Natural Language Data Queries on Multiple Heterogenous Data Sources

Alexander Wachtel, Dominik Fuchß, Matthias Przybylla, and Walter F. Tichy

Karlsruhe Institute of Technology
Chair for Programming Systems Prof. Walter F. Tichy
Karlsruhe, Germany
`alexander.wachtel@kit.edu, dominik.fuchss@student.kit.edu,`
`matthias.przybylla@student.kit.edu, walter.tichy@kit.edu`

Abstract. Motivated by a real-world scenario, we enable end users to query data due natural language from different sources like spreadsheets and databases. We provide a natural language user interface (NLUI) solution on how real-world entities and relations between them can be interpreted as a model to allow end user questions on the data. Therefore, the system enables end users to give instructions step-by-step, to avoid the complexity in full descriptions and give directly feedback of success. An evaluation is conducted with human users who had to perform a series of tasks using natural language. Overall, 13 end user took part in our survey with ten questions. 94.9% of all answers in the first part could be resolved on spreadsheet data, and 62,5% on SQL database.

Keywords: End User Development · Natural Language User Interfaces · Natural Language Processing; Relation Extraction · Dialog Systems.

1 Introduction

We aim for a breakthrough by making computers programmable in ordinary, unrestricted, written or spoken language. Rather than merely consuming software, users of the ever-increasing variety of digital devices and software building blocks could develop their own programs, potentially leading to novel, highly personalized, and plentiful solutions. With natural language, programming would become available to everyone. End users describe algorithms in their natural language and get a valid output by the dialog system for given description, e.g., selection sort of a set. The functionality is aimed at users with no programming knowledge, as the system enables simple routines to be programmed without prior knowledge. This makes it easier for users to get started with programming. The system also illustrates the relationship between a natural statement and its code representation, so it can also help to understand and learn a programming language [20]. At the end, the system is enabled for the object-oriented programming. Based on this, the system interprets the end user descriptions searching for classes, attributes, and methods. In general, classes represent entities from the real world, attributes specify different properties of an entity, and methods are functions or algorithms that allow to manipulate these attributes. Furthermore, end users will also be able to interact with already existing objects, e.g., Excel tables, images, graphs, but also external connections, such as connecting to SQL tables. Such objects should be addressed directly and manipulated by natural language input. In this case, our system analyses the data and allows user queries to different resources like tables, charts, and databases. End users could ask for information in their natural language on disconnected data that cannot be looked up in one step by the human.

2 Relation Extraction

In general, there are two obvious approaches to adaptation in a dialog system that works on data. On the one hand, we can adapt the system to the user. This means that we make it possible to resolve synonyms or the like. The other option is to adapt to the data. Exactly this adaptation of the data will be carried out in the following. As we have seen above, we can understand descriptions of relationships, classes and objects. But what about relationships that already exist in models, such as in and between tables. We also take a simple approach to extract from these tables. For this we abstract underlying platforms such as Excel or SQL and transfer the tables into a common model. Then we extract the contained elements and their relations from the tables.

Transformation of tables In a first step, we transform the different types of tables into a simplified table model. The first row represents all attributes. Each further row represents one object. A class is assigned to each table. Each cell represents either a primary key, a foreign key or simply an attribute. We transform the different tables into the common format via an interactive process. First, the user is asked for the class name for each table. Depending on the source the user will be asked to provide the identifying attribute for the elements of the table. In a last question, the user can specify which columns are to be used to create the object names. An object is built later from each line of the table. The information obtained in this way is used to establish connections between the tables in a follow-up step. In the last step, possible join relationships are searched. A join relationship results from the dissolution of a foreign key relationship (See Table 1). These tables show a set of owners in the first table. The second table shows different houses with their street, the number of parties and an entry to their owner. The entries in the *Owner* column represent a foreign key to the table of owners. This means that there is a relationship between the classes. To set up such join relationships between tables, the system asks the user during loading, whether the relationship is correctly recognized. If this is the case, the relationships are created automatically for each class and object. Adding SQL databases as a data source and transforming it into the table model is fairly easy. A lot of the concepts used in our table model exist in SQL databases as well. In the SQL metadata is all the information stored, necessary to answer all the questions mentioned in the subsection above. Currently only the primary key is extracted out of the metadata, since every table in SQL has a primary key.

Table 1. Houses and Owners

Name	**Reputation**
Smith	A+
Miller	A

Street	**Parties**	**Owner**
2 Piper St	9	Smith
1 Piper St	6	Smith

Relationships, classes, and objects After we have seen in subsection 2 that such join relationships can be resolved through interaction with the user, the next step is to form the relationships at the class and object level. For this purpose, a class is built from each of the tables. Columns that are not marked as foreign keys are used as attributes. Then the join relationships are converted to associations. The tables are transformed into the two classes *Owner* and *House*. These contain the attributes *Name*, and *Reputation* or *Street*, and *Parties*. Finally, one sees the association *owns*. The name of this is determined by the user in dialog with him or her. In the last step, an object is created for each row in the tables. In our example, the objects *Smith*, *Miller*, *HouseNo1* and *HouseNo2* are created. The names of the last two are generated automatically, since no name giving columns are selected (See Figure 1).

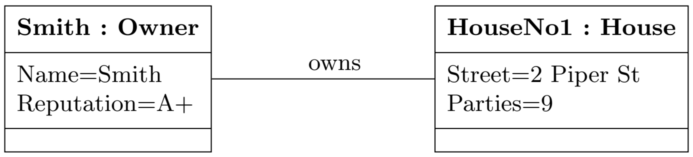

Fig. 1. Houses and Owners: Objects

Once a join relationship between tables has been established, the tables can be extended by new rows without having to rebuild the structures. The deletion of rows is also supported. In this respect, the synchronization of the underlying model with the data of the tables is possible without any further action on the part of the user. Semi-automatic extraction of models from tables allows the system to use existing data. Since data is available in various formats, we are looking forward to import data from various sources. As a first step, data from SQL databases will be processed. We support the combination of the different bases and abstract from them. For this we use the already known meta-format. Each data modification is synced by the converter incrementally. Our prototype operates on the abstract data. For this reason, it is independent on the underlying data source. For the demonstration of the compatibility with SQL databases a part of a help desk ticket system, based on a real-world scenario, is used. Before getting into details, some major differences between the two used knowledge bases have to be discussed. Generally the User can work on the Excel data better than a program, for SQL is the applies the opposite. In Excel no unnecessary information, like keys, are added and nothing is encoded. SQL on the contrary is based on encoding objects and its tables to maintain a clear structure, for an easy extraction of information. The simple conversion into the abstract data model is described in the following steps.

3 Natural Language Query Handling

Section 2 has shown how these models can also be obtained from tables. We are providing a concept to deal with different types of knowledge bases. The following scenario shows how these relationships between entities can be used to resolve user requests. We will show why adaptation in the direction of the data makes sense and how this can be used. The concept of countries and states is used for this, whereby we use the Austrian provinces with their capital, population and area as example. A human could simply answer the question “What is the overall area of Austria?” by calculating the sum of areas of all provinces. For this, the computer needs knowledge of the connection between the countries, states, Austria and the states of Austria. We can simply explain the concept by writing “A country consists of states” into the system. Afterwards, we can use the extraction process as shown in section 2 to load the data from the table. As a last step we have to explain that Austria is a country which has the states listed in the table. This is also possible by using natural language interface of JustLingo. Afterwards, the system is ready to answer user requests. Some simple requests are shown in Figure 2. We see three questions and the provided answers of the system. Currently, two groups of questions and a fallback is implemented to answer simple questions.

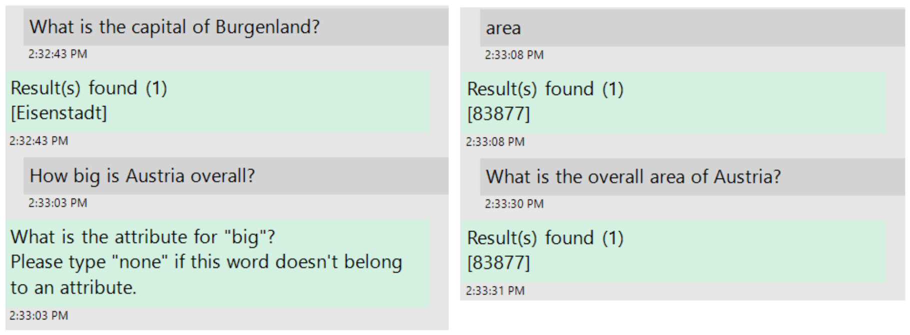
What is the capital of Burgenland?
2:32:43 PM
Result(s) found (1)
[Eisenstadt]
2:32:43 PM
How big is Austria overall?
2:33:03 PM
What is the attribute for "big"?
Please type "none" if this word doesn't belong to an attribute.
2:33:03 PM
area
2:33:08 PM
Result(s) found (1)
[83877]
2:33:08 PM
What is the overall area of Austria?
2:33:30 PM
Result(s) found (1)
[83877]
2:33:31 PM

Fig. 2. JustLingo NLUI: Example Interaction

User Queries The first type of request is the direct query of attributes. These are queries that refer directly to an attribute and a set of objects. E.g. “What are the capitals of the states of Austria?” To generate the answer, we first map the linguistic elements of the query to the elements in the model. After that we identify the *start* of the query. In the example case this would be the object Austria. As soon as this has been identified a search for the *target* will start. In our example the target would be the linguistic element *capitals*. With this information the search of connecting paths will be invoked. After this step the system obtains a set of 9 possible paths starting at the object Austria and ending at an element which represents the linguistic element *capitals*. These are the paths from the object Austria to the capitals of all 9 states of Austria. E.g. the path defined as $[Austria \rightarrow country\text{-}has\text{-}state\text{-}relation \rightarrow Styria \rightarrow Capital \rightarrow Graz]$. To discard paths that are not necessary for the result, the paths will be rated. To do this, the other mappings are compared with the paths. In our example, the mapping of the linguistic element *states* to the class State would be used. The evaluation of the paths found this way is explained detailed in subsection 3. The second type of queries are those in which the user paraphrases an attribute with an adjective. These are called indirect query of attributes. E.g. "How big is Austria overall?" To resolve this kind of queries the system uses the possibility of interaction with the user. Therefore, the system asks for which property is meant. After the answer of the user the system has learned the meaning of *big* for this context. Thus, the system would not ask the user again, about the meaning of *big*. All follow-up steps are performed in the same fashion as performed in the approach to resolve direct requests. We have made the resolution of requests extensible. In particular, we have created a fallback handler in case a request does not fit into any schema. This fallback tries to analyze a given request by extracting all possible mappings between linguistic elements of the query and all entities. After extraction, the candidates of each element are sorted by the type of entities. The idea here is, for example, to prefer named objects to classes in order to limit the size of the later answer.

Evaluation of Paths In this last step the final answer of the system will be generated. In a first step, the system tries to reduce the number of paths by adding further assignments of the input elements to model elements. Afterwards, every path will rated according to the appearance and order of the mappings within each path. The second step will only be applied to the group of paths with the highest rank. The second step consists of the application of several rules to determine the answer to the question. Please mention that by prerequisites the paths are equally structured and therefore have equal *lengths*. If a rule has been applied, we will start again at the beginning to check the rules. Rule 1 is used to abort the reduction if the length of the paths is too short for further simplification. Rule 2 reduces attributes reached to the value of the attribute for an object. Rule 3 combines all existing values if they do not have to be combined by any relation. The system therefore checks whether objects precede these values as the last element. Rule 4 simplifies the paths, if a value has already been created at the end of the paths, and the corresponding objects precede them by deleting the affected objects. Rules 5 and Rule 6 deal with the occurrence of relationships. If only one path exists, no function is used to merge the values, since only one value exists. Otherwise, a function is used that summarizes the values according to the relationships found. Rule 7 intercepts the case that the user did not ask for an attribute, but for objects. In this case, the objects are replaced by the values of the names. User interaction may needed to find the function which shall be used to aggregate the values if the function could not be determined in Rule 7. Afterwards, the result is presented to the user.

Rebuilding Path Search Since adding the compatibility of using different kind of knowledge bases, the amount of Data has grown significantly. A deep search for paths between tables couldn't keep up with the growing data. To avoid searching for too long paths, which would be either way eliminated in the evaluation of paths(See section 3), a quicker way of searching is a simple routing algorithm, in this case Dijkstra. As a first step, the program looks through the data with generated hints, to categorize hits into classes, object, attributes name or attributes value. If data in at least two categories is found, the search for a path can be done. Connections between the tables have been defined as in Section 2 mentioned, therefore performing a routing between the two tables, in which the data was found, is a simple operation. Because of that only the shortest possible connections between the two tables are found. This allows now the answer that there is no answer to that question, instead of finding an answer, which has a too long path. The paths found still will be Evaluated as mentioned in section 3.

Using Results for Complex functions As we have seen, our system can answer a user's simple question. We have extended the natural language interface so that the answers to such questions can be linked further. For this purpose, functions that we define in natural language can be used. For this we use the already known mechanisms for creating algorithms from natural language. Thus, we achieve a great flexibility in the way of linking. For example, intermediate results can be used as parameters for stochastic functions, or they can be ordered by sorting algorithms. This makes it possible to split more complex queries that the system would otherwise not understand into individual operations that the system understands.

4 Evaluation

As a first user study we designed a study in which undergraduate computer science students were asked to solve tasks in two different classes of complexity. This approach is used to evaluate the question and answering module and test its limits. The user study should show to what extent the system discussed in section 2 and 3 can answer user requests. In the first exercise of the user study, the 13 participants are given the task of asking questions to the system. As data basis the model from section 3 is used here, which contains the relationship from countries and federal states, as well as Austria with its states as data [2]. In the evaluation, the participants were then given tasks such as *Determine the average population of all Austrian states.* and had to use the system to get the answer.

First, the *query error rate* for a set of queries Q and a set of answers A is defined as follows:

$$QER(A, Q) = \frac{\#(answer\ wrong) + \#(no\ answer)}{|Q|}$$

This measure is based on typical measures of NLP. Secondly, we consider the number of *not understood inputs* ($\#(NUI)$) to the system. In the study, nine participants were able to solve all six tasks of this first part. The remaining four participants were each unable to solve one of the tasks. In total a QER of 0.051 was achieved. The average for $\#(NUI)$ was 2.62. The second part of the study examined how these results change when several tables are linked. In addition, the participants should define their own task after they have solved the all set tasks. For this part of the evaluation, two tables have been adopted by Eurostat and relations between them have been established [17,18]. All in all, the participants received three tasks for which they had to formulate a question and were allowed to create a task themselves. A predefined task in this context was exemplary *Determine the average population in Belgium in 2012 and 2013.* Here the QER was 0.26 and the $\#(NUI)$ was 2.31. These values are based on the fact that all participants, if they have formulated their own task (12 participants), have made a more complex request, which the system did not understand. These complex questions were characterized by comparisons or orders of data.

Multiple Datasources To be able to compare the changes made, the same input from the user study was used. The evaluation of this part only focuses on questions for data. Evaluating the generation of UML diagrams is neglected. For the same user input, the following results were produced: QER was 0.72 and the NUI was 2.70. Compared to the previous results a worsening in finding the correct answer is clearly visible. With these results the new answer finding got tuned to make the final evaluation. The final evaluation represents our intuition to include other kind of databases into the program. Since there are no other programs available to connect cross-platform databases and ask questions in natural language. In this evaluation tables in excel sheets and a SQL-database are used. Real data is used for this.

5 Related Work

Paternò [13] introduces the motivations behind end user programming defined by Liberman [8] and discusses its basic concepts, and reviews the current state of art. In 2006, Myers [11] provides an overview of the research in the area of End-User Programming. As he summarized, many systems for End User Development have already been realized [7]. During a study in 2006, Ko [7] identifies six learning barriers in End User Programming: design, selection, coordination, use, understanding and information barriers. In 2008, Dorner [16] describes and classifies End User Development approaches taken from the literature, which are suitable approaches for different groups of end users. Implementing the right mixture of these approaches leads to embedded design environments, having a gentle slope of complexity. Such environments enable differently skilled end users to perform system adaptations on their own. Sestoft [15] increases expressiveness and emphasizing execution speed of the functions thus defined by supporting recursive and higher-order functions, and fast execution by a careful choice of data representation and compiler technology. Cunha [4] realizes techniques for model-driven spreadsheet engineering that employs bidirectional transformations to maintain spreadsheet models and synchronized instances. The idea of programming in natural language was first proposed by Sammet in 1966 [14]. One of the difficulties is that natural language programming requires a domain-aware counterpart that asks for clarification, thereby overcoming the chief disadvantages of natural language, namely ambiguity and imprecision. [6] describes a method for the automatic acquisition of the hyponymy lexical relation from unrestricted text. This approach avoids the need for pre-encoded knowledge and also applicability across a wide range of text. The method discovers patterns and suggests that other lexical relations will also be acquirable in this way. In recent years, significant advances in natural language techniques have been made, leading, for instance, to IBM's Watson [5] computer winning against the two Jeopardy! world champions, Apple's Siri routinely answering wide-ranging, spoken queries, and automated translation services such as Google's becoming usable [9, 12]. In 1979, Ballard et al. [1] introduced their Natural Language Computer (NLC) that enables users to program arithmetic calculations using natural language. Metafor introduced by Liu et al. [10] has a different orientation. Based on user stories the system tries to derive program structures to support software design. NLP2Code [3] enables developers to request for code snippets from the Stack Overflow, and integrates these snippets directly into the source code editor.

6 Conclusion and future work

In this paper, we presented a solution on extracting the real-world entities from human descriptions. From the meta representation we then create meta models and generate source code. After that, our system allows end user to query data in disconnected tables in natural unrestricted language. However, programming in natural language remains an open challenge [12]. There are several user inputs $i_2, \dots, i_N$ that are similar to i_1 and also map to the known action a_1. There is also a problem on the classification of required system action or skill. The match of the user input i_1 to system action a_1 is unique until the function of the system could be clearly divided by the syntax of the user input. In case of object-oriented programming, the user input can be referenced to the Algorithm skill or Class interpretation skill. They have similar inputs but different actions on execution. For this reason, it is not enough to check the language on the syntactical level. Furthermore, the syntax matching should also be extended by the interpretation on the meta level. These methods should combine syntax, semantic and context classification. In our next paper, we will present the classification with several different classifiers. The result of the individual classifiers is summed and forms the overall similarity of the input to the skill. Ordinary, natural language would enable almost anyone to program and would thus cause a fundamental shift in the way computers are used. Rather than being a mere consumer of programs written by others, each user could write his or her own programs [19].

References

1. Ballard, B.W., Biermann, A.W.: Programming in natural language:nlc as a prototype. In: Proceedings of the 1979 annual conference. pp. 228–237. ACM (1979)
2. Bundesanstalt Statistik Österreich: Regional Atlas Austria (Online Atlas), `statistik.at/web_de/services/regionalatlas_oesterreich/index.html`
3. Campbell, B.A., Treude, C.: NLP2Code: Code Snippet Content Assist via Natural Language Tasks. ICSME (2017)
4. Cunha, J., Fernandes, J., Mendes, J., Pacheco, H., Saraiva, J.: Bidirectional transformation of model-driven spreadsheets. In: ICMT (2012)
5. Ferrucci, D., Brown, E., Chu-Carroll, J., Fan, J., Gondek, D., Kalyanpur, A.A., Lally, A., Murdock, J.W., Nyberg, E., Prager, J., et al.: Building watson: An overview of the deepqa project. AI magazine **31**(3) (2010)
6. Hearst, M.A.: Automatic acquisition of hyponyms from large text corpora. In: Proceedings of the 14th Conference on Computational Linguistics - Volume 2 (1992)
7. Ko, A.J., Myers, B.A.: Designing the whyline: a debugging interface for asking questions about program behavior. In: Proceedings of the SIGCHI conference on Human factors in computing systems (2004)
8. Lieberman, H., Paternò, F., Klann, M., Wulf, V.: End-user development: An emerging paradigm. In: End user development, pp. 1–8. Springer (2006)
9. Liu, H., Lieberman, H.: Toward a programmatic semantics of natural language. In: IEEE Symposium on Visual Languages and Human Centric Computing (2004)
10. Liu, H., Lieberman, H.: Metafor: visualizing stories as code. In: Proceedings of the 10th international conference on Intelligent user interfaces. ACM (2005)
11. Myers, B., Ko, A., Burnett, M.: Invited research overview: end-user programming. In: CHI extended abstracts on Human factors in computing systems (2006)
12. Ortiz, C.L.: The road to natural conversational speech interfaces. IEEE Internet Computing **18**(2), 74–78 (2014)
13. Paternò, F.: End user development: Survey of an emerging field for empowering people. ISRN Software Engineering (2013)
14. Sammet, J.E.: The use of english as a programming language. Communication of the ACM (1966)
15. Sestoft, P., Sørensen, J.Z.: Sheet-defined functions: implementation and initial evaluation. In: International Symposium on End User Development. Springer (2013)
16. Spahn, M., Dörner, C., Wulf, V.: End user development: Approaches towards a flexible software design. In: ECIS. pp. 303–314 (2008)
17. Statistical Office of the European Union (Eurostat): Population on 1 january by age and sex (2017), `appsso.eurostat.ec.europa.eu/nui/show.do?dataset=demo_pjan`
18. Statistical Office of the European Union (Eurostat): House price index - annual data (2018), `appsso.eurostat.ec.europa.eu/nui/show.do?dataset=prc_hpi_a`
19. Tichy, W.F., Landhäußer, M., Körner, S.J.: Universal Programmability - How AI Can Help. Artificial Intelligence Synergies in Software Engineering (May 2013)
20. Wachtel, A., Eurich, F., Tichy, W.F.: Programming In Natural Language Building Algorithms From Human Descriptions. The Eleventh International Conference on Advances in Computer-Human Interactions (March 2018)