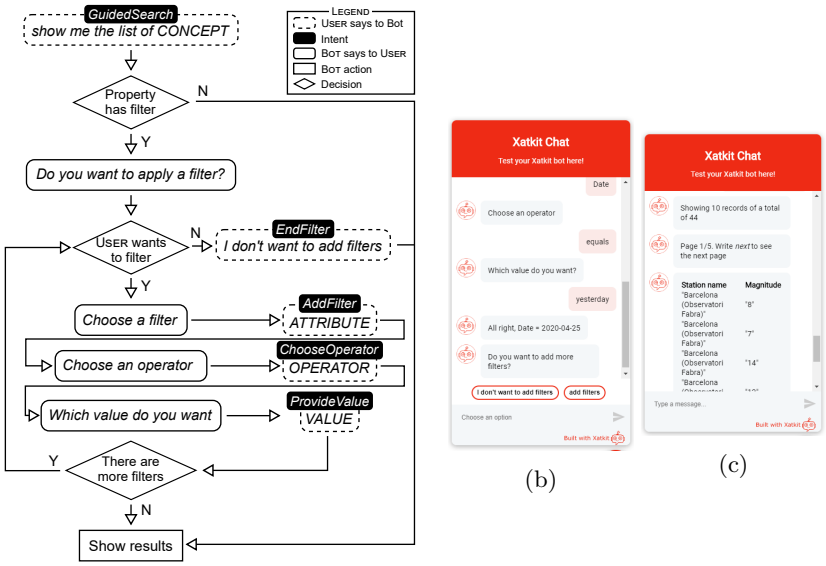
Our approach has been implemented as a new plugin for the Eclipse platform[15]. We rely on the environment extensibility and modeling support provided by Eclipse to import and generate the chatbot definition, which is then eventually executed by Xatkit.

Figure 6 shows several screenshots of the development environment. It comprises two wizards to perform the import and generation phases. During the import phase, the UML model is loaded (see wizard in Figures 6a and 6b) visualized and refined using the Papyrus modeling IDE. Once completed, our generation wizard (see Figures 6c and 6d) creates the definition of the chatbot.

7 Related Work

Facilitating the interaction with Open Data sources has been studied from different perspectives. For instance, [12] aims to generate REST APIs for Open

[15] `https://github.com/opendata-for-all/open-data-chatbot-generator`



(a) (b) (c)

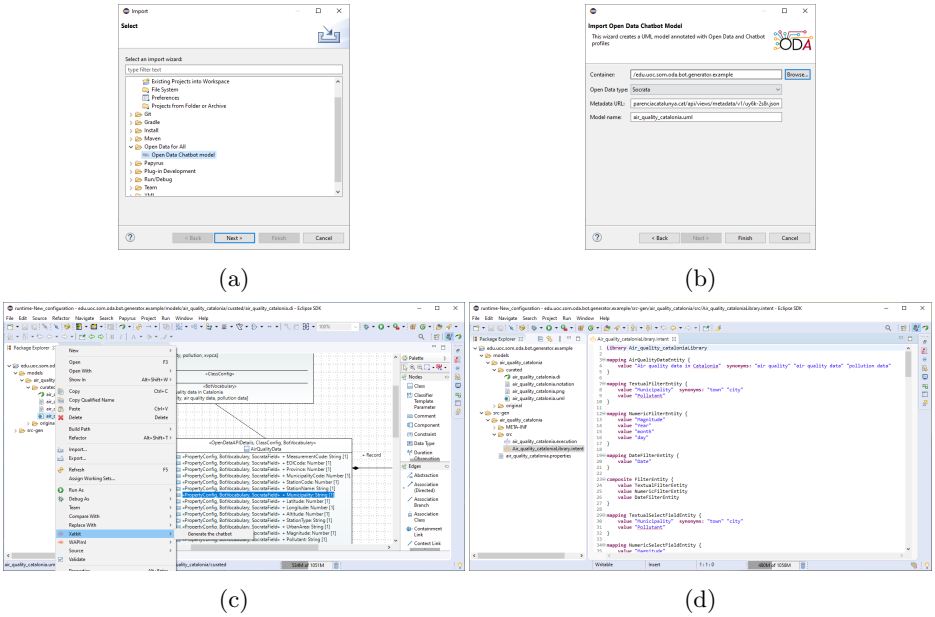
Fig. 5: (a) Conversation path in guided queries. (b) A guided conversation. (c) Showing the results.

Data sources while [3] and [11] look to generate the specification of such REST APIs to simplify their consumption by client applications.

Nevertheless, the role of chatbots in Open Data has not been widely studied. Keyner *et al.* [14] proposed a chatbot to help users find data sources in an Open Data repository by relying on geo-entity annotations. However, the chatbot only suggests the data sources to explore. It does not provide querying capabilities to consult those data sources. The work by Neumaier *et al.* [16] is similar, also focusing on the suggestion of potential useful datasets. Instead, Porreca *et al.* [19] described a case study of using a chatbot for a concrete dataset. In all cases, chatbots are manually created.

A couple of works address the creation of chatbots to query Web APIs. Our own OpenAPI bot [10] helps developers understand what they could do with an API if its OpenAPI definition is available, more than targeting the end-users. More similar to ours, the work by Vazir *et al.* [21] generates a chatbot to facilitate the execution of calls to the API itself. Nevertheless, they remain very implementation-oriented and focus on helping users learn how to query the API and assisting them in providing the right parameters for the call more than offering any abstraction mechanism to add further semantics, configuration and flexibility to the bot generation process, as we do.

Chatbot modeling and generation has also been proposed in some works (i.e., [4,20,18,5]) but none of these works proposes an end-to-end approach as ours, from the reverse engineering of the Open Data source to the generation of a chatbot actually able to directly call the initial source.



(a) (b)

(c) (d)

Fig. 6: Screenshots of the tool support: (a) and (b) the import wizard, (c) the generation wizard and (d) the generated bot.

Therefore, to the best of our knowledge, ours is the first work aimed at automatically generating chatbots to directly interact with Open Data sources using a model-based approach.

8 Conclusion

In this paper, we have presented a model-based approach to generate chatbots as user-friendly interfaces to query Open Data sources published as Web APIs. The resulting chatbot accepts both direct queries and guided conversations, where the chatbot and the user interact to precise the final query to send to the API. We have implemented our approach as an Eclipse plugin that fully supports Socrata and allows the integration of other Open Data specifications (i.e., OData, CKAN) as well as generic Web APIs (via OpenAPI specification); and generates chatbots using the Xatkit platform.

As further work, we plan to work on several extensions of this core framework: **Support for advanced queries**. Our approach supports descriptive queries where users navigate the data. However, there are other types of interesting queries; for instance, we could have: (i) diagnostic queries, which focus on the analysis of potential reasons for a fact to happen; (ii) predictive queries, aimed at exploring how a fact may evolve in the future; and (iii) prescriptive ones, which study how to reproduce a fact. We plan to extend our query templates to provide initial support for these types of queries. Some of these queries (especially

if detected at often ones) could even be an inspiration for an API extension to better match the API design with the actual information needs of the API users.

Composition of several Open Data sources. Many times, the data needs of a citizen span several Web APIs. The chatbot should be able to query and combine those different sources, dealing with potential composition links among them. This composition is not trivial and involves the well-known challenges of any data integration scenario (e.g., entity matching) plus some others more API-specific like finding the optimal paths (even based on non-functional properties), as sometimes similar information can be obtained from different overlapping sources. Existing works on API composition [9,15,6] can be used here to present to the chatbot a single unified API to simplify this process.

Massive chatbot generation for Open Data portals. Our approach works on either individual APIs or a set of interrelated ones (see the point above). We plan to extend our tool support with an automated pipeline able to retrieve and process all available APIs in a given open data portal.

Voice-driven chatbots. The growing adoption of smart assistants emphasizes the need to design chatbots supporting not only text-based conversations but also voice-based interactions. We believe that our chatbot could benefit from such a feature to improve the citizen's experience further when manipulating Open Data APIs. While Xatkit's modular architecture supports both textual and voice-based chatbots, additional research is required to translate raw data returned by the API into sentences that can be read by the bot.

Additional types of data sources. We cover the most common choices in governmental Open Data portals, but they are not the only ones. For instance, LinkedData sources, pure RDF files, GeoJSON collections, or database dumps, among others, are also used. We plan to develop additional import components that can target these technologies and integrate them into our framework.

References

1. Alghamdi, A., Owda, M.S., Crockett, K.A.: Natural language interface to relational database (NLI-RDB) through object relational mapping (ORM). In: 16th UK Workshop on Computational Intelligence. Advances in Intelligent Systems and Computing, vol. 513, pp. 449–464. Springer (2016)
2. Bizer, C., Heath, T., Berners-Lee, T.: Linked data: The story so far. In: Semantic services, interoperability and web applications: emerging concepts, pp. 205–227. IGI Global (2011)
3. Cao, H., Falleri, J., Blanc, X.: Automated Generation of REST API Specification from Plain HTML Documentation. In: Int. Conf. on Service-Oriented Computing. Lecture Notes in Computer Science, vol. 10601, pp. 453–461 (2017)
4. Castaldo, N., Daniel, F., Matera, M., Zaccaria, V.: Conversational Data Exploration. In: Int. Conf. on Web Engineering. pp. 490–497 (2019)
5. Chittò, P., Báez, M., Daniel, F., Benatallah, B.: Automatic generation of chatbots for conversational web browsing. In: Conceptual Modeling - 39th International Conference, ER 2020. Lecture Notes in Computer Science, vol. 12400, pp. 239–249. Springer (2020)

6. Cremaschi, M., Paoli, F.D.: Toward Automatic Semantic API Descriptions to Support Services Composition. In: Europ. Conf. Service-Oriented and Cloud Computing. Lecture Notes in Computer Science, vol. 10465, pp. 159–167 (2017)
7. Daniel, G., Cabot, J., Deruelle, L., Derras, M.: Xatkit: A Multimodal Low-Code Chatbot Development Framework. IEEE Access **8**, 15332–15346 (2020)
8. Ed-Douibi, H., Cánovas Izquierdo, J.L., Bordeleau, F., Cabot, J.: WAPIml: Towards a Modeling Infrastructure for Web APIs. In: Int. Conf. on Model Driven Engineering Languages and Systems Companion. pp. 748–752 (2019)
9. Ed-Douibi, H., Cánovas Izquierdo, J., Cabot, J.: APIComposer: Data-Driven Composition of REST APIs. In: Europ. Conf. Service-Oriented and Cloud Computing. Lecture Notes in Computer Science, vol. 11116, pp. 161–169 (2018)
10. Ed-Douibi, H., Daniel, G., Cabot, J.: OpenAPI Bot: A Chatbot to Help You Understand REST APIs. In: Int. Conf. on Web Engineering. p. to appear (2020)
11. Ed-Douibi, H., Izquierdo, J.L.C., Cabot, J.: Example-Driven Web API Specification Discovery. In: Anjorin, A., Espinoza, H. (eds.) Europ. Conf. on Modelling Foundations and Applications. Lecture Notes in Computer Science, vol. 10376, pp. 267–284 (2017)
12. González-Mora, C., Garrigós, I., Zubcoff, J.J., Mazón, J.: Model-based Generation of Web Application Programming Interfaces to Access Open Data (In Prepress). J. Web Eng. **19**(7-8), 194–217 (2020)
13. Kerlyl, A., Hall, P., Bull, S.: Bringing Chatbots into Education: Towards Natural Language Negotiation of Open Learner Models. In: Int. Conf. on Applications and Innovations in Intelligent Systems, pp. 179–192 (2006)
14. Keyner, S., Savenkov, V., Vakulenko, S.: Open Data Chatbot. In: Satellite Events of The Semantic Web. pp. 111–115 (2019)
15. Musyaffa, F.A., Halilaj, L., Siebes, R., Orlandi, F., Auer, S.: Minimally Invasive Semantification of Light Weight Service Descriptions. In: Int. Conf. on Web Services. pp. 672–677 (2016)
16. Neumaier, S., Savenkov, V., Vakulenko, S.: Talking Open Data. In: Satellite Events of The Semantic Web. pp. 132–136 (2017)
17. Pereira, J., Díaz, Ó.: Chatbot Dimensions that Matter: Lessons from the Trenches. In: Int. Conf. on Web Engineering. pp. 129–135 (2018)
18. Pérez-Soler, S., Daniel, G., Cabot, J., Guerra, E., de Lara, J.: Towards automating the synthesis of chatbots for conversational model query. In: Enterprise, Business-Process and Information Systems Modeling - 21st International Conference, BPMDS 2020, 25th International Conference, EMMSAD 2020. LNBIP, vol. 387, pp. 257–265 (2020)
19. Porreca, S., Leotta, F., Mecella, M., Vassos, S., Catarci, T.: Accessing Government Open Data Through Chatbots. In: Int. Workshop on Current Trends in Web Engineering. pp. 156–165 (2017)
20. Sindhgatta, R., Barros, A., Nili, A.: Modeling Conversational Agents for Service Systems. In: On the Move to Meaningful Internet Systems. pp. 552–560 (2019)
21. Vaziri, M., Mandel, L., Shinnar, A., Siméon, J., Hirzel, M.: Generating Chat Bots from Web API Specifications. In: ACM SIGPLAN Onward! pp. 44–57 (2017)
22. Xu, A., Liu, Z., Guo, Y., Sinha, V., Akkiraju, R.: A New Chatbot for Customer Service on Social Media. In: Conf. on Human Factors in Computing Systems. pp. 3506–3510 (2017)