Intuitive Querying of e-Health Data Repositories

Catalina Hallett, Richard Power, Donia Scott
Computing Reseacrh Centre
The Open University
{C.Hallett, R.Power, D.Scott}@open.ac.uk

Abstract

At the centre of the Clinical e-Science Framework (CLEF) project is a repository of well organised, detailed clinical histories, encoded as data that will be available for use in clinical care and in-silico medical experiments. An integral part of the CLEF workbench is a tool to allow biomedical researchers and clinicians to query – in an intuitive way – the repository of patient data. This paper describes the CLEF query editing interface, which makes use of natural language generation techniques in order to alleviate some of the problems generally faced by natural language and graphical query interfaces. The query interface also incorporates an answer renderer that dynamically generates responses in both natural language text and graphics.

1 Background

The Clinical e-Science Framework (CLEF) aims at providing a data repository of well organised clinical histories, which can be queried and summarised both for biomedical research and clinical care. In this context, the aim of the query interface is to provide efficient access to aggregated data for performing a variety of tasks, e.g., assisting in diagnosis or treatment, identifying patterns in treatment, selecting subjects for clinical trials, monitoring the participants in clinical trials. The intended users of this service are clinicians, biomedical researchers, and hospital administrators. Our current domain is cancer; however, the framework in principle supports a wide range of clinical fields.

An analysis of free text queries written by medical professionals show that they are mostly very complex and often ambiguous. This makes the design of the query interface to the CLEF repository particularly difficult, since our users will need to construct complex queries containing conditional and temporal structures.

The CLEF repository of clinical histories currently contains some 20000 records of cancer patients, includes codes such as SNOMED or ICD, and is implemented as a relational database that stores patient records modeled on the archetype for cancer developed at UCL (Kalra et al., 2001). Accessing relational databases involves expressing queries in a language that is understood by the database management system (typically SQL). Direct SQL querying requires specialist knowledge of the both the query language and the structure of the underlying database, and – in the case of medical databases – usually also knowledge of precise medical terminology codes. It clearly would be counter-productive to require this additional level of technical expertise of the clinicians and biomedical researchers who want to access the CLEF repository.

Attempts to overcome this problem in user interfaces to medical databases have traditionally made use of graphical devices such as forms, diagrams, menus, or pointers to communicate to the user the information content of a database (e.g., KNAVE (Shahar and Cheng, 1999) and TrialDB (Deshpande et al., 2001)), and research shows that they are much preferred over textual query languages such as SQL, especially by casual and non-expert users. Nevertheless, empirical studies have reported high error rates by domain experts using graphical modelling tools (Kim, 1990) and a clear advantage of text over graphics for understanding nested conditional structures (Petre, 1995).

However, it is also well-known that queries expressed in free natural language are sensitive

to errors of composition (misspellings, ungrammaticalities) or processing (at the lexical, syntactic or semantic level). A further drawback of natural language interfaces to databases is that such systems normally understand only a subset of natural language, and it is not always clear to casual users which are the valid constructions and whether the lack of response from the system is due to the unavailability of an answer or to an unaccepted input construction. On the positive side, natural language is far more expressive than SQL, so it is generally easier to ask complex questions and manipulate temporal constructions using natural language than using a database language.

2 The CLEF query interface

The CLEF query system is designed to answer questions relating to patterns in medical histories over sets of patients in the data repository. The current interface is designed for casual and moderate users who are familiar with the semantic domain of the repository but not with its technical implementation (e.g., clinicians, medical researchers and hospital administrators). For the reasons we described above, the guiding principle in the design of our interface is that its use requires no prior knowledge of the structure of the repository, no expertise in database access languages such as SQL, no familiarity with medical codes, and only minimal prior training. Users' interaction with the CLEF repository is *not* through SQL, or graphics or free text. Instead, query-construction is performed by interacting with an automatically-generated Natural Language feedback text (currently only English). This interaction method, based on the WYSIWYM technology developed by Power et al (Power et al., 1998), allows users of the profile described above to construct in an intuitive way, unambiguous, syntactically correct, complex natural language queries, such as:

```
(1)  What is the average number
of body scans performed in the
first 10 years after initial
diagnosis on individual patients
with adenocarcinoma with squamous
metaplasia who lived more than 10
years but less than 15 years after
the initial diagnosis?
```

3 Query analysis

3.1 Types of queries

An analysis of real queries from clinical trials and invented queries supplied by clinicians identified two general types of queries, as exemplified below.

```
(2)  For all patients with cancer of
the pancreas, compare the percentage
alive at 5 years for those who had a
course of gemcitabine with those who
didn't.
(3)  What is the average number of
body scans performed in the first
10 years after initial diagnosis
on individual patients with breast
cancer who lived more than 10 years
but less than 15 years after the
initial diagnosis?
```

In the first example, the expected answer is a comparison between a certain statistical measure (in this case, percentage) applied on two groups of patients differentiated by the treatment they received. The second example concerns a statistical measure (average) computed for a certain parameter (number of investigations of type "body scan") of a group of patients with some characteristics.

For either of these queries, the attributes involved in constructing the query can vary within a certain range: any statistical measure can be used, the differentiating parameter could be the diagnosis instead of the treatment, etc.

Additionally, there are a number of variations to these two main types of queries. For both types, the user may ask for simple assessment queries instead of comparisons:

```
(4)  For all patients with cancer of
the pancreas, what is the percentage
alive at 5 years for those who had a
course of gemcitabine?
```

There are also cases where several similar queries are combined into one more complex query:

```
(5) For all patients with cancer of
the pancreas, compare the percentage
alive at 1, 2 and 5 years for those
who had a course of gemcitabine with
those who didn't.
```

For all these queries, there is practically no limit to the complexity that can be achieved by using boolean operations. Each diagnosis description can in fact be a conjunction or disjunction of diagnoses, and the same applies for every concept included in a query. Therefore, the user can construct queries such as:

```
(6)  For all patients with cancer
of the pancreas or of the liver,
compare the percentage alive at 5
years for those who had a course of
gemcitabine and arimidex with those
```

who didn't.

The construction of complex queries is supported by the query editor, and they are not considered separate types of queries, nor extensions of the basic types.

3.2 Modeling queries

For presentation reasons, queries have to be decomposed into constituents that can be easily edited by the user. By way of exemplification, let us consider the query type (1). There are three elements to the query: the set of relevant patients, defined by a *problem*; the partition of this set according to *treatment*; and the further partition according to *outcome*, from which the percentages can be calculated. To avoid long complicated sentences, we consider a format in which these elements are presented separately:

> **Relevant subjects:** Patients with cancer of the pancreas.
>
> **Treatment profiles:** Patients who received a course of gemcitabine, compared with patients who did not.
>
> **Outcome measure:** Percentage of patients alive after 5 years.

This breakdown allows the following basic query pattern:

> **Relevant subjects:** [Some patients] with [some diagnosis].
>
> **Treatment profiles:** Patients who received [some treatment], compared with patients who did not.
>
> **Outcome measure:** Percentage of patients [with some status] [at some point in time].

Each of the bracketed elements are complex descriptions that model the concept definition in the CLEF archetype. For example, the concept *diagnosis* consists of the following obligatory and optional components: *tumour name*, *locus*, *type* (metastatic, primary, secondary) and *TNM staging code*. Each of the subcomponents can be extended through boolean operations (negation, conjunction, disjunction).

4 Query editing interface

4.1 General features

Conceptual authoring through WYSIWYM editing (Power et al., 1998) alleviates the need for expensive syntactic and semantic processing of the queries by providing the users with an interface for editing the conceptual meaning of a query instead of the surface text.

The WYSIWYM interface presents the contents of a knowledge base to the user in the form of a feedback text. In the case of query editing, the content of the knowledge base is a yet to be completed formal representation of the user's query. The interface presents the user with a natural language text that corresponds with the incomplete query and guides them towards editing a semantically consistent and complete query. In this way, the users are able to control the interpretation that the system gives to their queries. The user starts by editing a basic query frame, where concepts to be instantiated (anchors) are clickable spans of text with associated pop-up menus containing options for expanding the query. For example, one can start constructing a query that asks for a group of patients fulfilling some conditions by editing the following description:

> **Relevant subjects:** [some type of patients] [of a certain age description] diagnosed with [a certain diagnosis]
> **Treatment profile:** patients who received [some treatment]
> **Outcome:** [measure] of [patients with some status] at [some point in time] from [some index event]

Once the user selects an anchor and a new value for the concept represented by the anchor, the semantic representation of the query is updated and a new text is generated on the basis of this representation. Each anchor can be a combination of features or events of the same type, thus allowing for complex queries, with nested conditional structures to be built. Some concept instances can also be typed in manually, which is useful for numerical values or other fields with unpredictable content, such as names. This is also a way of enriching the ontology with new concepts. Figure 1 is a snapshot of the query editor with a partially constructed query.

The interface allows the execution of incomplete queries. The result of a user selection over the feedback text is treated as an intermediate query, which is sent to the DBMS. In return, the DBMS will transmit to the interface a feedback answer. At this point, the feedback answer is a set of paired values representing the number of patient records that match the query and the percentage from the total number of records. There is also a further breakdown of patient records by sex, which was considered a good discriminatory feature. For example, for an

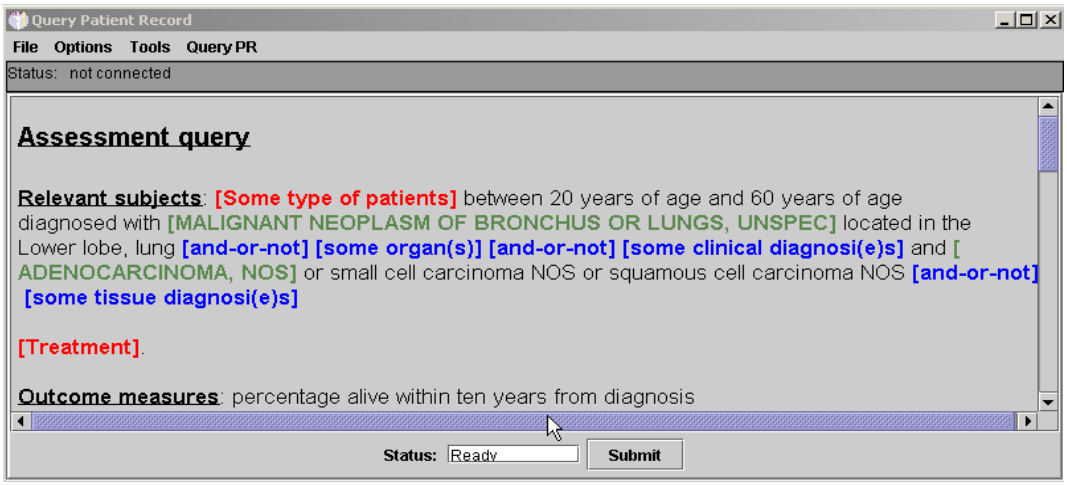

Figure 1: Query editing snapshot

intermediate query such as *Number of patients over the age of 60...*, the feedback answer could be *100 records (20% of 500), 55 men (55%), 45 women (45%).*

As a further consistency checking mechanism, the interface provides an additional rendering of the query in running text, which is performed once the editing of the feedback query has been completed, the user is presented with an alternative natural language query corresponding to the structure that has been edited (output query). While the feedback query is rather schematic to allow for more intuitive editing, the output query resembles in every respect a free text query, thus being more natural and easier to read.

The natural language interface is database-independent, since it does not require any knowledge of the database structure. The structure of the database is not only completely transparent to the user, but also to the interface developer: changes at the database level require no changes in the query editor. Queries can be saved for later re-use, which is particularly useful for frequent users who formulate queries with little variance.

4.2 Dealing with ambiguities

Since the processing of an edited query is deterministic and transparent to the user, the main challenge is not to construct valid database queries from edited queries but to ensure that the query the user is editing corresponds to the intended meaning. Therefore we want to ensure that the layout of the query conveys one meaning only to the user.

The process of defining a specific unambiguous layout for the queries was based on the analysis of some real queries that could be given multiple interpretations. Several categories of possible ambiguities are presented below, along with the solution provided by the CLEF query interface.

When the phrase describing a relevance set includes a conjunction or disjunction, there may be ambiguity over whether the intended query is single or multiple. Compare these three patterns:

```
(7)  (a).  For all patients with
lung cancer, and for all patients
with breast cancer ...
(b).  For all patients with lung
cancer and breast cancer ...
(c).  For all patients with lung
cancer or breast cancer ...
```

Example 6a is likely to be interpreted as two separate queries, while the others are ambiguous. Disjunctions like 6c occur often in real life queries:

```
(8)  For all patients younger than
60 years of age who have either
had bad prognosis myelodysplastic
syndrome only for at least 6
months or acute myelogenous
leukaemia caused by bad prognosis
myelodysplastic syndrome for at
least 6 months, what is the survival
rate...
```

In this case, it is not clear if separate answers are required for *bad prognosis myelodysplastic syndrome only* and for *acute myelogenous leukaemia caused by bad prognosis myelodysplastic syndrome*, or if it make sense to give a single answer lumping these two groups together.

This ambiguity can be avoided in WYSIWYM feedback texts by using different realisations for

conjunctions/disjunctions that imply multiple relevance sets, and conjunctions/disjunctions that do not. For example, we use bulleted lists for the former, and conjunction words (and, or) for the latter:

```
(9)  (a) Relevant subjects:
 • Patients younger than 60 years
of age who have had bad prognosis
myelodysplastic syndrome only for at
least 6 months
 • Patients younger than 60
years of age who have had acute
myelogenous leukaemia caused by bad
prognosis myelodysplastic syndrome
for at least 6 months

(b) Relevant subjects:
 • Patients younger than 60 years
of age who have either had bad
prognosis myelodysplastic syndrome
only for at least 6 months or acute
myelogenous leukaemia caused by bad
prognosis myelodysplastic syndrome
for at least 6 months
```

In 9a we have two relevance sets; in 9b we have only one.

Similar ambiguities can be found when several treatment profiles are mentioned, or several outcome measures. In each case, the ambiguity can be avoided in the WYSIWYM feedback texts the same way as before, by using bullets to mark separate queries.

Descriptions are boolean combinations of properties. A description can be elaborate either because it contains many boolean operators, or because the properties are themselves complicated. Displaying a large number of boolean combinations in running prose means that the scope of the operators can become ambiguous to the user. For this reason, layout is used to present boolean combinations more clearly:

```
(10)  Relevant subjects:
 • Patients with the following
properties:
    a.  They are younger than 60
years of age
    AND
    b.  They have one of these
properties:
       b1.  They have had bad
prognosis myelodysplastic syndrome
only for at least 6 months
       OR
       b2.  They have had acute
myelogenous leukaemia caused by bad
prognosis myelodysplastic syndrome
for at least 6 months
```

4.3 Specifying constraints and temporal relations

Guiding users towards editing correct and complete queries is essential and is one of the main points where our approach improves on classical natural language query interfaces. This is achieved by defining and implementing a system of semantic static and dynamic constraints.

Static (or ontological) constraints relate to the structure of the queries as defined in the *query model*. This includes specifying the super-class of an instance (for example, the anchor *cancer* can only be instantiated with names of cancers), its type (for example, *age* is numeric and editable, while *cancer* is a static string) and its status (compulsory vs optional).

Dynamic constraints are triggered at runtime by the user selection of certain instances. Most constraints simply serve the role of restricting the user selection so that the resulting query is meaningful and intelligible. In other cases, however, allowing the user to construct queries without implementing a system of dynamic constraints could yield ambiguous queries. Dynamic contraints can be either conceptual, which are compiled from a medical knowledge base and represent depedencies between medical concepts (for example, *nephroblastoma* is a type of kidney cancer, so users shouldn't be allowed to query for *nephroblastoma in the left breast*), or numerical (for example, *patients between 60 and 30 years of age* is a disallowed construction).

As medical records mirror the evolution in time of a patient, it is important to be able to access the patient's status at a certain point in time. The easy specification of time in natural language is an important advantage of natural language query interfaces over graphical interfaces. All temporal concepts in the medical record are stamped with a *valid time* stamp, i.e. the (precise[1]) moment in time when the event took place. Typically, a time interval is represented as a pair of *start* and *end* dates, where *start* and *end* are discrete time values of a certain predefined granularity. The query interface associates specific linguistic expressions to time intervals. For example, *between [date 1] and [date 2]* is interpreted as a closed interval [date 1, date 2], *in [this year]* is interpreted as [01/01/this year, 31/12/this year]. Such time expressions cover most temporal queries, such as: *patients diagnosed with cancer before 1999*,

[1] to a certain level of granularity imposed by the representation of time instances in the database

	Gender	Age	adenocarcinoma	small cell carcinoma	squamous cell carcinoma	death
1	female	35	true	false	false	true
2	male	41	false	true	false	true
3	male	43	false	false	true	false
4	female	53	true	false	false	false
	...					

Figure 2: Example of a result set

patients who received chemotherapy within 5 months of surgery.

5 Answer generation

A typical result set received from the DBMS consists of lists of patients that fulfilled the requirements of the query, for each patient having specified the age, gender, and the values for each of the query elements. For example, a query such as *Select all patients between the ages of 30 and 60 with a clinical diagnosis of malignant neoplasm of bronchus or lungs and histopathology diagnosis of adenocarcinoma, small cell carcinoma or squamous cell carcinoma, who were alive after 10 years of the diagnosis*, may yield the result set in Fig. 2.

The result set is processed in such a way as to allow the rendering of various groups of patients according to the age/gender breakdown and each individual query term. For each individual search parameter, the data is split into a dynamically determined number of age groups, and for each age group the number of patients is further split according to their gender. The result set thus processed is presented to the user in three types of format: tables, charts and text. Each individual chart also contains an automatically generated caption that explains the content of the chart.

The captions are generated using template-based techniques, where fillers are provided by the same result set that was used for generating the chart. For the bar chart in Fig. 3, a fragment of the explanation provided in the caption reads:
This chart displays the distribution of patients in 4 age groups according to their gender and histopathology diagnosis. 42 patients have been returned as a result to your query:
-in the 29-38 years age group there were 1 patients (0 men and 1 woman): all patients were diagnosed with adenocarcinoma. [...]
-in the 49-58 age group, there were 27 patients (14 men and 13 women): 11 were diagnosed with adenocarcinoma, 5 were diagnosed with squamous cell carcinoma, 11 were diagnosed with small cell carcinoma. [...]

6 Conclusions and further work

We have presented in this paper a query interface to a repository of patient records which makes use of natural language generation techniques. The query interface allows the editing of complex queries and is a viable alternative to natural language interfaces and visual query interfaces to medical databases. Answers to queries are provided in textual format using natural language generation techniques and also as tables and charts. The main features that set our approach apart from other querying interfaces to medical databases are:

- users require little training for using the query interface
- a set of semantic constraints are used to guide users towards constructing valid queries only, therefore incorrect queries are not possible
- the constructed queries are unambiguous, since ambiguity is dealt with in the editing stages. Therefore, one can always be sure that no errors have occurred in the interpretation of the query
- the query interface has wider applicability then strictly for accessing the current database, since it is mainly database independent

Whilst the query editing interface is fully implemented, extending the range of queries supported is an ongoing effort. This is performed in parallel with an evaluation of the usability and user-friendliness of the interface. It is expected that the evaluation will help formulate an extended range of queries and improve the editing interface. The improved query interface will provide means of interactively defining default values for instances that support them (for example, one may want to default all index events to the date of the first diagnosis). We also plan to extend the range of temporal operators to include, for example, trend operators for clinical

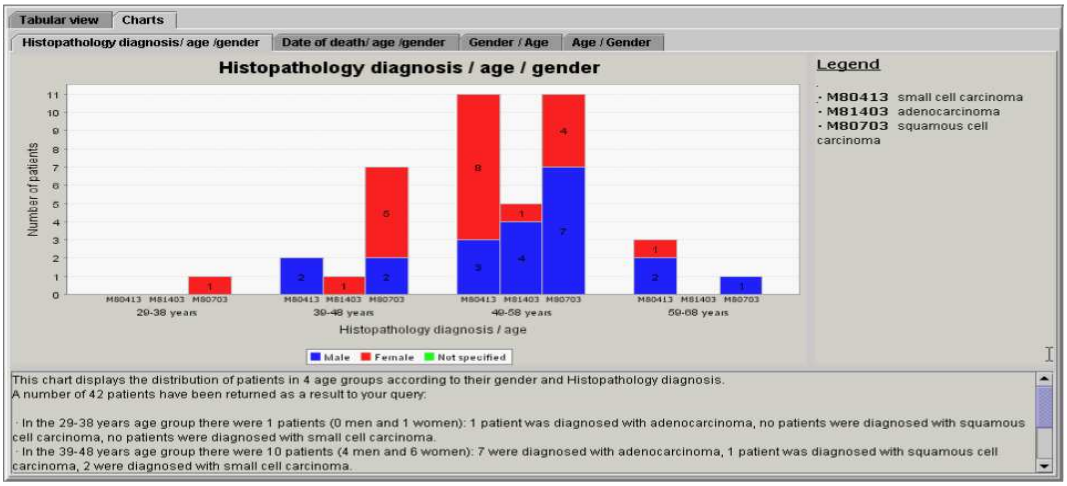

Figure 3: Generated bar chart: histopathology diagnosis/age/gender breakdown

measures that support them (e.g. *ascending blood pressure*, *stationary haemoglobin count*) and define independent variables for reporting statistical results (such as age groups, sex, education level).

References

A. Deshpande, C. Brandt, and P. Nadkarni. 2001. Ad hoc query of patient data: Meeting the needs of clinical studies. *Journal of the American Medical Informatics Association*, 9(4):369–382.

Dipak Kalra, Anthony Austin, A. O'Connor, D. Patterson, David Lloyd, and David Ingram, 2001. *Design and Implementation of a Federated Health Record Server*, pages 1–13. Medical Records Institute for the Centre for Advancement of Electronic Records Ltd.

Y. Kim. 1990. *Effects of conceptual data modelling fomalsms on user validation and analyst modelling of information requirements*. Ph.D. thesis, University of Minnesota.

M. Petre. 1995. Why looking isn't always seeing: readership skills and graphical programming. *Communications of the ACM*, 38(6):33–44.

Richard Power and Donia Scott. 1998. Multilingual authoring using feedback texts. In *Proceedings of 17th International Conference on Computational Linguistics and 36th Annual Meeting of the Association for Computational Linguistics (COLING-ACL 98)*, pages 1053–1059, Montreal, Canada.

Yuval Shahar and Cleve Cheng. 1999. Intelligent visualization and exploration of time-oriented clinical data. In *Proceedings of HICSS*, Maui, Hawaii.