Composing Questions through Conceptual Authoring

Catalina Hallett*
The Open University

Richard Power*
The Open University

Donia Scott*
The Open University

This article describes a method for composing fluent and complex natural language questions, while avoiding the standard pitfalls of free text queries. The method, based on Conceptual Authoring, is targeted at question-answering systems where reliability and transparency are critical, and where users cannot be expected to undergo extensive training in question composition. This scenario is found in most corporate domains, especially in applications that are risk-averse. We present a proof-of-concept system we have developed: a question-answering interface to a large repository of medical histories in the area of cancer. We show that the method allows users to successfully and reliably compose complex queries with minimal training.

1. Introduction

Where early attempts to build natural language question-answering systems focused on accessing and presenting information held in (closed domain) databases (e.g., Hendrix et al. 1978; Templeton and Burger 1983; Kaplan 1984; Hafner and Godden 1985), the advent of the World Wide Web has led to a shift towards (open domain) collections of texts. However, despite significant advances in open domain question answering since the simple pattern-matching systems of the first TREC competition in 1999, current systems are still largely restricted to simple questions. They can, for example, successfully find answers to questions like *Which is the highest peak in Africa?* or *Who first climbed Kilimanjaro?* but they cannot correctly answer more complex questions like:

What is the median height of the top twelve highest peaks in Africa?

Which explorer who climbed Kilimanjaro but not Everest between 1960 and 1995 died in the last three years before the age of 55?

How many of the explorers who climbed Kilimanjaro but not Everest between 1960 and 1995 did so more than three times during that period?

* Department of Computing, The Open University, Walton Hall, Milton Keynes, Buckinghamshire, MK7 6AA, UK. E-mail: D.Scott@open.ac.uk; C.Hallett@open.ac.uk; R.Power@open.ac.uk.
The research presented here was supported in part by Medical Research Council grant G0100852 under the e-Science GRID Initiative. Special thanks are due our colleagues on CLEF (Alan Rector, Jeremy Rogers, and James McKay) and to the CLEF clinical collaborators at the Royal Marsden and Royal Free hospitals—see `www.clinical-escience.org`.

Submission received: 17 July 2005; revised submission received: 3 May 2006; accepted for publication: 28 July 2006.

There are many reasons why such queries are unlikely to be successful. For example, although the first question is very simple to interpret, a correct answer is unlikely to be available (in a retrievable form) in any individual document in the target collection. A question-answering system would thus have to first retrieve the heights of each of the top twelve highest peaks, probably from different documents, and apply some calculations to obtain their median height, and then generate a response that aggregates answers from multiple documents. The answer to the second question, on the other hand, is very simple and likely to be found in a small number of documents, but the question itself is not trivial to interpret and would require (among other things) resolving the temporal information, correctly assuming that *55* refers to age at the time of death, and interpreting the negation *but not* as referring to the climbing of Everest only within the specified time span. For the third question, the difficulty comes from a combination of complex question and complex answer. Retrieving aggregated results from the World Wide Web also introduces issues of reliability because the sources may not all be trusted, and there is no guarantee that a different selection of sources would not yield a contrary result.

For many applications of question answering, the need for complex questions and trusted answers is paramount—for example, in the medical, legal, and financial domains, or indeed in any research area—and it is to this scenario that the work we present here applies. Our goal is to develop a general and intuitive method by which users can pose complex queries to data repositories; we are particularly concerned with scenarios where the users are domain experts (i.e., clinicians, lawyers, financiers, etc.) rather than database experts, where reliability of the answer is critical, where the method of posing questions should be easy to learn, and where the questions themselves should be transparent (i.e., clear and unambiguous) to both user and system.

Current methods for querying databases typically make use of formal query languages such as SQL. These languages are highly technical and require a great deal of training to achieve the level of proficiency required to pose the kinds of complex queries shown in the previous example. Successful query composition requires the user to be proficient in the query language and have detailed knowledge of the structure of the database to which the queries are being addressed. Users also need to be fluent in any formal codes employed to refer to entities in the domain (e.g., disease classifications, laws, bank codes). For example, in the medical domain alone there are a large number of clinical terminologies and classifications, used for different purposes: Some classifications, such as ICD-9, ICD-10, and OPCS-4, are employed in summarizing the incidence of diseases and operations on a national or worldwide level; others, such as CPT4 or ICD-9CM, manage the process of billing patients. Each covers a large number of terms and associated codes: SNOMED-CT alone, to name the most widely used medical terminology, currently contains some 365,000 individual concepts, and is being updated continuously (College of American Pathologists 2004). Finally, because database languages are not transparent, mistakes in query formulation can be difficult to spot; so even where the system itself may be highly reliable, there is a reasonable chance that—except for very highly experienced database programmers—the returned answer may not be an accurate response to the *intended* question.

A well-known alternative to formal database languages is available in visual query systems, which make use of graphical devices such as forms, diagrams, menus, and pointers to communicate the content of a database to the user. They are also widely used in commercial applications, and research shows that they are much preferred over textual query languages like SQL, especially by casual and non-expert users (Capindale and Crawford 1990; Bell and Rowe 1992; Catarci and Santucci 1995). However, visual interfaces are also problematic: empirical studies report high error rates by domain experts using visual object-oriented modeling tools (Kim 1990), and a clear advantage of text over graphics for understanding nested conditional structures (Petre 1995).

Natural language clearly provides a more intuitive means for users to pose their questions, but this is also highly problematic because queries expressed in free natural language are obviously very sensitive to errors of composition (e.g., misspellings, ungrammaticalities) or processing (at the lexical, syntactic, or semantic level).

2. Natural Language Interfaces

In a typical natural language interface to a database (henceforth NLIDB), the user requests database records through a query expressed in natural language. The question is first parsed and analyzed semantically by a linguistic front-end, which translates it into an intermediate meaning representation language (typically, some form of logic). The intermediate language expression is then translated into a database language (usually SQL) that is supported by the underlying database management system.

A large number of NLIDBs have been developed in the past 30 years, featuring a wide range of techniques. The general drawback of these systems[1] is that they normally understand only a subset of natural language. Casual users cannot always discern which constructions are valid or 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 formal database query languages such as SQL, so it is generally easier to ask complex questions using natural language (NL) than a database language (a single natural language query will have to be translated into multiple SQL statements). Natural language queries are not only more user-friendly for the non-expert user, but they also allow easier manipulation of temporal constructions.

Broadly, research in NLIDBs has addressed the following issues:[2]

- domain knowledge acquisition (Frank et al. 2005)
- interpretation of the NL input query, including parsing and semantic disambiguation, semantic interpretation, and transformation of the query to an intermediate logical form (Hendrix et al. 1978; Zhang et al. 1999; Tang and Mooney 2001; Popescu, Etzioni, and Kautz 2003; Kate, Wong, and Mooney 2005)
- translation to a database query language (Lowden et al. 1991; Androutsopoulos 1992)
- portability (Templeton and Burger 1983; Kaplan 1984; Hafner and Godden 1985; Androutsopoulos, Ritchie, and Thanitsch 1993; Popescu, Etzioni, and Kautz 2003)

1 Leaving aside here the possibility of errors in parsing and interpretation.

2 The extent of NLIDBs research is such that it is beyond the scope of this article to reference a comprehensive list of projects in this area. For a critical review of various NLIDBs, the reader is referred to Androutsopoulos, Ritchie, and Thanisch (1995).

In order to recover from errors in any of these steps, most advanced NLIDB systems also incorporate some sort of cooperative user feedback module that will inform users when the system cannot construct their query, and ask for clarification.

2.1 Our Solution: A Quasi-NL Interface

The solution that we propose partially overlaps with previous research in NLIDBs, in that a logical representation is constructed using a NL interface, and then mapped into the database query language. The difference lies in the nature of the NL interface, which in our case uses a method that we call **Conceptual Authoring**; this replaces the traditional method of free text entry followed by automatic interpretation.

There are two key ideas to Conceptual Authoring. The first (captured by 'Conceptual') is that all editing operations are defined directly on an underlying logical representation, governed by a predefined ontology. Instead of typing in text, the user builds the logical representation directly, so no problem of interpretation arises. The second key idea (captured by 'Authoring') is that the user interface presents the developing logical representation, and the options for editing it, in a way that is transparent to users—namely, natural language text, possibly supplemented by other familiar media; users therefore feel that they are performing a familiar activity, a kind of guided writing, rather than an unfamiliar activity akin to programming.

In general, then, Conceptual Authoring requires that some kind of formal knowledge encoding is edited by direct manipulation of a familiar presentation, the presentation being generated automatically from the underlying knowledge encoding, and updated every time knowledge is added (or removed) through an editing operation. The user need not be aware of the underlying formalism any more than a person using a text editor need be aware of ASCII codes. Conceptual Authoring therefore depends entirely on language *generation* technology; it does not use language *interpretation* at all.

Various applications of Conceptual Authoring are possible, depending on the nature of the underlying knowledge and the presentational medium. In the query editor described in this article, the underlying knowledge is a set of assertions (i.e., an A-box), and the presentational medium is natural language text. Elsewhere, we have used the term WYSIWYM (What You See Is What You Meant) for various systems of this kind that we have developed (Power and Scott 1998): as well as query interfaces they include programs that generate technical documentation in multiple languages. We use 'Conceptual Authoring' as a more general term that would also cover applications in which the underlying knowledge included conceptual definitions and rules as well as assertions, and the presentation medium included diagrams as well as text—provided, of course, that the diagrams were familiar to the relevant subject-matter experts (e.g., a molecular structure diagram if the user were an organic chemist).

The basic idea of Conceptual Authoring is that a special kind of natural language text is generated in order to present successive states of the underlying logical representation. This text includes generic phrases, called 'place-holders', which mark attributes that currently have no value. Place-holders serve as the locations where new objects may be added. By opening a pop-up menu on a place-holder, the user obtains a list of short (generated) phrases describing the types of objects that are permissible values of the attribute; when one of these options is selected, a new object of the specified type is added. New text is then generated to present the modified logical representation, including the attributes of the new object. As more information is added about an object, it will be presented by longer spans of text, comprising sentences or perhaps even paragraphs. These spans of text are also mouse-sensitive, so that the associated semantic material can be cut or copied. The cutting operation removes the logical fragment that was previously the value of an attribute, and stores it in a buffer, where it remains available for pasting into another suitable location. The text associated with the fragment may or may not remain the same, depending on the context of the new location.

As an illustration, suppose that the user wishes to define an event that might naturally be expressed by the sentence *The doctor examined the patient with a stethoscope*. The underlying logical structure could be an event object of type `examined`, with attributes for `actor`, `actee`, and `instrument`; this will of course be only one among many events allowed by the ontology. To define this content using a Conceptual Authoring interface, the user begins from a text containing a place-holder for any kind of event. By clicking on this place-holder, the user obtains a list of event patterns, each shown as a short phrase corresponding to a specific event type from the ontology:

FEEDBACK TEXT
[Some event]

MENU OF OPTIONS
.....
consulted
examined
treated
visited
.....

When the user selects *examined* from this list, an event object of type `examined` is added to the underlying semantic model, and the feedback text is regenerated to express the new event and its attributes (as yet unspecified), which are shown by short phrases in square brackets (the place-holders). A color code on place-holders indicates whether an attribute is obligatory (it must be specified) or optional (it can be left unspecified); here for convenience we use **boldface** for obligatory, and *italics* for optional. By clicking on a place-holder, say the first, the user can now obtain options for specifying the corresponding attribute (in this case the `actor`).

FEEDBACK TEXT
[Some person] examined **[some person]** *[in some way]*

MENU OF OPTIONS
.....
doctor
nurse
patient
.....

By making successive choices in this way, the user will complete the desired proposition, perhaps through the following sequence:

[Some event].

[Some person] examined **[some person]** *[in some way]*.

The doctor examined **[some person]** *[in some way]*.

The doctor examined the patient *[in some way]*.

The doctor examined the patient by using a stethoscope.

Note that because the feedback text is always generated by the system, we can try to design feedback texts in a way that minimizes ambiguity. We might, for instance, prefer to avoid the more natural phrase 'with a stethoscope', which introduces the well-known PP-attachment ambiguity, in favor of the slightly clumsy but unambiguous alternative employed herein.

As well as introducing new objects on place-holders, the user can select a span representing a filled slot and perform `Cut` or `Copy`. For instance, from the feedback sentence reached in the last example, the user could select the span 'the patient' and choose `Cut`, thus emptying the slot and reinstalling the place-holder:

The doctor examined **[some person]** by using a stethoscope.

Having freed up the slot in this way, the user might next select 'The doctor', choose `Copy`, select the place-holder '[some person]', and choose `Paste`. The `Copy` operation here applies to the actual instance selected: It does not create a new instance of the same type. Therefore, after the `Paste` operation, the doctor instance fills two slots, both the `actor` and the `actee` of the event. The resulting coreference is shown by the wording of the feedback text:

The doctor examined himself by using a stethoscope.

Conceptual Authoring (or WYSIWYM) has been applied as a tool for creating knowledge content for multilingual generation of instruction manuals (Power and Scott 1998; Power, Scott, and Evans 1998; Scott, Power, and Evans 1998) and pharmaceutical leaflets (Bouayad-Agha et al. 2002). It has also been applied in a tool for posing queries to a knowledge base of legal and regulatory information about maritime shipping (Piwek et al. 2000; Piwek 2002; Evans et al. in press). In some ways the interface resembles early menu-based techniques like Tennant, Ross, and Thompson (1983) and Mueckstein (1985); however, this resemblance is only superficial, because in these techniques the user edits a linguistic structure, whereas in Conceptual Authoring all editing operations are defined on an underlying logical structure.

3. A Test Application: Electronic Health Records

Typically, an individual's medical record is a collection of documents held in his or her doctor's office; most people will also have other records held at other sites, such as hospitals or clinics they have attended, or specialists they have seen. These records are primarily textual, and the record of the average hospital patient will consist of a large number of documents—around 100 narratives, plus hundreds of items of structured data derived from laboratory, pharmacy, or other hospital subsystems. There is a significant move—not just by medical providers, but by governments (e.g., the National Programme for Information Technology in Medicine [NPfIT] in the UK, and various e-Health initiatives in other countries)—to replace or supplement the current form of patient records with electronic records; these are intended to be not simply electronic text files of the existing records, but collections of 'messages' held in databases and accessible at the point of care. One of the disadvantages of text-only health records is that the information contained within them, because it is 'locked into' the text, is not available for statistical manipulation and cannot be easily interrogated.

Previous studies (Gorman and Helfand 1995; Ely et al. 2000; Jerome et al. 2001; Estrella et al. 2004; Koonce, Giuse, and Todd 2004), as well as our own preliminary analysis, show that free text queries written by medical professionals are mostly complex and often highly ambiguous. From this we conclude that when querying medical databases, such users need to be able to construct queries that are complex, both in volume of material and in the organization of this material (e.g., into temporal or conditional constructions). Traditionally, user interfaces to medical databases have been complex *visual* interfaces that are unsuitable for use by a casual user (Nadkarni and Brandt 1998; Shahar and Cheng 1999).

Electronic health records provide a good example of the kind of application for which question-answering systems are required for accessing large collections of trusted closed-domain data. Not only is there a requirement for complex queries of the sort that are extremely difficult to achieve in current natural language question-answering systems but, for obvious reasons, the veracity of the results of any query is critical, making it doubly important that queries put to the system, and their resulting responses, are unambiguous and clearly understandable to the user. Because the users will be medical professionals, with great demands on their time, the ease of use of the question-answering system is also extremely important. We have applied our Conceptual Authoring question-answering method to one such application: the Clinical E-Science Framework (CLEF).

3.1 The Clinical E-Science Framework

The Clinical E-Science Framework (CLEF) aims at providing a repository of well-organized clinical histories that can be queried and summarized both for biomedical research and clinical care (Rector et al. 2003). In this context, the purpose of the query interface is to provide efficient access to aggregated data for performing a variety of tasks: assisting in diagnosis or treatment, identifying patterns in treatment, selecting subjects for clinical trials, and monitoring the participants in clinical trials. Although the CLEF architecture is largely independent of any particular area of medicine, it is currently being applied to cancer, in collaboration with the Royal Marsden Hospital in London, one of the primary centers for the treatment of cancer in Britain.

The current CLEF database repository contains 22,500 patient records,[3] containing a total of more than 400,000 database entries, some 3.5 million record components

3 At present, the repository contains records of deceased patients only. In the near future, it will grow significantly with the addition of live patient records.

and more than 5 gigabytes of data, implemented as a relational database that stores patient records modeled on an archetype for cancer developed by Kalra et al. (2001). The information on each patient comes from hundreds of documents, and a single care episode or clinical problem is likely to be mentioned repeatedly in several documents. Within CLEF, a patient record is organized as a collection of individual entries, each entry representing an instance of the cancer archetype.

3.2 Users and Extent

The CLEF query system is designed to answer questions relating to patterns in medical histories over sets of patients in the repository. At this point, the system supports attribute-centric queries asking for aggregated results, such as:

{*Absolute count/percentage/statistical measure*} *of patients with certain characteristics.*

The answers to such queries can be produced by simple interrogation of the database, because they do not require inferences over the repository of patient records. However, the query interface is also coupled with a data-mining module to provide answers to more complex queries, such as

Given certain conditions, what is the treatment with the highest chance of success for a patient with certain characteristics?

The query interface can also be used for accessing information about individual patients.

The interface is designed for casual and moderate users who are familiar with the semantic domain of the repository (but not with its actual structure or encoding) and who require queries of little variance but with relatively high structural complexity. Under this description come three primary types of users, each having a different goal in interrogating the repository:

- clinicians, who use it for assisting in diagnosis or treatment
- medical researchers, who use it for identifying patterns in treatment, selecting subjects for clinical trials, or monitoring the participants in clinical trials
- hospital administrators, who use it for collecting information about patterns of treatment, frequency of tests, hospital admissions, and so on

Among these, we expect those users with little or no knowledge of formal database languages (e.g., SQL) to be the main beneficiaries of the query interface, although in the Evaluation (Section 6) we will show that even for SQL-aware users, the query interface represents an improved alternative to standard SQL. We also target the interface at users who are unfamiliar with medical encoding schemes, such as SNOMED or ICD, or who prefer to use natural language expressions instead of medical codes.

3.3 Previous Work on Querying Clinical Databases

There are a number of query systems for clinical databases, mostly designed for formulating patient-centric queries and typically using visual interfaces. For example:

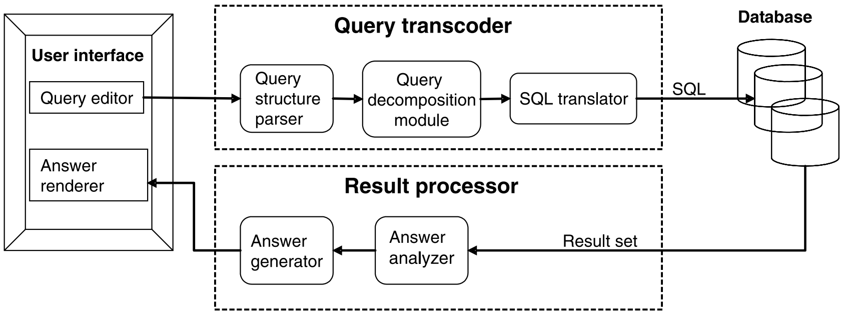

Figure 1
Architecture of the CLEF query interface.

The Columbia-Presbyterian Medical Center Query Builder works off the medical center's data repository and generates both HL7 and Arden Syntax[4] for several categories of data; for example, patient demographics, laboratory values, medications, and diagnoses (Wilcox, Hripcsak, and Chen 1997). The user interface is a simple HTML-based application that allows users to select the type of data they want to query and to specify constraints on it.

TrialDB (formerly ACT/DB) is a clinical study data management system that provides a complex visual interface for formulating attribute-centric temporal queries (Nadkarni and Brandt 1998; Deshpande, Brandt, and Nadkarni 2001, 2003). The interface allows for attributes to be searched and selected by specifying key words or part of the attribute name/caption. Once an attribute is selected, the user may optionally specify that an aggregate value for that attribute be returned. Searching criteria can be combined using boolean operators.

KNAVE is a visualization and navigation model that enables clinicians to query a specific patient record for time-oriented raw data, external interventions, abstractions, and temporal patterns, and to visualize the results of the temporal query (Shahar and Cheng 1999).

Natural language query interfaces have been used far less extensively and for more restricted tasks. For example, the HERMES system (Rivera and Cercone 1998) allows the formulation and interpretation of ad hoc queries relating to doctor and patient demographic information, patients' personal details, visit information, and insurance coverage information. It is designed as an aid to hospital administration, and not to clinical care.

4. The CLEF Query Interface

The CLEF Query Tool has four components which are invoked in sequence whenever a query is posed by the user (Figure 1). The first is the **Query Editor**, a natural language interface which guides the user in building a clear and valid query. The output of

4 HL7 is an internationally adopted communication language used for healthcare data. It covers the whole scope of healthcare communication (`http://www.hl7.org`). Arden Syntax is a standard specification of defining and sharing modular health knowledge bases, providing procedural representations of medical knowledge and explicit definitions.

this component is a logical representation underlying the query text that the user has created. The second component, the **Query Transcoder**, converts this logical representation to a Java encoding accepted by the CLEF database management system (DBMS). In this form, the query is sent to the DBMS, which recodes it again into SQL and submits it to the database. The result of the query, usually a list of records for relevant patients, returns to the third component of the Query Tool, the **Result Processor**, which transforms the raw data into an aggregated representation defining the content of the answer. This representation then passes to the fourth and final component, the **Answer Renderer**, which configures a convenient display for the user by combining fluent text with diagrams (tables and charts).

We now describe these components in more detail.

4.1 Query Editor

The Query Editor allows the user to create a logical representation of the query by means of Conceptual Authoring. When beginning a new query, the user is shown a minimally specified feedback text based on a model of query structure in this domain; this model is described in Section 5. By inserting content in the initial place-holders, the user can build up the full text of a query in a few dozen choices, a process that takes a few minutes once the user has become accustomed to the editing process (for details, see Section 6). A query is potentially complete when all obligatory slots have been filled. This is easy for the user to verify because obligatory place-holders are shown in red: When no red text remains, the query is complete. At this point, the user can hit the `Submit` button, whereupon the current A-box is passed to the Query Transcoder.

4.2 Query Transcoder

The Query Transcoder takes as input an A-box from the Query Editor, and recodes it in the format expected by the DBMS. This conversion depends on a mapping between the ontology (or T-box) employed by the Query Editor, and the concepts of the database archetype. The T-box cannot be exactly the same as the archetype, because it has to serve a different purpose—that of providing logical representations suitable for generating linguistic structures like clauses and nominals.

4.3 Result Processor

The Result Processor receives the data returned by the DBMS, normally a set of records for relevant patients, and constructs the logical representation of an answer for the user.

A typical result set received from the DBMS would list the patients that fulfilled the requirements of the query, and specify, for each patient, the features AGE and GENDER along with values for each of the query elements. For example, the query

How many patients between 30 and 70 years of age, who had a clinical diagnosis of malignant neoplasm of breast and underwent surgery, had a haematoma after surgery?

may yield the result set shown in Figure 2.

From such data, the Result Processor plans aggregate presentations in which patients are grouped according to the age/gender breakdown and the individual query terms. For each query term, the data are split into a dynamically determined number of age groups, and for each age group the patients are further subdivided by gender.

	Gender	Age	Hematoma
1	Female	32	False
2	Female	56	False
3	Male	69	False
4	Female	58	True
5	Male	59	True

Figure 2
Example of a result set.

4.4 Answer Renderer

The data thus organized are presented to the user in three types of formats: tables, charts and text. Each individual chart is accompanied by an automatically generated caption that explains its content.

Captions are generated using template-based techniques, where fillers are provided by the same data that were used for generating the chart. For example, the results we saw in Figure 2 are presented as an answer that includes the bar chart in Figure 3. This is accompanied by a textual explanation in the form of a caption, a fragment of which reads:

> Your query has returned 965 patients between 30 and 70 years of age who had a clinical diagnosis of malignant neoplasm of breast and underwent surgery. This chart displays the distribution of patients in five age groups according to their gender and time of haematoma after surgery.
>
> - In the 30–39 years age group there were 163 patients (2 men and 161 women): 151 patients did not have haematoma after surgery, 12 patients had haematoma after surgery.
> - In the 40–49 years age group there were 326 patients (no men and 326 women): 304 patients did not have haematoma after surgery, 22 patients had haematoma after surgery.
> - In the 50–59 years age group there were 363 patients (8 men and 355 women): 337 patients did not have haematoma after surgery, 26 patients had haematoma after surgery.
> - In the 60–69 years age group there were 110 patients (2 men and 108 women): 97 patients did not have haematoma after surgery, 13 patients had haematoma after surgery.
> - In the 70–79 years age group there were 3 patients (one man and two women): two patients did not have haematoma after surgery, one patient had haematoma after surgery.

5. Query Model

A controlled editing environment is most effective when based on a model of the kinds of queries that users will wish to make. There is a trade-off here between flexibility and ease of use. If we have no preconceptions about the general nature of queries, we have to provide users with a wide set of possible patterns, leaving them to search for the particular pattern they happen to want. If instead we can assume that the query will belong to a known set of patterns, the editor can help the user to get started by offering a manageable list of alternatives, so avoiding the 'blank page' problem.

Investigations on a taxonomy of queries posed by *general practitioners* in an outpatient setting has shown that in primary care, queries are relatively simple and

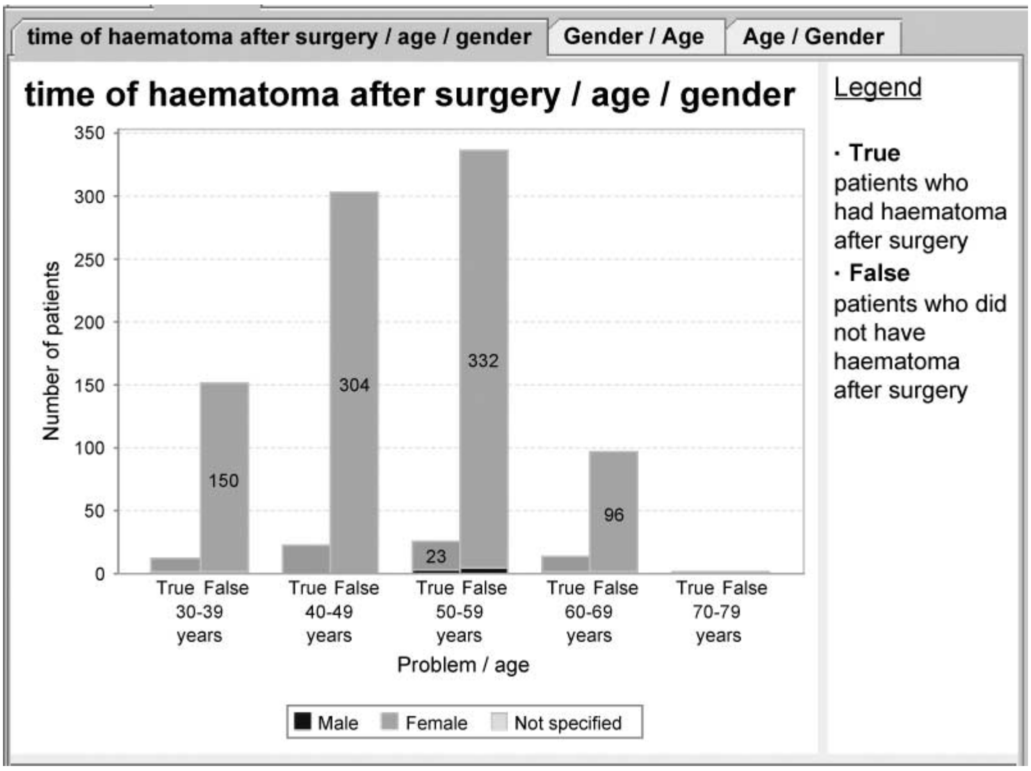

Figure 3
Chart generated as part of a response displaying the distribution of patients who developed and did not develop haematoma according to their age and gender.

generally ask for evidence-based advice for treatment decisions (Ely et al. 2000). For example, of 64 generic question types, the three most common are:

What is the drug of choice for condition X?

What is the cause of symptom X?

What test is indicated in situation X?

In contrast to these findings, our consultation with *cancer clinicians* revealed that questions posed in a clinical research setting tend to have a more complex nature and to be directed at groups of patients, searching for relationships rather than simple values:

What is the average time of relapse in Acute Myeloid Leukaemia for patients with a complete response after two cycles of treatment?

Can this time be linked to the cytogenetic findings in the original blood sample?

What is the median time between first drug treatment for metastatic breast cancer and death?

Our policy in CLEF has been to aim first at a relatively specific model, under the guidance of the relevant experts—clinicians and medical researchers in the area of cancer.

5.1 Elements of a Basic Query

The structure of a typical query (according to our experts) is shown by the following relatively simple example:

For all patients with cancer of the pancreas, what is the percentage alive at five years for those who had a course of gemcitabine?

As can be seen, this query breaks down into three elements: 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 percentage can be calculated. For maximum clarity, the Query Editor can format the query so that these three elements are marked explicitly and presented separately:

`Relevant subjects:` Patients with cancer of the pancreas.
`Treatment profile:` Patients who received a course of gemcitabine.
`Outcome measure:` Percentage of patients alive after five years.

Generalizing from this example, we can identify the following basic query pattern:

`Relevant subjects:` Patients with [some diagnosis].
`Treatment profile:` Patients who received [some treatment].
`Outcome measure:` [Measure] of patients [with some status] [at some point in time].

An important requirement on this formulation of the query is that it should be unambiguous—namely, that users should understand correctly how the outcome measure will be calculated. We assume that the calculation will proceed through the following steps. First, retrieve all the patients in the database who satisfy the conditions in the first two paragraphs (Relevant subjects and Treatment profile)—in this example, all patients with cancer of the pancreas who received a course of gemcitabine. Call this set S and let its cardinality (i.e., the number of patients in the set) be $C(S)$. Next, find the subset of S also satisfying the outcome condition—in this example, the patients still alive after five years. Call this set M and its cardinality $C(M)$. Finally, divide $C(M)$ by $C(S)$ and express the result as a percentage.

With a slight elaboration of this basic pattern, we can obtain a second kind of query, which requests a comparison rather than a single value:

For all patients with cancer of the pancreas, compare the percentage alive at five years for those who had a course of gemcitabine with those who didn't.

Again this can be presented to the user using a separate paragraph for each element:

`Relevant subjects:` Patients with cancer of the pancreas.
`Treatment profile:` Patients who received a course of gemcitabine, compared with patients who did not.
`Outcome measure:` Percentage of patients alive after five years.

For a comparison question we need to compute two outcome measures, so the steps in the calculation have to be elaborated as follows. First, retrieve two sets of patients, S_1 and S_2, satisfying the conditions that we want to compare. In the example, S_1 will be the set of patients with cancer of the pancreas who had a course of gemcitabine; S_2 will be the set of patients with the same type of cancer but no gemcitabine treatment.[5] Next, for each set, find the subset of patients still alive after five years: Call these subsets M_1 and M_2. Finally, compute the measures to be compared by dividing $C(M_1)$ by $C(S_1)$, and $C(M_2)$ by $C(S_2)$, and expressing the resulting ratios as percentages.

5.2 Complex Queries

Each element of a query can be made more complex in two ways. First, it can be replaced by a conjunction or disjunction, so that the query in a sense becomes several queries requiring several answers. Second, the content of the description can be elaborated, for example by adding more qualifications. Here is an example of the first kind:

> *For all patients with a brain glioma, what percentages are still alive at 1, 2, and 5 years if they take Imatinib Mesylate every day?*

This can be analyzed as a single relevance group, single treatment, and multiple outcome measures (survival at 1, 2, and 5 years). Separate answers for these measures will be needed. An example of the second kind is the following:

> *For all patients with cancer of the vulva that is locally advanced and/or metastatic or recurrent, and where this cannot be treated with either surgery or radiotherapy of any kind, what is the survival rate for those given Taxol only?*

We assume this is a single rather than a multiple query, and that separate answers are not needed for the various conjunctions and disjunctions. The treatment profile (Taxol) and the outcome measure (survival rate) have a content that can be easily specified—a single choice from a menu would suffice. However, the set of relevant patients requires a very elaborate description because there are so many qualifications.

5.3 Multiple Relevance Sets

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:

(1) For all patients with lung cancer, and for all patients with breast cancer ...

(2) For all patients with lung cancer and breast cancer ...

(3) For all patients with lung cancer or breast cancer ...

Here (1) seems a clear case of a multiple query, whereas the others are ambiguous but tending to a single-query interpetation. It is hard to eliminate such ambiguities altogether while wording the query in a way that is reasonably natural, but at least we can impose consistency by using different realization devices for the two cases—for

5 These sets are obviously disjoint.

example, bulleted lists for conjunctions (or disjunctions) that imply multiple queries, and discourse connectives (*and*, *or*) for ones that imply single queries. For example:

`Relevant subjects:`

- Patients younger than 60 years of age who have had bad prognosis myelodysplastic syndrome only for at least six months
- Patients younger than 60 years of age who have had acute myelogenous leukaemia caused by bad prognosis myelodysplastic syndrome for at least six months

versus

`Relevant subjects:`

- Patients younger than 60 years of age who have either had bad prognosis myelodysplastic syndrome only for at least six months or acute myelogenous leukaemia caused by bad prognosis myelodysplastic syndrome for at least six months

In the first we have two relevance sets; in the second we have only one.

5.4 Multiple Treatment Profiles

A similar ambiguity is found when several treatment profiles are mentioned. Either there are several queries, or there is a single query concerning a logical combination of the treatments. The following query text (written as an example by a medical researcher) could be interpreted either way:

For all patients younger than 60 years of age who have either had bad prognosis myelodysplastic syndrome only for at least six months or acute myelogenous leukaemia caused by bad prognosis myelodysplastic syndrome for at least six months, what is the survival rate if you give them intensified remission induction chemotherapy followed by either an autologous or allogeneic bone marrow transplant?

Perhaps the researcher's aim is to compare autologous bone marrow transplants with allogeneic ones. Alternatively, it might not matter whether the transplant is autologous or allogeneic provided that it is one or the other, as suggested by the singular verb ('what *is* the survival'). In the query interface, the ambiguity can be avoided in the same way as before, by using bullets to mark separate queries. For example:

`Treatment profiles:`

- Intensified remission induction chemotherapy followed by an autologous bone marrow transplant
- Intensified remission induction chemotherapy followed by an allogeneic bone marrow transplant

versus

`Treatment profiles:`

- Intensified remission induction chemotherapy followed by an autologous **or** allogeneic bone marrow transplant

In the first we have two treatment profiles and hence separate queries; in the second we have only one.

5.5 Multiple Outcome Measures

There are several examples in which survival rates are requested at, say, one year, two years, and five years. It makes no sense to combine these into a single query, so they are always interpreted as separate queries.

5.6 Comparison Queries

Comparison queries are those that ask for certain outcomes of separate groups of patients that do not share common diagnosis or treatment profiles.

> *Compare survival rates at 5 years after diagnosis for patients with adenocarcinoma who received chemotherapy and patients with invasive ductal carcinoma who received radiotherapy.*

We represent such queries as two independent queries, with separate profiles.

5.7 Elaborate Descriptions

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. The following description of a reference set is elaborate in both ways:

> *For all patients younger than 60 years of age who have either had bad prognosis myelodysplastic syndrome only for at least six months or acute myelogenous leukaemia caused by bad prognosis myelodysplastic syndrome for at least six months, what is the survival rate. . . ?*

Complex boolean combinations of this kind often cannot be presented in running prose without the scopes of the boolean operators becoming unclear. To avoid this problem, the feedback text generator formats complex boolean combinations using hierarchical layout:

`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 six months
    OR
    b2. They have had acute myelogenous leukaemia caused by bad prognosis myelodysplastic syndrome for at least six months

Descriptions of treatment profiles can be elaborate in the same ways. The following excerpt has two treatment profiles, the first using a combination of AND and NOT, the second using AND combined with temporal sequence (marked by THEN).

> *. . . compare the survival rates over time for those who had no surgery but did have mitomycin C injected into the bladder once a month, with those who had transurethral resection of the tumor and then a single one-time injection of mitomycin C into the bladder.*

The corresponding part of the feedback text is laid out as follows:

`Treatment profiles:`

- a. NO surgery
  AND
  b. Mitomycin C injected into the bladder once a month

- a. Transurethral resection of the tumor
  THEN
  b. A single one-time injection of mitomycin C into the bladder

5.8 Representing Time

Queries about patient records contain many references to events occurring at particular times: diagnoses, tests, treatments, deaths, and so forth. These time specifications are crucial. To deal with them effectively, the query tool must meet two requirements. First, the feedback text should express temporal relations naturally and unambiguously, using familiar devices like tenses and adverbials. Second, the resulting conceptual representation (in the A-box) must be aligned with the fields through which time is represented in the database.

Like most databases, the CLEF medical records employ several types of time stamp. First of all, an event has a **valid time**, the moment when it actually took place. For example, if a mastectomy was performed on 29th January 2000, the valid time would be some representation of this day, perhaps '29-01-2000', assuming a granularity calibrated in days (rather than hours, weeks, etc.). Next, an event has a **recording time**, the moment when it was written down. Obviously this might differ from the valid time, although they would be the same if the doctor kept prompt records. We also use a concept of **query time**, the moment when a query was formulated by the CLEF user: this is needed in order to interpret deictic time references in the feedback text, based for example on tenses or on phrases like 'after 1995', which can be interpreted to mean 'from 1995 until now'.

For some events the valid time can be a single moment, specified for example by a date. For events that last for longer intervals, like a whole course of treatment, two valid time stamps have to be given, one for the start time and one for the end time. Of course this distinction is related to granularity. With a granularity based on days, a week has to be treated as an interval with a start date and an end date; with a granularity based on weeks, the same week could be identified by a single time stamp (e.g., 'W40-2000', meaning the 40th week of the year 2000).

To model time effectively in queries, we need to provide a range of natural and clear options, and map them to the time stamps used in the database. At present, the temporal modifiers offered during query editing are as follows:

- *between [date 1] and [date 2]*: interpreted as a closed interval [date 1, date 2]
- *after [some date]*:interpreted as a closed interval [this date, query time]
- *before [some date]*
- *in [this year]*: interpreted as [01/01/this year, 31/12/this year]
- any of the above, where instead of a specific date the user enters an index event *after the surgery;* in this case, the implied time will be computed by the DBMS instead of explicitly entered by the user

- *event1* {*while/at the same time as/during event2*} : will be interpreted as two overlapping time intervals corresponding to the two events

For example, such time expressions cover queries like: *patients diagnosed with cancer before 1999*, or *patients who received chemotherapy within 5 months of surgery*. The interface allows Allen's 13 basic interval relationships to be expressed in natural language (Allen 1984).

In principle we could require users to associate a time stamp with every event mentioned in a query; however, by imposing this further requirement on users we would pay a high price in usability, virtually doubling the number of operations needed in order to complete the query, and damaging the transparency of the resulting text. In the CLEF query interface we have decided instead to associate default values to time descriptions and to make the time stamp anchors visible only on demand in the feedback text. The output text will contain all the time stamps, with the values either entered by the user or defaulted, so allowing the user to review the query and amend it where necessary.

6. Evaluation

The best evaluation of any question-answering system is one which looks at real users making information-seeking requests in real-life contexts. Because the complete CLEF system is not yet ready for deployment, this is impractical at this stage. However, we have been able to perform usability tests on the query interface in isolation from the full system, and this is what we report on here. Our current study does not cover the Query to SQL Translation and the Answer Retrieval components, which are part of the server components side of the query interface. This separation is not always possible in practice. For example, we cannot at this stage test the full range of queries that can be constructed in the interface, because some are not yet supported by the back-end. Similarly, we can only assess the time necessary for editing queries, not for retrieving answers, because this is almost entirely dependent on the communication procedure and on the speed of the SQL translator.

We have thus far conducted two formal experiments, to address the following questions:

- Are users able to successfully compose complex queries using the system?
- Can the system be used with minimal training?
- Are the queries, as presented in the interface, easily understandable?

6.1 Experiment 1: Query Composition

As mentioned earlier (Section 1), one of the main desiderata behind the design of our querying method is that it should be intuitive. With respect to the system we have implemented for CLEF, what this means is that medics and bio-informaticians should be able to pose the kind of complex queries that they require, without the need for extensive training, or for knowledge of the structure or language of the underlying repository. This experiment tests the extent to which our querying method fulfills these requirements.

Subjects. Fifteen medics and bio-informaticians participated in the experiment. All had previously been granted clearance[6] to see the information in the confidential repository of patient records. All subjects were knowledgeable in the domain of cancer, and all but two had no knowledge of the representation language of the repository (SQL), or of how the data contained therein were structured; none had any prior experience with the query-formulation interface.

Methodology. Each subject was given a short (5–10 minute) introduction to the interface, which included a demonstration of the construction of a fairly simple query. Subjects were then given a set of four queries, which they were asked to compose using the interface. To increase the difficulty of the task, the questions presented to the subjects avoided, where possible, the wording required by the user interface, so that users were obliged to think about the meaning rather than to aim for particular target phrases. To avoid effects of practice, we varied randomly the order in which the questions in the set were presented to subjects. Subjects were allowed as much time as they needed to compose each query.

For each subject, we measured the time taken to build each query, and recorded the number of operations used for constructing it.

Materials. The materials for the experiment consisted of the following set of four queries:

> *How many patients who received surgical treatment for malignant neoplasm of the central portion of the breast had no curative radiotherapy?*

> *How many patients between the ages of 40 and 60 when they were first diagnosed with lung cancer* (malignant neoplasm of bronchus or lung, unspec) *received radiotherapy and had a platelet count higher than 300 and a leukocytes count lower than 3?*

> *What percentage of patients under the age of 60 treated for breast cancer* (malignant neoplasm of breast, unspec) *died within 5 years of a mastectomy?*

> *How many patients with acute lymphoid leukaemia have been given chemotherapy?*

These are representative of the query types that emerged from an earlier requirements analysis with oncologists and cancer bio-informaticians. They also vary in their levels of structural complexity and in the number of interface operations required to successfully complete them.

As can be seen, these questions are far more complex than the queries standardly posed to search engines or to most other interactive query engines (as described, for example in [Hovy, Hermjakob, and Ravichandran 2002]; [Soricut and Brill 2004]; [TREC 2005]).

Results. The main finding of this experiment is the achievement of 100% success in subjects' ability to use the interface for the purpose for which it was intended: *All subjects successfully composed all queries*. The mean completion time per query was 3.9 minutes (noting that subjects were under no time pressure to complete the individual

6 By the UK Medical Research Ethics Committee (MREC).

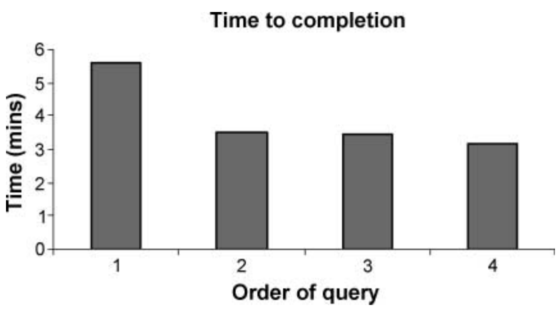

Figure 4
Mean completion time for queries in order of occurrence.

queries).[7] Figure 4, which gives the average time to completion across all subjects, shows that subjects learned to use the interface quickly: they take much longer on their first query, and their performance asymptotes by the time they get to the second query. This effect is confirmed by an analysis of variance (ANOVA)[8], which shows a highly significant effect of order of presentation ($F = 9.8427$; $p < .0001$). Furthermore, significant differences were found between subjects' performance on the first query they composed compared to the second, third, and the fourth (each at $p < .01$ on the Tukey HSD test). However, application of the same test showed no significant difference in subjects' performance on the second versus third, second versus fourth, or the third versus fourth composed query.

Because the queries vary in structural complexity, some will require the user to perform more interface actions than others, and so one would predict a difference in subjects' performance (i.e., time to completion) on the individual queries; this was borne out by the analysis (ANOVA, $F = 5.5015$; $p < .0028$).

If the method is easy to learn, one would predict that subjects' proficiency with the interface will increase fairly quickly as they move from the first query they encounter to the last, irrespective of complexity. This can be tested by measuring subjects' performance on the interface in terms of the number of interface operations (mouse clicks and selections) they perform, normalized for complexity: a value of 1 would mean that subjects perform twice as many operations as are required; a value of 0 would mean that subjects perform the minimal number of required operations (i.e., perfect performance). The result of such an analysis is shown in Figure 5. The picture that emerges from this is one where, overall, subjects are very efficient, achieving an average score of 0.19 over their first four encounters with the method. They make a fair number of false starts when composing their first query, but become extremely proficient by the time they get to their second query, and near perfect by the time they get to the fourth. Analysis of variance[9] shows a highly significant effect of order of presentation ($F = 7.4993$; $p < .0004$). Once again, the Tukey HSD Test shows a significant difference between the first query encountered and each of the subsequent ones ($p <. 01$), and that the differences between the second and third, the second and fourth, and the third and fourth, were nonsignificant.

7 For the last 5 subjects, all of whom used a version of the interface that had been improved to respond faster to interface actions, this average went down to 2.7 minutes.
8 One-way ANOVA for correlated samples.
9 One-way ANOVA for correlated samples.

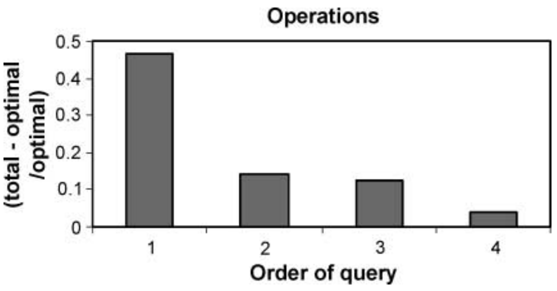

Figure 5
Proficiency with the interface as a function of experience.

6.2 Experiment 2: Clarity of the Queries

Interfaces to databases based on natural language interpretation inevitably suffer from the ambiguity and imprecision of the input texts, unless users can be trained in a controlled language. Our method of composing queries avoids this problem altogether: because the natural language feedback text is generated by the system rather than the user, there is no need for the system to choose among alternative interpretations. Of course, this does not guarantee that the query text is equally transparent to the user: this will depend on the efficacy of our feedback text design—the point we wish to evaluate in the present experiment, which explores the extent to which composed queries, as presented in the feedback texts, can be clearly understood.

Subjects. Fifteen subjects participated in the experiment. Of these, ten had previously participated in Experiment 1; the new subjects had the same profile as those previously seen.

Methodology. Subjects were given a paper-based questionnaire containing 24 trials, each showing a completed complex query as presented in the interface (i.e., as a 'feedback text'). Each query was associated with three alternative interpretations, presented as full natural language questions: only one of these represented the correct meaning; the other two represented plausible but incorrect meanings. Subjects were given a forced-choice task to identify which of the three alternatives corresponded to the meaning of the given feedback text.

The queries were presented to all subjects in the same (random) order. We devised five presentation sets, each containing a different ordering of the options for each query, and these were randomly assigned to subjects. We suggested to subjects that a useful strategy might be to read the alternatives *before* looking at the associated feedback text. There was no time limit.

Materials. The materials comprised four examples, each of six patterns of ambiguity:

Type 1: Attachment of temporal expression. Most events can have a temporal expression associated. When there is more than one event that could be subsumed by a temporal expression, the text may become ambiguous. For example:

> `Relevant subjects:` patients with a clinical diagnosis of breast cancer
> `Treatment:` patients who did not receive adjuvant chemotherapy in the past year
> `Tests:` [ ]
> `Outcome:` absolute number of patients

Options:[10]

- *How many patients diagnosed with breast cancer had no adjuvant chemotherapy in the past year?*
- How many patients treated for breast cancer in the past year had no adjuvant chemotherapy?
- How many patients diagnosed with breast cancer in the past year had no adjuvant chemotherapy?

Type 2: Scope of conjunctions. Whenever a complex expression contains a combination of conjunctions and disjunctions, potential ambiguities may occur, especially when combined with negations or prepositional phrases. For example:

`Relevant subjects:` patients with a clinical diagnosis of invasive ductal carcinoma
`Treatment:` patients who received breast conservation surgery, no auxillary surgery, and radiotherapy
`Tests:`[ ]
`Outcome:` absolute number of patients

Options:

- *How many patients diagnosed with invasive ductal carcinoma underwent breast conservation surgery, did not undergo auxillary surgery, and received radiotherapy?*
- How many patients diagnosed with invasive ductal carcinoma underwent breast conservation surgery, did not undergo auxillary surgery, and did not receive radiotherapy?
- How many patients diagnosed with invasive ductal carcinoma did not undergo breast conservation surgery, did not undergo auxillary surgery, and received radiotherapy?

Type 3: Scope of conjunctions plus attachment of temporal expression. This is an extension of the first two cases, where a temporal expression post-modifies an expression that is part of a conjunction of events. For example:

`Relevant subjects:` patients with a clinical diagnosis of malignant neoplasm, unspecified
`Treatment:` patients who received radiotherapy and chemotherapy within 1 year of the diagnosis
`Tests:`[ ]
`Outcome:`absolute number of patients

Options:

- *How many patients diagnosed with cancer had radiotherapy and chemotherapy both within 1 year of diagnosis?*
- How many patients diagnosed with cancer had radiotherapy within 1 year of diagnosis and also had chemotherapy at any time?

10 In the examples that follow, the correct interpretations are indicated with italics.

- How many patients diagnosed with cancer had radiotherapy and chemotherapy and received any kind of treatment within 1 year of diagnosis?

Type 4: Combination of various query components. Events in a query can be linked to each other by various means, including temporal expressions, conjunctions, and disjunctions. Complex combinations may render the feedback text ambiguous. For example:

> `Relevant subjects:` patients with a clinical diagnosis of breast cancer and who had nausea within 1 year of the chemotherapy
> `Treatment:` patients who received [some surgical procedure] [at some point in time] and chemotherapy but no radiotherapy within 1 year of the diagnosis
> `Tests:[ ]`
> `Outcome:` percentage of patients who were alive after 5 years of the diagnosis

Options:

- *What percentage of patients diagnosed with breast cancer who underwent a surgical procedure at any time, received chemotherapy within 1 year of the diagnosis, had nausea within 1 year of the chemotherapy, and received no radiotherapy within 1 year of the diagnosis, survived more than 5 years after diagnosis*?
- What percentage of patients diagnosed with breast cancer who underwent a surgical procedure at any time, received chemotherapy at any time, had nausea at any time after chemotherapy, and received no radiotherapy within 1 year of the diagnosis, survived more than 5 years after diagnosis?
- What percentage of patients diagnosed with breast cancer who underwent a surgical procedure at any time, received chemotherapy within 1 year of the diagnosis, had nausea after chemotherapy but within 1 year of the diagnosis, and received no radiotherapy within 1 year of the diagnosis, survived more than 5 years after diagnosis?

Type 5: Complex queries, non-ambiguous components. We introduced this category in order to test the readability of complex queries that do not necessarily contain ambiguous components. Because most queries in the medical domain are likely to be very complex, can the sheer number of query components render the query ambiguous to the users? For example:

> `Relevant subjects:` patients under the age of 50 at the time of diagnosis, with a clinical diagnosis of breast cancer
> `Treatment:` patients who received [some surgical procedure] [at some point in time] and no chemotherapy within 1 year of the diagnosis
> `Tests: [ ]`
> `Outcome:` absolute number of patients

Options

- *How many patients with breast cancer, under the age of 50, had a surgical procedure at any time and did not have chemotherapy within 1 year of the diagnosis*?

- How many patients with breast cancer, under the age of 50, had a surgical procedure within one year of the diagnosis and did not have chemotherapy within one year of the diagnosis?
- How many patients with breast cancer, below the age of 50, had a surgical procedure within one year of the diagnosis and had chemotherapy after one year of the diagnosis?

Type 6: Attachment/interpretation of outcome. The outcome section generally describes a condition holding between a reference and a target set of patients. If the query contains multiple features describing the patient set, it may be difficult to differentiate between features that contribute to the reference set and features that contribute to the target set. For example:

`Relevant subjects:` patients with a clinical diagnosis of breast cancer and who had anaemia after chemotherapy
`Treatment:` patients who received chemotherapy
`Tests:[ ]`
`Outcome:` percentage of patients who were alive after 5 years from the diagnosis

Options:

- *Of the patients diagnosed with breast cancer who developed anaemia after chemotherapy, what percentage survived 5 years after diagnosis*?
- Of the patients in the database, what percentage were diagnosed with breast cancer, developed anaemia after chemotherapy, and survived 5 years after diagnosis?
- Of the patients diagnosed with breast cancer, what percentage developed anaemia after chemotherapy and survived 5 years after diagnosis?

Results. If the presented feedback text is incomprehensible, the probability that subjects will select the correct interpretation will be 0.33 (i.e., they will get the right answer only a third of the time). Our results show that subjects' precision is 0.84; that is, *on average, they select the intended interpretation 84% of the time, rather than 33% as would be predicted if their selections were random*. Statistical analysis of these results, using a one-sample *t*-test, shows this effect to be highly significant (mean = 0.8361, d = 0.5028, t = 16.76, $p < .0001$). The breakdown by type of ambiguity is shown in Table 1.

Table 1
Interpretation of feedback texts.

Ambiguity	Total correct	Percent
Type 1	51	85
Type 2	54	90
Type 3	50	83
Type 4	48	80
Type 5	47	78
Type 6	51	85

6.3 Summarizing the Evaluation

The true test of any system comes with its use *in situ* by the users for which it was designed. Normally, this would be preceded by a bank of formal empirical studies under more controlled conditions. For a question-answering system like the one we are addressing in this article (which is but a small part of a much larger system), a formal controlled evaluation would ideally cover a large number of exemplars of each type of query supported by the system, and a large number of subjects. Given our constraints on the number of available subjects (and the concomitant effect this has on the possible design of any experiments), the evaluation reported here is necessarily more limited in scope. This is not an unusual situation in system development, where evaluation must proceed by gradual refinement through the application of rigor, wherever possible, but also applying along the way intuition, common sense, and past experience. The evaluation we have presented here shows what can be done during the early phases of the development of a large and complex system whose components are in different stages of completion, and where access to representative users is limited.

Given these caveats, the picture that emerges from this study is nonetheless very encouraging. Our results suggest that our target users (medical researchers) can quickly learn to construct queries of the type and complexity that they have identified as relevant. Specifically:

- They are able to use the Conceptual Authoring method successfully to compose complex queries, with no prior exposure to the method and with the benefit of only minimal training.
- They become quickly proficient with the system, achieving near perfect performance by their fourth attempt at query composition.

The study also indicates that the feedback texts employed to construct queries of a high degree of structural complexity are not difficult to understand. This is extremely important, as it means that users can be confident that they are obtaining an answer that pertains to the question that they *think* they are asking, as opposed to an answer to some other similar question.

Additionally, in a separate, informal study, we have found suggestive evidence that the Conceptual Authoring method of query composition may be much more user-friendly than the traditional method of direct SQL editing, even for extremely skilled SQL coders with a high level of familiarity with the database and the domain (Hallett, Scott, and Power 2006). Our tests showed that (an albeit small sample of) such experts, even in a situation that is heavily biased towards optimal performance of SQL codes, found it much easier to compose queries with the Conceptual Authoring interface than in SQL. Not only did it take them more than three times longer, on average, to compose the query in SQL, but they were not able to produce the complete SQL in that time.

7. Conclusion

Most question-answering systems make use of natural language understanding and allow users to pose simple questions to textual repositories. We have presented here a generic method for composing natural language questions within a question-answering system that avoids the well-know pitfalls of natural language understanding while allowing users to pose complex questions to data repositories. The method, Conceptual Authoring, involves no natural language *interpretation*—only *generation*—and is particularly well-suited to query interfaces to closed-domain systems. We have elucidated the method through its use in the CLEF query tool, which has been designed to meet the requirements of a particular context: the querying of large repositories of electronic health records by doctors and medical researchers. Similar requirements almost certainly apply in other fields of expertise (e.g., engineering, genomics, law, finance), as data are increasingly available in machine-usable electronic form; they can be summarized as follows:

- *Users*: The query tool must be usable by the relevant domain experts—doctors, lawyers, or whomever—with no training in database query languages.
- *Training*: The users must be able to use the query tool after minimal learning time (minutes rather than hours).
- *Time*: After training, users must be able to construct complex queries in a time comparable to writing the query down on paper—that is, a few minutes.
- *Reliability*: The query tool must be close to 100% reliable, in the sense that any query correctly formed by the user will be correctly transcoded into the database query language and therefore answered by the system.
- *Transparency*: Queries must be presented to users in a form that is clear and unambiguous, so that they know exactly what question they have asked.

Although not exactly a requirement, a practical consideration is that the queries should be frequent and important enough to justify the effort needed to meet the very stringent requirements on usability and transparency. There is no point investing in a natural language interface like the CLEF query tool except in contexts where the query results are highly valuable.

In our view, transparent communication with expert users depends first and foremost on using a familiar medium—the medium the experts use in their normal work, which in this case means natural language. However, as argued in Section 2, traditional natural language interfaces to database systems cannot meet the requirements on reliability and training, because reliable interpretation of an input text can be achieved only if the text conforms strictly to a controlled language (which our users would not have time to learn). We therefore proposed a modification of the traditional approach, in which the semantic representation of the query is edited directly, through an interactive feedback text generated by the system. Otherwise the approaches are the same: once obtained, the semantic representation is transcoded to the database query language and passed to the database management system; when the answer is returned, it is organized to suit the purposes of users, and presented in a familiar display such as a text or diagram.

Ultimately, the value of such a tool must be proved in everyday use, but our evaluation study provides some evidence that our approach can meet the requirements. First, our studies were performed with the relevant *users*, in this case medical experts. The *training* required for reaching a reliable level of performance was a matter of minutes—usually a single demonstration followed by a single trial. Thereafter, most users could formulate fairly complex queries in a reasonably short *time* (3–4 minutes); in contrast, we have found in informal tests that expert SQL coders take at least three times as long, while often failing to achieve a complete query (Hallett, Scott, and Power 2006). The *reliability* achieved over 60 queries was 100%, in the sense that all users managed to formulate all their targets. Finally, a further experiment showed that the formulations of the queries in the feedback texts were *transparent*, with accuracy rates of 80–90% on a multiple-choice comprehension task with a random baseline of 33%.

These evaluation results offer strong support for Conceptual Authoring as an approach to this class of problems: we know of no alternative approach that can achieve similar success in meeting these requirements. However, there are several areas where improvement should be possible. First, we need to find ways of facilitating the development process: building and maintaining a system like the CLEF query tool requires, at present, the hand-coding of linguistic and conceptual resources; as a step in this direction, we have developed a method of automatically inferring relevant types of queries for any database, and automatically constructing resources that inform a Conceptual Authoring interface (Hallett 2006). Second, we cannot be sure yet that the wording of feedback texts is optimal—perhaps with more research the comprehension rates can be pushed higher. Finally, we have only begun to explore the possibilities for improving the GUI for Conceptual Authoring.

References

Allen, James. 1984. Towards a general theory of action and time. *Artificial Intelligence*, 23(2):123–154.

Androutsopoulos, I. 1992. Interfacing a natural language front end to a relational database. Masters thesis, Department of Artificial Intelligence, University of Edinburgh.

Androutsopoulos, I., G. Ritchie, and P. Thanitsch. 1993. An efficient and portable natural language query interface for relational databases. In *Proceedings of the 6th International Conference on Industrial Engineering Applications of Artificial Intelligence and Expert Systems Edinburgh*, pages 327–330, Edinburgh.

Androutsopoulos, I., G. D. Ritchie, and P. Thanisch. 1995. Natural language interfaces to databases—an introduction. *Natural Language Engineering*, 2(1):29–81.

Bell, J. E. and L. A. Rowe. 1992. An exploratory study of ad hoc query languages to databases. In *Proceedings of the 8th International Conference on Data Engineering*, pages 606–613, Tempe, AZ.

Bouayad-Agha, Nadjet, Richard Power, Donia Scott, and Anja Belz. 2002. PILLS: Multilingual generation of medical information documents with overlapping content. In *Proceedings of the Third International Conference on Language Resources and Evaluation*, pages 2111–2114, Las Palmas, Spain.

Capindale, R. A. and R. G. Crawford. 1990. Using a natural language interface with casual users. *International Journal of Man–Machine Studies*, 32(3):341–361.

Catarci, T. and G. Santucci. 1995. Are visual query languages easier to use than traditional ones? An experimental proof. In *Proceedings of the International Conference on Human-Computer Interaction (HCI95).*

College of American Pathologists, 2004. *SNOMED Clinical Terms User Guide.* July 2004 release.

Deshpande, A., 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.

Deshpande, A., C. Brandt, and P. Nadkarni. 2003. Temporal query of attribute-value patient data: Utilizing the constraints of clinical studies. *International Journal of Medical Informatics*, 70:59–77.

Ely, John W., Jerome A. Osheroff, Paul N. Gorman, Mark H. Ebell, M. Lee Chambliss, Eric A. Pifer, and P. Zoe Stavri. 2000. A taxonomy of generic clinical questions: Classification study. *British Medical Journal*, 321:429–432.

Estrella, Florida, Chiara del Frate, Tamas Hauer, Richard McClatchey, Mohammed Odeh, Dmitry Rogulin, Salvator Roberto Amendolia, David Schottlander, Tony Solomonides, and Ruth Warren. 2004. Resolving clinicians queries across a grids infrastructure. In *Proceedings of the 2nd International HealthGRID Conference*.

Evans, Roger, Paul Piwek, Lynne Cahill, and Neil Tipper. In press. Natural language processing in CLIME, a multilingual legal advisory system. *Natural Language Engineering*.

Frank, Annette, Hans-Ulrich Krieger, Feiyu Xu, Hans Uszkoreit, Berthold Crysmann, Brigitte Jorg, and Ulrich Schafer. 2005. Querying structured knowledge sources. In *AAAI-05 Workshop on Question Answering in Restricted Domains*, pages 10–19, Pittsburgh, Pennsylvania.

Gorman, P. N. and M. Helfand. 1995. Information seeking in primary care: How physicians choose which clinical questions to pursue and which to leave unanswered. *Medical Decision Making*, (15):113–119.

Hafner, Carole D. and Kurt Godden. 1985. Portability of syntax and semantics in datalog. *ACM Transactions on Information Systems*, 3(2):141–164.

Hallett, Catalina. 2006. Generic querying of relational databases using natural language generation techniques. In *Proceedings of the 4th International Natural Language Generation Conference (INLG'06)*, pages 95–102, Sydney, Australia.

Hallett, Catalina, Donia Scott, and Richard Power. 2006. Evaluation of the CLEF query interface. Technical Report 2006/01, Centre for Research in Computing, The Open University.

Hendrix, Gary G., Earl D. Sacerdoti, Daniel Sagalowicz, and Jonathan Slocum. 1978. Developing a natural language interface to complex data. *ACM Transactions on Database Systems*, 3(2):105–147.

Hovy, E. H., U. Hermjakob, and D. Ravichandran. 2002. A question/answer typology with surface text patterns. In *Proceedings of the DARPA Human Language Technology Conference*, pages 247–250, San Diego, CA.

Jerome, R. N., N. B. Giuse, K. W. Gish, N. A. Sathe, and M. S. Dietrich. 2001. Information needs of clinical teams: Analysis of questions received by the clinical informatics consult service. *Bulletin of the Medical Library Association*, 89(2):177–184.

Kalra, Dipak, 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.

Kaplan, S. Jerrold. 1984. Designing a portable natural language database query system. *ACM Transactions on Database Systems*, 9(1):1–19.

Kate, R. J., Y. W. Wong, and R. J. Mooney. 2005. Learning to transform natural to formal languages. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, pages 1062–1068, Pittsburgh, PA.

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

Koonce, Taneya Y., Nunzia Bettinsoli Giuse, and Pauline Todd. 2004. Evidence-based databases versus primary medical literature: An in-house investigation on their optimal use. *Bulletin of the Medical Library Association*, 92(4):407–411.

Lowden, B. G. T., B. R. Walls, A. De Roeck, C. J. Fox, and R. Turner. 1991. A formal approach to translating English into SQL. In *Proceedings of the 9th British National Conference on Databases*, pages 110–127, Wolverhampton, UK.

Mueckstein, Eva-Martin. 1985. Controlled natural language interfaces (extended abstract): The best of three worlds. In *CSC '85: Proceedings of the 1985 ACM thirteenth annual conference on Computer Science*, pages 176–178, New York, NY.

Nadkarni, P. and C. Brandt. 1998. Data extraction and ad hoc query of an entity-attribute-value database. *Journal of the American Medical Informatics Association*, 5(6):511–527.

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

Piwek, Paul. 2002. Requirements definition, validation, verification and evaluation of the clime interface and language processing technology. Technical Report ITRI-02-03, ITRI, University of Brighton.

Piwek, Paul, Roger Evans, Lynne Cahill, and Neil Tipper. 2000. Natural language generation in the MILE system. In *Proceedings of the IMPACTS in NLG Workshop*, pages 33–42, Schloss Dagstuhl, Germany.

Popescu, Ana-Maria, Oren Etzioni, and Henry Kautz. 2003. Towards a theory of natural language interfaces to databases. In *IUI '03: Proceedings of the 8th international conference on*

Intelligent user interfaces, pages 149–157, New York, NY.

Power, Richard 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.

Power, Richard, Donia Scott, and Roger Evans. 1998. What you see is what you meant: direct knowledge editing with natural language feedback. In *Proceedings of the 13th Biennial European Conference on Artificial Intelligence*, pages 675–681, Brighton, UK.

Rector, Alan, Jeremy Rogers, Adel Taweel, David Ingram, Dipak Kalra, Jo Milan, Robert Gaizauskas, Mark Hepple, Donia Scott, and Richard Power. 2003. CLEF—joining up healthcare with clinical and post-genomic research. In *Second UK E-Science "All Hands Meeting"*, Nottingham, UK.

Rivera, Carlos and Nick Cercone. 1998. Hermes: Natural language access to a medical database. Technical report CS-98-03, University of Regina, Canada.

Scott, Donia, Richard Power, and Roger Evans. 1998. Generation as a solution to its own problem. In *Proceedings of the 9th International Workshop on Natural Language Generation*, pages 256–265, Niagara-on-the-Lake, Canada.

Shahar, Yuval and Cleve Cheng. 1999. Intelligent visualization and exploration of time-oriented clinical data. In *Proceedings of HICSS*, pages 4019–4030, Maui, HI.

Soricut, R. and E. Brill. 2004. Automatic question answering: Beyond the factoid. In *Proceedings of the HLT/NAACL 2004*, pages 57–64, Boston, MA.

Tang, Lappoon R. and Raymond J. Mooney. 2001. Using multiple clause constructors in inductive logic programming for semantic parsing. In *EMCL '01: Proceedings of the 12th European Conference on Machine Learning*, pages 466–477, London, UK.

Templeton, Marjorie and John Burger. 1983. Problems in natural-language interface to DBMs with examples from EUFID. In *Proceedings of the First Conference on Applied Natural Language Processing*, pages 3–16, Morristown, NJ.

Tennant, Harry R., Kenneth M. Ross, and Craig W. Thompson. 1983. Usable natural language interfaces through menu-based natural language understanding. In *CHI '83: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 154–160, New York, NY.

TREC. 2005. Question answering data. `http://trec.nist.gov/data/qa/t2005_qadata.html`.

Wilcox, Adam, George Hripcsak, and Cynthia Chen. 1997. Creating an environment for linking knowledge-based systems to a clinical database: A suite of tools. In *Proceedings of AMIA Annual Fall Symposium*, pages 303–307, Nashville, TN.

Zhang, Guogen, Wesley W. Chu, Frank Meng, and Gladys Kong. 1999. Query formulation from high-level concepts for relational databases. In *UIDIS '99: Proceedings of the 1999 User Interfaces to Data Intensive Systems*, page 64, Washington, DC.