MEDINFO 2007
K. Kuhn et al. (Eds)
IOS Press, 2007

Design and Evaluation of a Temporal, Graph-Based Language for Querying Collections of Patient Histories

Ole Edsberg[ac], Stein Jakob Nordbø[c], Erik Vinnes[bc], Øystein Nytrø[ac]

[a] *Department of Computer and Information Science (IDI)*
[b] *Faculty of Medicine (DMF)*
[c] *Norwegian EHR Research Centre (NSEP)*
Norwegian University of Science and Technology (NTNU), Trondheim, Norway

Abstract

Giving clinicians and researchers the ability to easily retrieve and explore relevant fragments of patient histories would greatly facilitate quality assurance, patient follow-up and research on patient treatment processes. Established database query languages are inconvenient for such exploration, and may also be too complex for users with limited backgrounds in informatics. We believe that understandability can be increased in return for a sacrifice of some of the power of expression found in general query languages. In order to design a specialized query language, we have collected and synthesized a tentative list of requirements. Based on these requirements, we have designed and implemented Practice Explorer, a prototype for visual query of collections of patient histories, and evaluated the understandability of its query language by testing with medical students. The results indicate that parts of the language are intuitive enough for users to understand without demonstrations, examples, feedback or assistance. They also provide some lessons for future work in this area.

Keywords:

information retrieval. Data display. Medical record

Introduction

Clinicians and health researchers have a need for querying their patient records for relevant history fragments. Four tasks where this need may arise are:

1. Retrospective study of guideline compliance [1].
2. Re-consideration of treatment plans for groups of patients possibly affected by the discovery of new medical knowledge, such as the connection between H. pylori and peptic ulcers.
3. Selection of patients for scientific studies.
4. Development of research hypotheses through explorative search of patient records.

The text-based database query languages available in today's patient record systems are either too complex for users without informatics competence or has insufficient power of expression. As one of the sources in our requirements collection put it, "(...) I lack basic search functionality, mostly because of my own aversion against learning a programming language for searching (...)".

The research question this article addresses is: How can we design a system for formulating temporal queries against patient history databases that is easily understandable for users with little competence in informatics, but still satisfies most of their query needs? The contributions of this article are 1) a tentative list of requirements for the expressiveness of patient history query languages, 2) an outline of our design for a query system satisfying the requirements and 3) results, observations and lessons learned from understandability testing of a prototype implementing our design.

Related work

The system most closely resembling Practice Explorer is the very recent PatternFinder [2]. The most important differences are that Practice Explorer visualizes queries as hierarchical, directed acyclic graphs, whereas PatternFinder uses linear chains of forms, and that Practice Explorer's query language, prompted by the requirements described below and enabled by the more flexible query model, is also able to express queries with parallel and alternative branches. It would be interesting to compare a form-based and a graph-based visual query system to see which is more intuitive for users without a background in informatics. Also related is TVQL [3], a visual query language where binary interval relations are specified via sliders, and where multiple such relations can be combined via neighborhood and disjunction. Another related study [4] proposes three new notations (elastic bands, springs and paint strips) for interval relations and experimentally compares their understandability. It also provides a table comparing different approaches to visualizing temporal relations and specifying combinations of such relations with logical expressions. In our case, the query graph both specifies local temporal relations and conjunction and disjunction of sub-queries through the parallel and alternative branching constructs.

Our visual representation of branching and joining constructs is borrowed from UML activity diagrams [5], a flowchart notation for defining workflows. In fact, our

query graphs can be viewed as flowcharts, extended with some new constructs and given an alternative semantics suitable for matching against histories.

Materials and methods

Requirements collection

Through discussions with two general practitioners, a rheumatologist and a health researcher interested in clinical processes, and through a pilot study applying an early prototype to a general practitioner's database and letting him verbally specify queries to be executed by the developers, we collected example queries and synthesized the following list of requirements for patient history query languages. Queries should be able to find patterns consisting of:

1. A primitive history element, such as a patient encounter, lab test, prescription or correspondence event.
2. Results limited by the date or the age of the patient at a specific point in a pattern.
3. A time interval in which a medication has been prescribed, including overlapping prescriptions, or the start or end of such an interval.
4. Time periods of variable length, with the possibility of specifying that a specified pattern should, or should not, occur during the period.
5. Repetitive occurrence of a specified pattern.
6. Sequences of specified patterns.
7. Parallel occurrence of specified patterns.
8. Alternative occurrence of specified patterns.
9. The first occurrence of a pattern in the whole history.

It should also be possible to:

10. Specify encounter events and medication intervals at various points of abstraction in relevant coding hierarchies.
11. Perform union and intersection set operations on query results.
12. Save and re-use queries or their components.

Table 1 shows a natural-language specification of a query need that exemplifies many of the requirements.

Table 1 - Natural-language specification of a query need

Find all patients who initiated medication with an ACE-inhibitor for the first time in their histories without having any encounters coded as angina, myocardial infarction or heart failure in the preceding two-year period, and who had an en-counter coded as hypertension some time between the history's start and the initiation of the ACE-inhibitor medication.

Find all patients who initiated medication with an ACE-inhibitor for the first time in their histories without having any encounters coded as angina, myocardial infarction or heart failure in the preceding two-year period, and who had an encounter coded as hypertension some time between the history's start and the initiation of the ACE-inhibitor medication.

We have no illusions that our list of requirements is complete. In our experience, potential users are often not fully able to understand the possibilities offered by a temporal query system without having such a system available for use on their own data. Therefore, it will be necessary to iterate between developing prototypes and collecting more requirements.

Data source and data model

Our test case is a patient record database from a general practitioner's office. For the sake of query performance, Practice Explorer extracts relevant data from the database at startup and represents the patient histories in main memory as lists of events. Each event has a time stamp and is considered to last for 24 hours because the test case database does not contain accurate time information at smaller granularities than days. The types of events include patient encounters (with diagnosis codes), lab tests, prescriptions and correspondence. For prescriptions, cessation dates are, where possible, heuristically deduced from fields in the prescription. Events have various attributes, such as codes, values and text, depending on their type. Practice Explorer is currently only able to extract data from Profdoc Vision, a patient record system widely used by Norwegian general practitioners. In this system, encounter diagnoses are coded according to the International Classification of Primary Care (ICPC), and prescriptions are coded according to the Anatomical Therapeutic Chemical Classification (ATC).

Query language

The main idea behind Practice Explorer's query language is to visualize queries as directed acyclic graphs, with each vertex describing a part of the history and with the edges always directed to the right, towards the present time. A query defines what a segment of a history must be like to constitute a match for that query. The informal interpretation of a query is that, for a query to match a segment of a history, it must be possible to simultaneously walk from left to right through the query and the history segment, encountering a matching history part for every query element passed. An edge between two elements indicates that the match for the element on the right hand side must begin at the exact same time that the match for the element on the left hand side ends. In other words, edges do not represent passage of time, but connects temporally juxtaposed events. Figure 1 shows two very simple queries. The bottom query specifies that a contact with diagnosis code K86 must occur, immediately followed by a period of medication with beta blockers. The top query specifies the same situation, except that an unlimited amount of time, represented by the spring-like middle element, is allowed to pass between the contact and the start of the medication period.

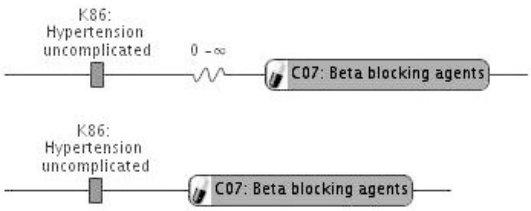

Figure 1 - Simple queries.

Table 2 - The central structure of the query language

```
<Q>        ::= <Point>
             | <Interval>
             | sequentialComposition(<Q>, <Q>)
             | parallelComposition(<Q>, <Q>)
             | alternativeComposition(<Q>, <Q>)
             | firstOccurrence(<Q>)
<Point>    ::= encounter(<IcpcCode>)
             | prescription(<AtcCode>)
             | medicationStart(<AtcCode>)
             | medicationEnd(<AtcCode>)
             | test(<TestType>, <TestValueRange>)
             | correspondence(<CorrType>)
             | dateControl(<DateRange>)
             | ageControl(<AgeRange>)
<Interval> ::= medicationInterval(<AtcCode>)
             | timeWindow(<Dur>, <Dur>)
             | timeWindowWith(<Dur>, <Dur>, <Q>)
             | timeWindowWithout(<Dur>, <Dur>, <Q>)
             | repetition(<Q>, <Int>, <Int>, <Dur>, <Dur>)
```

The user builds a query by dragging components from a menu to a panel containing the query graph under construction. Dialog boxes ask for necessary parameters as components are added. Figure 2 shows a moderately complex query graph.

The query language has a textual syntax, the central structure of which is defined by the grammar in table 2. A query is any derivation from <Q>. The visual representation of many of the elements can be seen in figure 2.

Space does not permit giving a full formal definition of the query language and its semantics. We will now informally describe the semantics of the query components given by the grammar.

Point queries, except for date controls and age controls, match events of the corresponding types satisfying the criteria in the parentheses. Date controls and age controls match all points in histories where the date or patient age could be successfully verified to belong to the specified range.

- Of the interval queries, medication intervals match any continuous time period where the patient is deduced, from prescription events, to be taking medication of the given code. The three types of time windows match time periods of duration between a given minimum and a given maximum, with a possible additional requirement that a given query must, or must not, match within the period. Repetition queries require that a given query gives repeating matches with upper and lower limits for the count and the duration allowed between repeats.

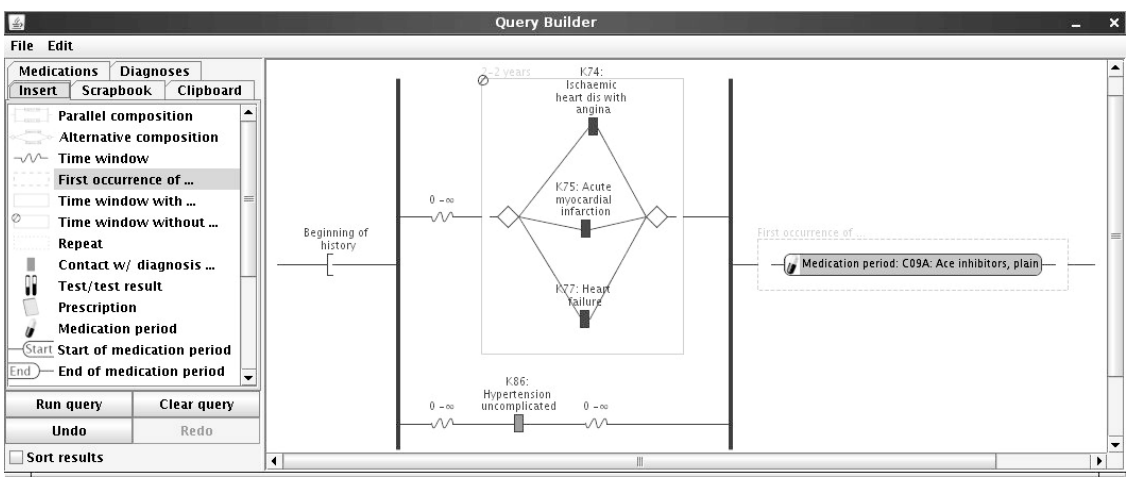

Figure 2 - Query building window showing a query satisfying the query need from table 1. From left to right, we see the following: an element matching the beginning of the history, followed by a branching into two parallel threads, along the upper of which the last two-year-period must not contain a contact with any of three codes and along the lower of which a contact with a given code must occur somewhere, followed by a re-joining of the parallel threads, followed by an interval of medication with a given type of drug, which must be the first of its kind in the entire history. (The query builder was constructed using the prefuse toolkit [6].)

- Sequentially composed queries require that the queries match, in such a way that the end of match of the first query coincides with the beginning of the match of the second query. This construction gives rise to the edges in the query graph.
- Parallelly composed queries require that the queries match, and in such a way that the start points of the matches coincide and that the end points of the matches coincide.
- Alternatively composed queries require that at least one of the queries matches.
- First-occurrence queries merely require that there is a match of the given query and that this is the first such match in the entire history. (This effect can be achieved with a combination of other elements, but the requirement was important enough to warrant support as a simpler formulation.)

The grammar gives the query graph an underlying tree structure where the matches of a node depends on how the matches of its children cohere with the rules and parameters for the node itself. The matches of a query graph are the matches of its root node. Execution of a query is performed with the leaf nodes scanning the history sequentially and the internal nodes iterating through the matches from their children.

The language elements described above straightforwardly satisfy requirements 1-10. Intersection and union of query results can easily be achieved by parallel and alternative composition of the queries, thus satisfying requirement 11. The recursive definition of the query language means that parts of queries, down to primitive components, are queries in their own right that can easily be collapsed, given names, saved and re-used, thus satisfying requirement 12.

Result visualization

Practice Explorer consists of two main windows. The query builder was described in the previous section. The history explorer, is shown in figure 3. It displays a number of vertically stacked horizontal bars, each providing a compact, very simple, LifeLines-like [7], explorable visualization of a history above a common time axis. The history explorer dynamically limits its view to the histories containing matches for the query given by the current state of the query builder, marks the hits of the query with red boxes and synchronizes the histories so that they are aligned on the first match.

Understandability testing

We performed 12 two-hour understandability tests, each followed by a brief questionnaire. The goal was not to test general usability, but to investigate the intuitive understandability of the underlying principles. We therefore simplified the query builder, keeping only the following query elements: encounter, history start, history end, date control, age control, time window, time window without ..., sequential, parallel and alternative composition. We applied the system to 2066 general practice patient histories of lengths up to 12 years. The test subjects were 4th- and 5th-year medical students.

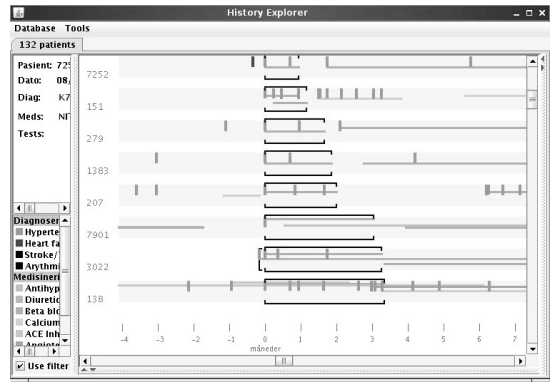

Figure 3 - Visualization of a query result

Tests 1-3 were informal, with demonstration and explanation provided during the test. In tests 4-6, the subjects were given a leaflet containing instructions and examples, as well as a number of query construction tasks to be performed. During tests 1-6 we made the observation that the learning process was greatly enhanced by the availability of examples, demonstration or interactive assistance. In particular, getting feedback on the correctness of one's queries improved ability to accomplish further tasks. We suspected that these factors could obscure issues related to our stated goal, which was to investigate the intuitive understandability of the underlying principles of the query system. Therefore, we devised a testing framework with the following rules: 1) The user will receive written instructions and query construction tasks to solve within an allotted time, 2) There must be no demonstration or assistance and 3) The instructions must contain no example queries. We carried out tests 7-12 in compliance with these rules. The subject was left alone for 105 minutes to read the instructions and attempt to solve the tasks, occasionally prompted via a loudspeaker to explain his or her thinking. Of the 25 tasks, the first 5 were point queries, the next 8 also involved intervals, the next 4 added branching constructs, and the final 8 required complex combinations of different types of elements, In the final 15 minutes, a developer interviewed the subject about the tasks that the test subject had failed to solve, and explained how those tasks should have been solved. Screen, video and audio were captured for further investigation.

Results

Figure 4 summarizes the correctness scores of test subjects 7-12 on the 25 query construction tasks. For an query to be classified as correct, it had to give the exact intended result.

From studying the users' actions on the screen, listening to them thinking aloud, and interviewing them afterwards, we made the following observations:

1. When a time window was specified as having a duration between an upper and a lower limit, for example 0 and 12 months, test subjects would often think of this as a fixed-length window from relative time point 0 to

relative time point 12 months, even though the instructions explicitly stated otherwise.

2. The test subjects made many errors where they seemed to assume that the query described the whole history and not just a fragment of it.
3. Test subjects frequently and successfully used the match-aligned result visualization to check if their queries were correct.
4. The test subjects made many errors related to not understanding that matches of parallel queries must cover the same time period.
5. On tasks requiring nested branching constructs, test subjects seemed to strain under the mental effort required. Some ceased serious efforts to find a solution.

Most of these observations either did not occur before tests 7-12, or occurred much more strongly in tests 7-12.

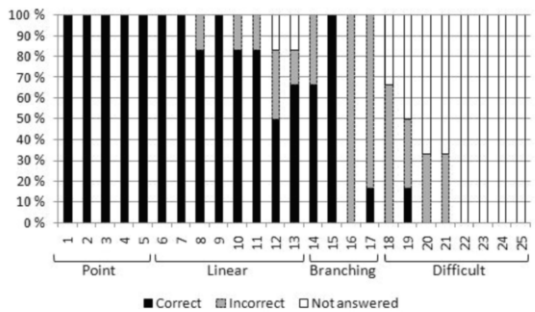

Figure 4: Scores for test subjects 7-12 on the 25 tasks.

On the questionnaire, which used a 5-point Lickert scale, 3 out of 12 marked “agree” or “fully agree” on the proposition “I think the visualization and building of the queries was easy to understand based on the instruction”. 9 out of 12 marked “agree” or “fully agree” on the proposition “I think the visualization and building of the queries was easy to understand based on the explanation given afterwards.”

Discussion

Studying figure 4, it appears that test subjects 7-12, even without examples, demonstration or assistance, arrived at a reasonably good understanding of the linear parts of the query language. The branching constructs appear to have been poorly understood. The survey answers indicate that explanation did help a lot on understanding. From the observations, we arrive at some lessons that may be helpful when designing this kind of system:

1. It seems much more natural for users to interpret a minimum and maximum number of time units as time points defining a constant-length interval rather than as bounds on the duration of a flexible interval.
2. Users may find it more natural to build a query describing the whole history rather than just a fragment.
3. Match-aligned result visualization can help users correct their own thinking and build correct queries.
4. Attention must be paid to make the visualization of branching constructs reflect their properties and reduce the effort required in reasoning about them.
5. Refraining from giving examples, demonstration, assistance or feedback may make testing more effective in uncovering problems.

Another question, partly addressed by our requirements collection, is whether the language is sufficiently expressive. On this topic we also note that, out of Allen's 13 primitive interval relations [8], our query language does not support *overlaps* or *overlapped-by*. The other 11 can be constructed with parallel composition and time windows. Supporting *overlaps*/*overlapped-by* would probably make the language more complex. Since we can match the start or end of a medication interval occurring during the match of another interval, we have not yet seen any query need requiring *overlaps*/*overlapped-by*.

Conclusion

We have designed, implemented and evaluated a temporal, graph-based query language based on our collected tentative list of requirements. Our understandability tests indicate that domain users relatively easily can construct point and interval queries, but not branching queries. Based on observations done during testing, we arrived at a list of lessons of potential relevance for the design of similar systems.

Acknowledgments

The authors wish to thank Arild Faxvaag, Anders Grimsmo, Joe Siri Ekgren, Berit Brattheim, Dag Svanæs, Yngve Dahl and Terje Røsand for valuable assistance and comments. Part of the work was funded by The Norwegian Research Council.

References

[1] Quaglini S, Ciccarese P, Micieli G, Cavallini A. Non-compliance with guidelines: motivations and consequences in a case study. Stud Health Technol Inform. 2004;101:75-87.

[2] Fails JA, Karlson A, Shahamat L, Shneiderman B. A Visual Interface for Multivariate Temporal Data: Finding Patterns of Events across Multiple Histories. IEEE Symposium on Visual Analytics in Science and Technology 2006.

[3] Hibino S, Rundensteiner EA. A visual multimedia query language for temporal analysis of video data. MultiMedia Database Systems, pages 123--159. Kluwer Aca, 1996.

[4] Chittaro L, Combi C. Visualizing queries on databases of temporal histories: new metaphors and their evaluation. *Data & Knowledge Engineering* 2003;44:239-264.

[5] Fowler M. UML Distilled, 3rd edition. Boston: Pearson, 2004.

[6] Heer J, Card SK, Landay JA. prefuse: a toolkit for interactive information visualization. Proceedings of CHI’05, ACM Press, 2005.

[7] Plaisant C, Milash B, Rose A, Widoff S, Shneiderman B. Lifelines: Visualizing Personal Histories. Proceedings of CHI’96. ACM Press, 1996.

[8] Allen J. Maintaining knowledge about temporal intervals. *Communications of the ACM* 1983:26(11):832-843.

Address for correspondence

Ole Edsberg, edsberg@idi.ntnu.no