Facilitation of Health Professionals Responsible Autonomy with Easy-to-Use Hospital Data Querying Language

Edgars Rencis[1(✉)], Juris Barzdins[2], Mikus Grasmanis[1], and Agris Sostaks[1]

[1] Institute of Mathematics and Computer Science, Faculty of Computing, University of Latvia, 29 Raina blvd., Riga 1459, Latvia
{edgars.rencis,mikus.grasmanis,agris.sostaks}@lumii.lv
[2] Centre of Health Management and Informatics, Faculty of Medicine, University of Latvia, 19 Raina blvd., Riga 1586, Latvia
juris.barzdins@lu.lv

Abstract. Support for the development of responsible autonomy as opposite to management that is based on direct control is found to be by far more effective approach in healthcare management, especially when it concerns physicians as the most influential group of health professionals. It is therefore important to obtain a process-oriented knowledge system where physicians would be able to autonomously answer questions which are outside the scope of pre-made direct control reports. However, the ad-hoc data querying process is slow and error-prone due to inability of health professionals to access data directly without involving IT experts. The problem lies in the complexity of means used to query data. We propose a new natural language- and semistar ontology-based ad-hoc data querying approach which reduces the steep learning curve required to be able to query data. The proposed approach would significantly decrease the time needed to master the ad-hoc data querying thus allowing health professionals an independent exploration of the data.

Keywords: Responsible autonomy · Hospital management
Self-service knowledge system · Ad-hoc querying · Semistar ontologies
Controlled natural language · Hierarchical data · Medical data

1 Motivation for Further Development of Responsible Autonomy of Healthcare Professionals

Responsible autonomy as opposite to direct control is an organizational system that is based on individual or group autonomy [1]. In healthcare domain, professional autonomy of physicians for independent bedside decisions (accordingly to their specialization certificate) is nowadays increasingly linked also to responsibility to balance clinical and resource dimensions of the whole organization [2]. This is a profound change in the medical profession, and it arises from a variety of interconnected factors linked to one main factor – the enormous progress of medical science that comes at a significant financial cost.

A. Lupeikiene et al. (Eds.): DB&IS 2018, CCIS 838, pp. 202–215, 2018.
https://doi.org/10.1007/978-3-319-97571-9_17

The buy-in of the most critical group in healthcare, the physicians, is regarded as mandatory precondition for almost any organizational innovation including also any managerial effort to guide operational and systemic changes by exponentially growing the amount of clinical process efficiency- and quality-oriented healthcare data. In the context of fundamental transformation needed to improve the safety and quality of healthcare, Braithwaite et al. [3] propose "harnessing the natural properties which emerge (often spontaneously) at the interface between the socio (human behavioral) and technical components of complex systems" arguing that "a bottom-up strategy led by clinicians is badly needed to balance the predominantly top-down approaches which frequently result in only modest improvements which are difficult to sustain" [3]. In a wider meaning of the responsible autonomy a bottom-up approach is also needed to access and use the administrative data representing the utilization of limited resources and patient outcomes beyond a single episode of care. However, the complexity and heterogeneity of such data demand either advanced programming and data processing skills to query them, or subordinated data professionals. Both of those options are currently available to management teams, but not to physicians. This fact could be regarded as an obstacle for further development of responsible autonomy, as the care processes are seen through the perspective of performance indicators pre-made by the management rather than through creative exploration of data by physicians themselves.

This situation highlights the need for health professionals to also benefit from advances in IT and to overcome the insufficient opportunity of direct learning from their own practice. To take the responsibility for their own performance and for outcomes of their own concrete decisions, there is a clear demand for a tool that would allow motivated physicians to independently find answers to questions which are outside the scope of pre-made reports.

This paper presents a possible solution to the problem mentioned above. Section 2 gives insight into the related work done in the field of querying the data by domain experts. A lot of work has been done here, but a fully satisfying solution still doesn't exist there. Section 3 introduces the concept of Semistar data ontology as a convenient format for storing healthcare data. Section 4 displays the proposed natural language-based querying language that works upon semistar ontologies. The language is already implemented, and some tests have been done with it. Practical experiments with the language have shown the need for another important facet – the subclass definition feature. This has been described in Sect. 5. Section 6 describes a case study and experiments with groups of domain experts trying to exploit the language in their everyday life. Section 7 concludes the paper.

2 Ad-Hoc Data Querying for Medical Professionals

Direct access to medical data by healthcare professionals would be beneficiary for independently answering questions which are outside the scope of pre-made reports. However, there are no satisfying ad-hoc data querying tools available for end-users which are non-programmers. The main obstacle is the fact that end-users do not have the required skills to define queries by themselves. The main reason is the complexity

of the query languages used to get answers from databases. Three main problems are: how to describe data to be easily perceived by end-users; how to formulate queries simply enough for end-users; how to execute queries efficiently in order to retrieve answers in a reasonable time.

Most of the data are nowadays stored in relational databases. The SQL (SEQUEL) language is the de facto standard of querying such data. However, the way data (ER-model) and queries are described in the SQL is too complicated for end-users. There are similar languages for data querying in other types of data storage, e.g. SPARQL for RDF ontologies. The SPARQL as well as SQL requires an accurate formulation of the query (both semantics and syntax) and solid skills in the underlying technology. It thus makes a definition of queries too complex to learn and use. To reduce the complexity wrappers have been made for SQL and SPARQL, e.g. a graphical query builder "Graphical Query Designer" for SQL Server, a graphical query builder "ViziQuer" [4] for SPARQL and RDF databases, a form-based tool which uses standard GUI elements (e.g. tables and lists) and wizards "SAP Quick Viewer SQVI". Another means for direct data access is Self-Service Business Intelligence by Microsoft [5]. It offers a set of tools called "Power BI". Power BI allows an end-user creating advanced visualizations of data and performing the data analysis through a spreadsheet application (MS Excel). Although even IBM with its Watson Analytics [6] is dealing with the direct data access problem, a flawless solution has not been found yet. The most significant downside of these approaches is a steep learning curve necessary to master a new query approach and to understand the way the data are described.

Another feasible option for medical professionals is a natural language or more precisely – a natural language interface to databases (NLIDB) through which end-users can query data in the relational databases [7–11]. However, the definition of the accurate query itself is a hard task for end-users without IT background. NLIDB systems are not easily usable by end-users due to the problem of linguistic coverage. Firstly, explanations to users are needed about what the database contains and what types of questions can be asked. Data descriptions used by database experts (like ER models) are too complicated and contain too many technical details to be useable for the understanding of the data. Secondly, the software does not properly understand what a user means by a query due to the ambiguity and richness of the natural language. Data schema representation satisfying requirements of both human and software is needed. NLDIB uses ontologies to define concepts, relationships between them and their properties. Concepts form a vocabulary that can be used by end-users. Although ontologies hide technical details they require the knowledge of basic object-oriented modeling principles. The NLIDB approaches have thus not been used widely, especially for complex querying with nontrivial calculations.

Process visualization as "an easy way to discover and create value" [12] is often mentioned in engineering and lean management literature as another way to explain the data to the end-user. However, meaningful description and visualization of pathways of individual patients as a clinical process of the whole hospital is a complicated task due to its extremely variable nature. Therefore Vos et al. [12] propose to use a generalized visual model of hospital processes for clinicians to understand what data, where and when during the treatment are recorded. If the clinicians understand the localization of

the data in the model they may explicitly and clearly formulate explorative questions for building process-oriented knowledge base. As examples of the representative clinical process variables or of numerical indicators (with desirable limits set) the following should be mentioned: average treatment time, mortality rate and other clinical outcomes of patients, frequency of hospitalization, waiting time at various stages of the process, patient satisfaction, utilization of resources (rooms, equipment, staff), costs of certain manipulations or patient groups, costs of certain patients, details of particular event – when it has happened, how many times, within what time interval etc.

3 In Search of the Appropriate Format for Storing Healthcare Data

When we think about the various data representation formats from which to choose, several options can be available. One of the most exploited data storage formats is the relational database, because usually the data can indeed be represented in the form of ER model. If we choose to use this data storage format, we are later able to query the ontology using the SQL as a query language. This is a common solution, and thus it is quite easily to implement it.

However, we must consider not only the ease of implementation of the chosen solution, but also its friendliness to end-users that are not IT specialists. As was stated in Sect. 1, the main types of users of our system will be healthcare professionals (managers and physicians), so we cannot assume that for them the ER model would be the best representation of healthcare data in a natural and understandable way. Since ER model is almost never granular (naturally dividable into data slices), it is usually not easily understandable by non-programmers [13]. Moreover, although the SQL language was initially designed to be used by standard end-users, hardly any non-programmer has nowadays acquired the necessary skills to be able to understand SQL queries, not to mention writing them him/herself.

We therefore have to cope with at least two challenges: (1) how to depict the data ontology to be easily understandable by healthcare professionals; (2) how to develop a query language based on this representation of the underlying data ontology, such that a healthcare professional could formulate queries him/herself and understand their answers. Finally, if we had developed such a user-friendly query language, we would then encounter also the third challenge: how to implement the query language efficiently enough in order to get the answer to a sufficiently wide class of queries in a reasonable time. These three challenges together form the so-called “3How” problem which we have described in more detail in our previous work [13].

If we now think about the most suitable format for storing healthcare data, we should look at how these data were stored before they were digitalized. When the information about patients was filled in by hand, hospitals used so-called patient cards where each patient had his separate card and each card contained information about each occurrence of this patient in the hospital, and each occurrence contained information about the treatments provided for the patient in this particular occurrence and so on. This division into smaller and smaller subdivisions is a very natural way of storing healthcare data, and it is also very familiar to healthcare professionals. We have therefore chosen exactly

such structure to be the basis of the data ontology that solves the first challenge of the abovementioned "3How" problem. The described structure is known in literature as the reversed star data schema, because in it "certain key characteristics of the classic star schema are 'reversed'" [14]. Indeed, in typical situations we always have one central class (the class "Patient" in this case), from which several paths can lead to other connected classes that have the relation one-to-many, as can be seen in Fig. 1. It is also known that any database organized in the third normal form can be converted to a reversed star schema thus making the reversed star ontologies very powerful [15]. In addition to the classical reversed star ontologies we also allow addition classes outside the star which are devoted for registers and classifiers (classes "CPhysician", "CDiagnosis" and "CManipulation" in Fig. 1). We call such enriched ontologies the semistar ontologies, and we have described them in detail in our previous work [13, 16–19]. The simplified version of a semistar ontology seen in Fig. 1 has been introduced for use in Riga Children's Clinical University Hospital (RCCUH).

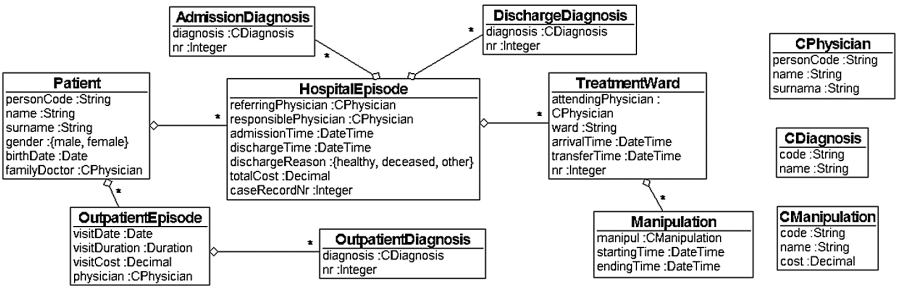

Fig. 1. Semistar ontology exploited in Riga Children's Clinical University Hospital

Semistar ontologies are by their nature granular, that is – they can be naturally divided into slices [13, 19] where each slice contains concluded information about one particular patient. This intrinsic feature of semistar ontologies has allowed us developing a new kind of querying language that would solve the second challenge of the abovementioned "3How" problem. The goal for the language was to be very easily understandable by healthcare professionals, so that they would be able to formulate their own ad-hoc queries in a convenient manner. This basic feature of the language was tested in an experiment with healthcare professionals. This is described in more detail in Sect. 6. The language itself is briefly described in Sect. 4.

Finally, we have also solved the last challenge of the "3How" problem by developing a very efficient implementation of the proposed query language [13]. Details of the implementation are, however, outside the scope of this paper.

4 Development of a Natural Language-Based Query Language

The main reason for choosing a semistar ontology for storing the data is its simplicity from the end-user point of view. It has been proved that this format is inherently intuitive for domain experts [17, 18]. It can be explained by the fact that this format is very similar

to the old paper forms that all healthcare professionals are familiar with. Moreover, the data storing format of the semistar ontology also allows us developing an easy-understandable query language in a natural manner. The query languages are usually complicated due to the technical details forced by non-intuitive data storage formats (such as relation databases and their respective query language SQL). For example, the concept of a role name can be quite hard to comprehend by non-programmers. In the case of semistar ontologies we have managed to not embed such technical details in the query language. As a result we have succeeded in developing a query language that is based purely on the natural language and exploits only those concepts included in the ontology. Therefore during the language development process we had hoped that also the query language would turn out to be understandable by end-users who are familiar with its underlying semistar ontology and its concepts. Our hopes proved right after we performed the end-user survey described in Sect. 6.

Of course, the natural language, on which the query language is based, is still quite controlled. Nevertheless, it is much more suitable for healthcare professionals (physicians and hospital managers) than SQL-like languages.

The level of control of the natural language grammar forms allowed in our query language is defined by seven sentence constructs called sentence templates. These templates are described in more detail in our previous work [17], but we are listing them also here for the purpose of gaining more general understanding of the proposed query language.

The sentence templates are written in a natural language and contain placeholders for inserting conditions and expressions written in special syntax. Formal explanation of this syntax is not a part of this paper, but it can be done using a special kind or grammar called the Definite Clause Grammar. Here we will instead explain the construction of those conditions using examples.

Let us now introduce the seven sentence templates illustrated by examples.

T1. COUNT AClass [x] WHERE <selection condition>

Semantics: counts instances of AClass, which satisfy the selection condition.

Example:

- COUNT Patients, WHERE EXISTS HospitalEpisode, WHERE referringPhysician=familyDoctor (count of patients who have been referred to hospital by their family doctors);

T2. {SUM/MAX/MIN/AVG/MOST} <attribute expression> FROM AClass [x] WHERE <selection condition>

Semantics: selects instances of AClass, which satisfy the selection condition, calculates the attribute expression for each of these instances obtaining a list to which the specified aggregate function is then applied. Example:

- SUM totalCost FROM HospitalEpisodes, WHERE dischargeReason=healthy AND birthDate.year() =2012 (how much successful treatments of patients born in 2012 have cost);

T3. SELECT FROM AClass [x] WHERE <selection condition> ATTRIBUTE <attribute expression> ALL DISTINCT VALUES

The semantics is obvious. Example:

- SELECT FROM HospitalEpisodes, WHERE dischargeReason=deceased, ATTRIBUTE responsiblePhysician.surname ALL DISTINCT VALUES;

T4. SHOW [n/ALL] AClass WHERE <selection condition>

The semantics: shows n or all instances of AClass which satisfy the selection condition.

T5. FULLSHOW [n/all] AClass WHERE <selection condition>

The semantics: the same as "show", but shows also the child class instances attached to the selected AClass instances.

T6. SELECT AClass x WHERE <selection condition>, DEFINE TABLE <x-expr'1> [(COLUMN C1], …, <x-expr'n> [(COLUMN Cn)] [, KEEP ROWS WHERE <Ci selection condition>] [, SORT [ASCENDING/DESCENDING] BY COLUMN Ci] [, LEAVE [FIRST/LAST] n ROWS]

The semantics: selects all instances of AClass, which satisfy the selection condition, then makes a table with columns C1 to Cn, which for every selected AClass instance x contains an individual row, which in column C1 contains the value of the <x-expr'1>, …, in column Cn contains the value of the <x-expr'n>. Then it is possible to perform some basic operations with the table like filtering out unnecessary rows, sorting the rows by values of some column and then taking just the first or the last n rows from the table. Example:

- SELECT HospitalEpisodes x, WHERE dischargeReason=deceased, DEFINE TABLE x.surname (COLUMN Surname), x.dischargeTime.date() (COLUMN Death_date), (COUNT x.Manipulation, WHERE manipul.code=02078) (COLUMN Count_02078), (SUM manipul.cost FROM x.Manipulation, WHERE manipul.code=02078) (COLUMN cost_02078);

T7. There are two more cases in the definition of the table, where table rows come from some other source, not being instances of some class. Being very similar, these two cases form two subtemplates of the last template:

(a) SELECT FROM AClass [a] WHERE <selection condition> ATTRIBUTE <attribute expression> ALL DISTINCT VALUES x, DEFINE TABLE…
(b) SELECT FROM INTERVAL (start-end) ALL VALUES x, DEFINE TABLE…

The semantics of both cases is obvious. Examples:

- SELECT FROM TreatmentWards ATTRIBUTE ward ALL DISTINCT VALUES x, DEFINE TABLE x (COLUMN Ward), (SUM manipul.cost FROM Manipulations, WHERE ward=x) (COLUMN Cost);

- SELECT FROM INTERVAL (1–12) ALL DISTINCT VALUES x, DEFINE TABLE x (COLUMN Month), (COUNT HospitalEpisodes, WHERE admissionTime.month()=x) (COLUMN Episode_count) (MOST diagnosis.code FROM AdmissionDiagnoses, WHERE nr=1 AND admissionTime.month()=x) (COLUMN Most_frequent_main_diagnosis).

5 The Subclass Definition Feature

Practical experiments with former version of our query language had shown [17] that there was yet one more very important feature that should be added to increase the usability of the language – the subclass definition feature. It was very common that users

wanted to exploit a certain subclass of some ontology class multiple times in their querying process. It was not very convenient to repeat the part of the query defining the class each time that class is needed. It would be much more convenient if one could develop the class only once and then use the class name in each of the subsequent occurrences.

Let us inspect an example of when the introduction of a subclass could be beneficial. Let us say we want to deal with patients who have been healed in the hospital and calculate the total cost of their hospital episodes in which they have been healed. The query that answers this question looks like this:

SUM totalCost FROM HospitalEpisode WHERE dischargeReason = healthy

Now we would perhaps like to refine our query and ask how many of those episodes lasted more than a week:

COUNT HospitalEpisode WHERE dischargeReason=healthy AND discharge-Time-admissionTime>7d

Or we could want to create a table of 10 most expensive episodes and show the responsible physician for each of these episodes:

SELECT HospitalEpisode WHERE dischargeReason=healthy, DEFINE TABLE totalCost (COLUMN Cost), name (COLUMN Patient name), surname (COLUMN Patient surname), responsiblePhysician.name (COLUMN Physician name), responsiblePhysician.surname (COLUMN Physician surname), SORT DESCENDING BY COLUMN Cost, LEAVE FIRST 10 ROWS

We can see that here persists the need for repeating the definition of hospital episodes with a healthy outcome. It would be more convenient if we had a class name (e.g. HealthyEpisode) for such episodes. When introducing new features in the language we must consider two things – the usability from the end-user point of view and the performance of query execution from the implementation point of view. To optimize the performance we had to consider several possible subclass implementation options:

(1) to translate each definition of subclass S of class A to a predicate PS<A> and then substitute every occurrence of class S in queries with A and PS<A>;
(2) to create a fully materialized view in the subclass definition moment by calculating values of the defining predicate for each instance of the class A and to store this table in the persistent database;
(3) to create the subclass definition table when the subclass is first used and to store this table in the session memory;
(4) to store the information about the subclass affiliation in the instance level of the class A by using the lazy evaluation principle and to calculate the affiliation of particular instance to the subclass S only when the instance is being used.

Options 2–4 are somewhat similar one to another, because they all propose storing the information about the instance affiliation to the subclass in the memory. Option 1 allows us to save memory by introducing another layer of translation into the query where the occurrences of the subclass are substituted with its defining predicate call before the query is passed to the real translation into the executable Java code. The negative aspect of this option is that it could compromise the performance of the query execution in cases when the subclass defining predicate is very complex (i.e., it includes

aggregate functions over the base class neighbors). Option 1 would perhaps be the best solution in case we would restrict the definition of the subclass S only to the attributes of the superclass A. That is, however, often not the case. The subclass definition is usually based on various conditions and may consider also attributes of other classes. Nevertheless, we chose Option 1 as the first attempt to implement the subclass definition feature in our query language, because we wanted to test whether the query execution performance would indeed be compromised and to what degree. In future we are planning to test also other subclass definition feature implementation approaches.

The subclass definition feature consists of two parts – the subclass definition part and the subclass application part. To allow end-users to define subclasses we have introduced another type of sentence templates (let us call it T8) in the query language that looks like this:

T8. DEFINE SClass=AClass [x] WHERE <selection condition>

Semantics: defines SClass as a subclass of AClass whose instances satisfy the selection condition.

After the subclass is defined it is accessible to end-users just like any other class of the ontology. When the subclass is used in the query formulation all its occurrences are wisely substituted with its definition before the query is passed to its translation phase. For example, one can now define the class HealthyEpisode as a subclass of the class HospitalEpisode:

DEFINE HealthyEpisode=HospitalEpisode WHERE dischargeReason=healthy

Now the class HealthyEpisode can be used as a normal class to, for example, calculate the total cost of all such episodes with a healthy outcome that have lasted for more than a week:

COUNT HealthyEpisode WHERE dischargeTime-admissionTime>7d

Such a query would then be automatically translated into a full query that does not exploit the subclass HealthyEpisode:

COUNT HospitalEpisode WHERE dischargeReason=healthy AND dischargeTime-admissionTime>7d

This translation result would then be passed to the translator to Java code and processed normally. Our first observations regarding the performance of such query execution show no major loss in most cases. Of course, the subclass defining predicate is not limited in scope, and it can grow quite big in some cases thus compromising the query execution performance to a greater degree. However, our experiments show that such complex predicates are not typical in everyday work of domain experts of the medical domain. Nevertheless, we must continue to explore the subclass definition feature in more detail to obtain more precise data of its performance.

6 Concept Testing

According to the description of a significantly wide range of information that semistar ontologies could cover it was stated in Sect. 3 that the hospital data scheme and query language are potentially “readable” by the physicians. We therefore wanted to test two aspects of our concept among domain experts – (1) whether the proposed ontology and

the language are sufficiently expressive for practical tasks and (2) whether it is simple enough to be used by a motivated physician – a domain expert. To administer such a test we implemented a software prototype which allowed users writing queries in our query language and executing them upon a sample database.

Initially the expressiveness was tested for a simplified hospital data scheme. We wanted to find out whether the simplification with the aim of displaying data scheme on one A4 page in an easy-to-use and memorizable manner would not compromise meaningful data analysis. The capability of data analysis was compared against a set of performance indicators used in hospitals for local operational management and indicators measured nationally to benchmark various international quality and efficiency aspects of patient care. The analysis we performed proved a potential of the elaborated scheme to cover full data set needed for the calculation of all currently used patient- and admission-based indicators.

The expressiveness of the query language was tested when introduced for the preparation of the detailed annual administrative and clinical reports for Intensive Care Unit of Riga Children's Clinical University hospital. The language proved to be sufficiently expressive for this rather complicated task. During two years of testing and further improving, it became a working language for the involved managing staff to formulate data queries without referring to SQL programmers in most of the cases thus approaching the self-service criteria. With or without the help of a limited set of pre-written example queries users formulated various types of queries independently – queries that have a single number result, queries that highlight multiple data items for a single patient with a particular tracer and queries that create data tables with or without some calculations. Regarding calculations and tables our aim was to incorporate only the very basic mathematical and logical calculations in the language, because there is always a possibility to export a table containing all the necessary data to MS Excel or other tools specially designed for statistical analysis.

The last part of concept testing was focused on the practical teaching aspects of the language to potential end-users who were not previously involved in the process of language development or in previous analysis of hospital data in general. We performed both individual and group experiments. The potential of the language to be taught to domain experts was best demonstrated in an experiment with a random group of domain experts – experienced health professionals in M.Sc. in the Nursing studies course at the University of Latvia. Following a regular lecture in the Healthcare management module covering general aspects of data management in healthcare, we presented elaborated simplified hospital data scheme, the querying language and the experimental web-based tool for querying data. Students were also informed that the data they will be offered for analysis, albeit anonymous, are real (our previous experience had showed that domain experts were significantly more motivated to learn querying on real data, because it often allowed satisfying natural curiosity related to previously unknown details and patterns of their everyday work). The first third of our two-hour long lecture was dedicated to the explanation of the simplified data ontology (the semantics of such concepts as 'class', 'attribute', 'classifier', 'cardinality', etc.). The rest of the lecture was devoted to an example-based explanation of the language. We chose the teaching approach based on the principles of natural language acquisition, because we had already found in early

stages of elaboration of the language that the technical explanation of the language syntax is too cumbersome and uninspiring for potential end-users who are busy health professionals.

To explain the language we started with very simple examples like asking the number of patients treated at hospital with a following inquiry of the number of treatment episodes. Since there are definitely cases of patients treated at hospital for more than one time in the given year, technically correct queries give different results. In such a way we demonstrated the ambiguity of a natural language and the importance of formulating questions precisely already in the natural language, so that they can be used to construct the query afterwards. The next question regarding the number of patients treated more than once allowed us to introduce the important notion of existence (number of 'patients' for whom there 'exists' at least one 'episode') in an easy manner. Following the introduction of each new construction it was strengthened with a set of related examples. For example, we asked the number of patients having at least one episode with a registered treatment in the particular treatment ward or the number of episodes with the patient being treated in the same ward (additionally counting also cases when the same patient was admitted several times during the given year), or finally the number of treatment cases in the same ward (additionally counting the cases when the patient had changed wards during one episode and was admitted in the particular ward more than once). In a similar way we advanced from basic constructions to more sophisticated ones while introducing other concepts. To mention just few of many examples used, a list follows here of some of the used inquiries: the number of episodes started between 5 PM and the midnight (including); the workload of a particular department (the total duration of all treatment cases in that ward); the number of episodes when the patient was referred to a hospital by relevant family doctor; the number of unique patients referred to a hospital; the number of episodes when the attending physician was the particular person; the number of patients admitted (in fact – the episodes initiated) in the particular day; the number of hospital and outpatient episodes in the particular month; the number of treatment episodes longer than the particular number of days; the number of patients having the defined age range at the time of admission; the number of patients having the treatment costs above the defined limit. The introductory lecture also included several examples used to generate a list of values for some attributes according to the defined selection (e.g. the three most common main diagnoses (codes) for patients younger than 5 years), or an extended list (e.g. the most expensive manipulation performed to patients younger than 5 years showing costs and manipulation codes).

Following the lecture we gave students homework in order to test their level of understanding. In its first part the students were asked to translate the set of pre-written queries into a good natural language. Its second part was the opposite – to rewrite natural language sentences into the formal query language. Students were advised to complete homework as soon as possible. As the result in the group of 15 students there were 9 students who did the reading task and 3 students who also did the writing task of the correctness level of at least 90%. We consider this result inspirational, because it shows that a significant number of health professionals previously not being involved in healthcare data analysis acquired important elements of the proposed querying language relatively easily. This confirms the need for further research in the use and development of

a controlled natural language for querying data by domain experts at least within the healthcare domain.

7 Conclusions and Future Work

In this paper we have proposed an easy-to-learn data querying language based on the principles of a natural language. Health professionals admit that clinical governance can only be effective if three components are integrated together – the financial control, the service performance and the clinical quality. In such a case the clinicians would be engaged, thus generating the service improvements [20]. To establish such an engagement it is also very important to understand how the data underlying the clinical process are collected and integrated, as well as what are the meanings of the numerous data elements. Self-regulation ability of health professionals would greatly benefit from the data querying approach proposed in this paper. This is in its turn necessary to optimize both technical and social aspects of the system and to move towards the goal of a more effective hospital.

We believe our proposed query language is much easier to use than various traditional query languages that are based on data located in the databases (like the SQL language). We have tested our approach by performing practical experiments which showed that the domain experts in the healthcare domain are able to learn the language quite rapidly and to use it to extract the necessary information from the available data stored in a form of a semistar ontology.

Another big topic that we plan to address in future is the mechanism of access rights. Data located in ontologies of medical domain are very sensitive and therefore various types of users are allowed to access different parts of these data. We are planning to introduce the notion of a role in the language and to supplement the language with some constructs for defining the subclass of data slices that would be accessible to a particular role.

The query language described in this paper is based on a rather controlled natural language. Our future goals therefore also include reducing the level of control by allowing users to express their needs in more informal manner. That way the query language would become even more usable for healthcare professionals. Our aim here is to allow formulating queries by providing only some basic keywords which our system would then recognize and try to generate the most probable queries in the language proposed in this paper. Since it has been proved in our end-user experiments that our language is very readable, we hope that the user would then be able to choose among the proposed queries and affirm the one he/she has actually meant. This would be the next logical step taken towards even more user-friendly natural language-based query language.

Acknowledgements. This work is supported by the ERDF PostDoc Latvia project Nr. 1.1.1.2/16/I/001 under agreement Nr. 1.1.1.2/VIAA/1/16/218 “User Experience-Based Generation of Ad-hoc Queries From Arbitrary Keywords-Containing Text” and the joint project of University of Latvia and Centre for Disease Prevention and Control “Towards a public monitoring system for the quality and efficiency of health care” under agreement Nr. ZD2017/20443.

References

1. Friedman, A.: Responsible autonomy versus direct control over the labour process. Cap. Cl. **1**(1), 43–57 (1977)
2. Degeling, P., Maxwell, S., Kennedy, J., Coyle, B.: Medicine, management, and modernisation: a "danse macabre"? BMJ Br. Med. J. **326**(7390), 649–652 (2003)
3. Braithwaite, J., Runciman, W.B., Merry, A.F.: Towards safer, better healthcare: harnessing the natural properties of complex sociotechnical systems. Qual. Saf. Health Care **18**(1), 37–41 (2009). https://doi.org/10.1136/qshc.2007.023317
4. Zviedris, M., Barzdins, G.: ViziQuer: a tool to explore and query SPARQL endpoints. In: Antoniou, G., et al. (eds.) ESWC 2011. LNCS, vol. 6644, pp. 441–445. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21064-8_31
5. Aspin, A.: Self-service business intelligence. In: High Impact Data Visualization with Power View, Power Map, and Power BI, pp. 1–18. Apress, Berkeley (2014)
6. IBM: Watson Analytics (2017). https://www.ibm.com/watson-analytics
7. Androutsopoulos, I., Ritchie, G.D., Thanisch, P.: Natural language interfaces to databases – an introduction. Nat. Lang. Eng. **1**(1), 29–81 (1995). https://doi.org/10.1017/S135132490000005X
8. Li, F., Jagadish, H.V.: Constructing an interactive natural language interface for relational databases. J. Proc. VLDB Endow. **8**(1), 73–84 (2014)
9. Llopis, M., Ferrández, A.: How to make a natural language interface to query databases accessible to everyone: an example. Comput. Stand. Interfaces **35**(5), 470–481 (2013)
10. Papadakis, N., Kefalas, P., Stilianakakis, M.: A tool for access to relational databases in natural language. Expert Syst. Appl. **38**, 7894–7900 (2011)
11. Popescu, A.M., Armanasu, A., Etzioni, O., Ko, D., Yates, A.: Modern natural language interfaces to databases: Composing statistical parsing with semantic tractability. In: Proceedings of the 20th International Conference on Computational Linguistics, COLING 2004, article no. 141 (2004)
12. Vos, L., Chalmers, S.E., Dückers, M.L., Groenewegen, P.P., Wagner, C., van Merode, G.G.: Towards an organisation-wide process-oriented organisation of care: a literature review. Implement. Sci. **6**(1), 8 (2011). https://doi.org/10.1186/1748-5908-6-8
13. Barzdins, J., Rencis, E., Sostaks, A.: Data ontologies and ad hoc queries: a case study. In: Haav, H.M., Kalja, A., Robal, T. (eds.) Proceedings of the 11th International Baltic Conference, Baltic DB&IS, pp. 55–66. TUT Press (2014)
14. Kasprzyk, A., Keefe, D., Smedley, D., et al.: EnsMart: a generic system for fast and flexible access to biological data. Genome Res. **14**, 160–169 (2004)
15. Zhang, J. et al.: BioMart: a data federation framework for large collaborative projects. Database J. Biol. Databases Curation (2011). http://doi.org/10.1093/database/bar038
16. Barzdins, J., Grasmanis, M., Rencis, E., Sostaks, A., Barzdins, J.: Self-service ad-hoc querying using controlled natural language. In: Arnicans, G., Arnicane, V., Borzovs, J., Niedrite, L. (eds.) DB&IS 2016. Communications in Computer and Information Science, vol. 615, pp. 18–34. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40180-5_2
17. Barzdins, J., Grasmanis, M., Rencis, E., Sostaks, A., Barzdins, J.: Ad-hoc querying of semistar data ontologies using controlled natural language. In: Databases and Information Systems IX. Frontiers in Artificial Intelligence and Applications, vol. 291, pp. 3–16. IOS Press (2016). https://doi.org/10.3233/978-1-61499-714-6-3

18. Barzdins, J., Grasmanis, M., Rencis, E., Sostaks, A., Steinsbekk, A.: Towards a more effective hospital: helping health professionals to learn from their own practice by developing an easy to use clinical processes querying language. Procedia Comput. Sci. J. **100**, 498–506 (2016). https://doi.org/10.1016/j.procs.2016.09.188. International Conference on Health and Social Care Information Systems and Technologies
19. Barzdins, J., Rencis, E., Sostaks, A.: Granular ontologies and graphical in-place querying. In: Short Paper Proceedings of the PoEM, CEUR-WS, vol. 1023, pp. 136–145 (2013)
20. Smith, L.F.P., Shepperd, J.: Making clinical governance work for you. Br. Med. J. **322**(7302), 1608 (2001)