EHR Query Language (EQL) – a query language for archetype-based health records

Chunlan Ma, Heath Frankel, Thomas Beale, Sam Heard

Ocean Informatics Pty. Ltd, Australia

Abstract

OpenEHR specifications have been developed to standardise the representation of an international electronic health record (EHR). The language used for querying EHR data is not as yet part of the specification. To fill in this gap, Ocean Informatics has developed a query language currently known as EHR Query Language (EQL), a declarative language supporting queries on EHR data. EQL is neutral to EHR systems, programming languages and system environments and depends only on the openEHR archetype model and semantics. Thus, in principle, EQL can be used in any archetype-based computational context. In the EHR context described here, particular queries mention concepts from the openEHR EHR Reference Model (RM). EQL can be used as a common query language for disparate archetype-based applications. The use of a common RM, archetypes, and a companion query language, such as EQL, semantic interoperability of EHR information is much closer. This paper introduces the EQL syntax and provides example clinical queries to illustrate the syntax. Finally, current implementations and future directions are outlined.

Keywords:

openEHR reference model, archetype, query language, electronic health record

Introduction

The National E-Health Transition Authority of Australia (NEHTA) defines an EHR query as a formal user or system request for information to the EHR repository/database that specifies the constraints on precisely what part(s) of the EHR content needs to be retrieved [1]. Due to the lack of any clear standards in EHR query services, NEHTA has identified it as a major area for review[2]. Easy accessibility of data from an electronic health record (EHR) is considered as one of the essential features of the EHRs that can enhance a hospital revenue cycle [3]. There are two major challenges commonly encountered in clinical data accessibility: one is that people who understand the clinical data best, such as health professionals, are not the ones most competent of querying the data; another challenge is that qualified SQL programmers or other query language programmers must spend much time on exploring the data, composing tedious code to provide the data that end user needs [4]. EHR data include thousands of facts to do with a patient's clinical status, which can be highly structured, semi-structured or non-structured, or most commonly, a mixture of all three. The lack of discipline commonly found in EHR data not only increases the difficulties in storing them, but particularly the difficulty in querying them.

The openEHR specifications have been developed to standardise the representation of an international electronic health record (EHR)1. Although there are implementations based on these specifications in development, experience with querying archetype-based EHR data is still limited, to the point where it is not clear what kind of query language(s) are even appropriate for information systems based on archetypes.

Since the openEHR specifications intend to define a standardised EHR infrastructure, a query language for this infrastructure should aim to become an open specification as well. The query language described here will be submitted to the openEHR foundation as a candidate openEHR query language.

There are four requirements for an archetype-based query language. First, the query language should be able to express queries for requesting any data item from an archetype-based system, i.e. data defined in archetypes and/or the underlying reference model . Second, the query language should be able to be used by both domain professionals and software developers. Third, the query language should be portable, i.e. be neutral to system implementation, application environment and programming language. Lastly, the syntax should be neutral with respect to the reference model, i.e. the common data model of the information being queried. Particular queries will of course be specific to a reference model.

The current available query languages that might potentially be used to query openEHR data include the XML Query Language (XQuery) [5] and the Structured Query Language (SQL) . XQuery uses eXtensible Mark-up Language (XML) as its underlying data model. It has rich predefined functions and allows user-defined functions supporting the kind of clinical data requests required in the EHR context. It is platform independent. Nevertheless, its main strength is also its main flaw: it is limited to purely XML data environments. Direct use of XQuery for the openEHR EHR would require that all

[1]http://www.openehr.org/

openEHR data must be represented in XML format. However, openEHR is designed as an object-oriented framework, and allows for a multitude of data representations, including as programming language persistent objects (e.g. in the form of Java objects in a product such as db4o2); as language neutral objects (such as in a database like Matisse3); as relational structures (governed by an object/relational mapping layer), and in various XML storage representations (e.g. XML blob or XML databases). XQuery is therefore problematic, because the query syntax is directly tied to the representational format of the data. Considerable efforts would be required to convert openEHR data in each deployment context to XML just for the purpose of querying; each such transformation may well be custom, requiring special work on the part of the system implementers.

A further disadvantage is that XQuery is a native XML programming language requiring intimate familiarity of both users (in this case, health professionals and software developers) alike with XML and XQuery, something that cannot be assumed.

SQL in its standard form is also not a viable candidate for querying archetype-based EHR data, because it does not support object-structured data, e.g. the data modelled in archetypes. It has been found that considerable intellectual efforts are required when using SQL to search and retrieve clinical data for both individual subject care and clinical research studies [6].

Object Query Language (OQL) is a query language for object-oriented databases. OQL was developed by the Object Data Management Group (ODMG4), which was disbanded in 2001. Comparing with XQuery and SQL, OQL would be the best candidate query language used for archetype-based EHR data. However, it is complex and as a result, it has not been widely implemented. For large object models, such as the openEHR RMs, OQL query statements can become extremely verbose. OQL uses an object programming style dot-notation to express object members, while an XPath-based syntax is specified by openEHR to locate archetype data elements. Using OQL with archetype-based EHRs would require the use and translation between these two notation styles.

To satisfy the aforementioned requirements of an archetype-based query language, a new language – EHR Query Language (EQL) is under development. This paper introduces the EQL features and syntax. Example clinical query scenarios are used to demonstrate the use of the EQL expression.

Methods

EQL was developed based on the analysis of a set of clinical query scenarios, the study of the current available query language syntaxes (including XQuery, SQL and Object Query Language), and the study of the archetypes technology, openEHR RM and openEHR path mechanisms.

What is EQL

EQL is a declarative query language developed exclusively for expressing the queries used for searching and retrieving the clinical data found in archetype-based EHRs. It is applied to the openEHR EHR Reference Model (RM) and the openEHR clinical archetypes, but the syntax is generic across applications, programming languages, system environment, and reference model. The EQL is designed as a common language used for expressing clinical data requests across multiple openEHR-based applications.

The EQL has two innovations: 1) utilizing the openEHR path mechanism to represent the query criteria and returned results; and 2) using a 'containment' mechanism to indicate the data hierarchy and constrain the source data to which the query is applied.

OpenEHR path mechanisms

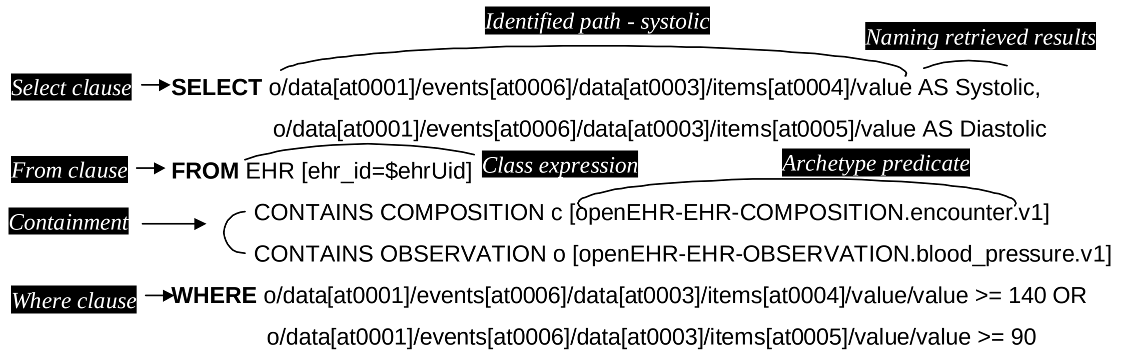
OpenEHR path syntax is used to locate clinical statements and data values within them using Archetypes. An Archetype is a computable expression of a clinical concept in the form of structured constraint statements, based on some reference model [7], such as the openEHR RM which provides the support for clinical archetypes. Each archetype has a global unique identifier and each node of this archetype has a unique archetype node identifier. The openEHR architecture has a path mechanism that enables any node within a top level structure to be specified from the top of the structure using a "semantic" X-path compatible path. The availability of such paths radically changes the available querying possibilities with health information, and is one of the major distinguishing features of openEHR [8]. Consequently, it is possible to locate any node in an archetype, including leaf data elements by using the archetype and archetype node identifiers within openEHR paths.

Features of the EQL

The EQL features are listed below:

- Neutral expression syntax. EQL does not have any dependencies on the underlying RM of the archetypes. It is neutral to system implementation and environment. This is one of the distinguishing features of the EQL.
- Allows setting query criteria using archetype and node identifiers, data values within the archetypes, and class attributes defined within the openEHR RM.
- Allows the returned results to be top-level archetyped RM objects, data items within the archetypes or RM attribute values.
- Supports naming returned results.
- Support queries with logical time-based data rollback.
- Supports query criteria parameters.
- Supports arithmetic operations (such as count, addition, subtraction, multiplication, and division), relational operations (>, >=, =, !=, <=, <) and Boolean operations (or, and, xor, not).

[2] http://www.db4o.com/
[3] http://www.matisse.com/
[4] http://www.odmg.org/



```
SELECT o/data[at0001]/events[at0006]/data[at0003]/items[at0004]/value AS Systolic,
       o/data[at0001]/events[at0006]/data[at0003]/items[at0005]/value AS Diastolic
FROM EHR [ehr_id=$ehrUid]
     CONTAINS COMPOSITION c [openEHR-EHR-COMPOSITION.encounter.v1]
     CONTAINS OBSERVATION o [openEHR-EHR-OBSERVATION.blood_pressure.v1]
WHERE o/data[at0001]/events[at0006]/data[at0003]/items[at0004]/value/value >= 140 OR
      o/data[at0001]/events[at0006]/data[at0003]/items[at0005]/value/value >= 90
```

Figure 1: A typical EQL statement

- Supports some functions that are supported in XQuery, such as current-date().
- Users could specify their preference on the retrieved data, such as ordering preferences, or total number of retrieved results.
- Supports the queries for individual clinical subjects at the point of care, administrative purposes and clinical research purposes.

EQL Syntax

Figure 1 shows a typical EQL statement, which would return all blood pressure values where systolic value is greater or equal to 140 or diastolic value is greater or equals to 90 within a specified EHR. EQL syntax is a synthesis of SQL structural syntax and openEHR path syntax. The SQL clauses SELECT, FROM, WHERE, and ORDER BY provide the basic structure of EQL. The SELECT clause specifies the data elements to be returned. The FROM clause specifies the result source and the corresponding containment criteria. The WHERE clause specifies data value criteria within the result source. The ORDER BY clause indicates the data items used to order the returned result set.

The openEHR path syntax [7], which is XPath compatible, is used in EQL to identify data items. Apart from the path mechanisms, EQL has a containment constraint, which specifies the hierarchical relationships between parent and child data items. The details of the EQL syntax are introduced below.

openEHR paths

The details of the openEHR path can be found elsewhere [7]. Herein, we briefly introduce its basic syntax.

A general pattern of the openEHR path syntax starts with a slash and followed by an attribute name of an object defined by the RM, and followed by another slash and attribute name and so on if there are multiple attributes involved, e.g. /ehr_id.

Another pattern of the path is a subset of the XPath syntax for predicates with a small number of short-cuts, i.e. archetype path. This path utilises an archetype ID or archetype node ID (i.e. at code, such as at0006) to identify an object. The example shown in Figure 1 uses the openEHR path syntax to locate both systolic and diastolic value in the blood pressure archetype.

Predicates

A predicate pattern is delimited by square brackets ([]). The path predicate has two operands and an operator. The operands are either an RM class attribute, an openEHR path or data value, such as a string or integer. A parameter name prefixed with a $ symbol can be used instead of a data value, which is substituted for a real data value at run time, e.g. ehr_id=$ehrUid shown in Figure 1. The operator can be =, >, <, >=, <=, or !=.

The archetype node ID predicates are a shortcut of a path predicate, which include either an archetype ID or archetype node ID. The archetype node ID predicates can also use a parameter name for the archetype node ID value, e.g. [$compositionArchetypeId]. The archetype node ID predicate shortcut is equivalent to a long-form path predicate with a left operand of archetype_node_id, and an operator of =. For example COMPOSTION c[openEHR-EHR-COMPOSITION-encounter.v1] shown in Figure 1 is equivalent to COMPOSTION c[archetype_node_id = 'openEHR-EHR-COMPOSITION-encounter.v1'].

EQL variables

All variables must have a RM class type assigned and they must be defined in the FROM clause, e.g. character c and o shown in the FROM clause of Figure 1 are EQL variables. EQL variables need to be declared when other clauses need a reference to them.

FROM clause

A FROM clause represents the data source for the query. It starts with the keyword – FROM, followed by a class expression indicating the RM classes, containment constraints and object identifying criteria used to constrain the data source for the query (refer to Figure 1).

For the openEHR RM, the classes declared in the FROM clause may include EHR, COMPOSITION, entry classes (e.g. OBSERVATION), and data structure class (e.g. ITEM_LIST).

Class expressions

An EQL class expression consists of a RM class name, an EQL variable (optional), and an optional class predicate. The class predicate further constrains the objects used as the query source and is normally applied to the archetype node ID or unique object ID class attributes.

Containment is indicated in EQL using the CONTAINS keyword between two class expressions. The left class expression indicates the parent object of the right class expression.

Identified paths

Identified paths are used to locate data items within an archetyped RM class. The identified path starts with an EQL variable that is declared within the class expression. This is followed by a slash and an openEHR path. The expression – o/data/[at0001]/event/…/value – shown in Figure 1 is an example of an identified path.

WHERE clause

A where clause is used to represent further criteria applied to the data items within the objects declared in the FROM clause. A WHERE clause starts with the keyword – WHERE, followed by a criteria expression. It consists of two operands and an operator. The operands may be an identified path or data value, such as a string or integer. A parameter name prefixed with a $ symbol can be used instead of a data value and substituted at run time. The operator can be =, >, <, >=, <=, or !=. Multiple criteria expressions can be combined using Boolean operators: AND, OR, XOR, and NOT.

SELECT clause

A SELECT clause starts with the keyword – SELECT, and followed by a set of identified paths. The set of paths are separated using a comma (see Figure 1).

Results naming

EQL allows users to rename the returned items specified in the SELECT clause. The keyword is AS, followed by the specified name, e.g. AS Systolic shown in Figure 1 SELECT clause.

ORDER BY clause

An ORDER BY clause starts with the keywords – ORDER BY, followed by a set of identified paths indicating the data items used to sort the returned result set. The keywords ASCENDING and DESCENDING (and abbreviations, ASC and DESC) can be used after each identified path as per SQL.

Arithmetic Functions

A set of arithmetic functions, such as addition, subtraction, multiplication, and division can also be used in EQL. The use of these functions is the same as SQL, and is not described here.

TIMEWINDOW clause

TIMEWINDOW is an addition query clause used in EQL to constrain the query to data that was available in the system within the specified time criteria. This supports a time-based logical system rollback allowing a query to be executed as though it was performed at that specified time, which is essential for medico-legal reporting. It starts with the keyword – TIMEWINDOW, and followed by a string compatible with the ISO 8601 representation of time interval. The first example below constrains the query source to data committed to the system before 2006-01-01. The second example constrains the query source to data committed within the period of two years before 2006-01-01.

1) TIMEWINDOW /2006-01-01

2) TIMEWINDOW P2Y/2006-01-01

Clinical scenarios

To illustrate the use of the EQL syntax, this section provides the EQL expressions for two typical clinical scenarios.

Scenario one

Scenario description

Get the number of all patients with diabetes who have HbA1c results greater than 7.0 in last 12 months.

EQL expression

```
SELECT COUNT(e/ehr_id)
FROM EHR e
  CONTAINS
  (COMPOSITION c
       [openEHR-EHR-COMPOSITION.problem_list.v1]
     CONTAINS EVALUATION e
       [openEHR-EHR-EVALUATION.problem-
diagnosis.v1]
   AND
   COMPOSITION c1
       [openEHR-EHR-COMPOSITION.report.v1]
     CONTAINS OBSERVATION o
       [openEHR-EHR-OBSERVATION.laboratory-
hba1c.v1])
WHERE
  e/data/items[at0002.1]/value/value=’diabetes
mellitus’ AND
  c1/context/other_context/items[at0006]/
    items[at0013]/value > current-date() -P1Y AND
   o/data/events[at0002]/data/items[at0013.1]
    /value/numerator > 7
```

Scenario two

Scenario description

Get all HbA1c observations that have been done in the last 12 months for a specific patient.

EQL expression

```
SELECT o
FROM EHR e[ehr_id=$ehrId]
  CONTAINS COMPOSITION c
   [openEHR-EHR-COMPOSITION.report.v1]
   CONTAINS OBSERVATION o
    [openEHR-EHR-OBSERVATION.laboratory-hba1c.v1]
WHERE
c/context/other_context[at0001]/items[at0006]/
items[at0013]/value > current-date()-1PY
```

Scenario three

Scenario description

Get a patient's current medication list

EQL expression

```
SELECT c
FROM EHR e[ehr_id=$ehrId]
  CONTAINS COMPOSITION c
   [openEHR-EHR-COMPOSITION.medication_list.v1]
WHERE c/name/value= 'current medication list'
```

Discussions

This paper has introduced EQL – a declarative query language developed for querying openEHR-based EHRs by utilising the openEHR path mechanisms and unique containment syntax. However, the EQL syntax is not specific to the openEHR RM and can be used for any archetype-based information system. It could be used as a common query language for disparate archetype-based applications. The EQL will be submitted to the openEHR foundation as a candidate openEHR query language.

EQL implementation

Ocean Informatics Ptd Lty has implemented the components to process the EQL within the OceanEHR suite of EHR tools. These components include an EQL parser, the EHR query object model and a query processor. The implementation does not currently support all features of the EQL. However it has demonstrated the power and flexibility of using a common RM, archetypes and EQL, independent from the underlying system implementation, to retrieve any data set from an EHR.

The EQL is not easily understood by health professionals due to the computer-oriented openEHR path syntax used in EQL. An EQL query editor, which allows users to generate and edit EQL statements, has been developed to empower clinicians with fine-grained access to their EHR data. The query editor provides users with access to an archetype repository to build FROM containment constraints and a tree representation of RM attributes and archetype structures used to set WHERE criteria, SELECT data items and ORDER BY preferences. The tool can then execute the generated query and display the returned results without the user seeing or knowing how to write an EQL statement.

Future directions

The EQL continues to be developed based on requirements from additional clinical query scenarios. New EQL features may need to be provided, such as statistical, string pattern matching and user-defined functions. Existential ($\exists$) and universal ($\forall$) quantifiers may be also required.

Other research areas may include 1) explore how the EQL supports clinical decision support technologies, e.g. clinical guidelines presentation; 2) investigate the integration of EQL with clinical terminology servers; and 3) conduct field trials using the EQL to represent common clinical queries to retrieve openEHR-based EHR data sets.

Conclusion

The use of a common RM and archetypes supports the sharing of EHR data, and with the addition of a companion query language, such as EQL, semantic interoperability of EHR information is much closer.

Acknowledgements

We extend our thanks to Lisa Thurston for her support, including the development of the EhrView component – a generic EHR viewer – that uses EQL for retrieving EHR data.

References

[1]. National E-Health Transition Authority (NEHTA). Acronyms, Abbreviations & Glossary of Terms in Publications. In: eds. National E-Health Transition Authority (NEHTA), 2005.

[2]. National E-Health Transition Authority (NEHTA). Review of Shared Electronic Health Records Standards in Publications. In: eds. National E-Health Transition Authority (NEHTA), 2006.

[3]. Amatayakul M. When EHRs are a-ok. Healthcare financial management, 2006: 60(2): 146-148.

[4]. Nadkarni PM and Brandt C. Data Extraction and Ad Hoc Query of an Entity-- Attribute-- Value Database. J Am Med Inform Assoc, 1998: 5(6): 511-527.

[5]. W3C.2006. XQuery 1.0: An XML Query Language. http://www.w3.org/TR/xquery/, accessed on 17 Oct. 2006

[6]. Johnson S and Chatziantoniou D. Extended SQL for manipulating clinical warehouse data. Proceedings / AMIA ... Annual Symposium, 1999: 819-23.

[7]. Beale T and Heard S. Archetype Definitions and Principles in The *open*EHR foundation release 1.0. In: Beale T and Heard S, eds. The *open*EHR foundation, 2005.

[8]. Beale T and Heard S. openEHR Architecture: Architecture Overview in The *open*EHR foundation release 1.0.1. In: Beale T and Heard S, eds. openEHR Foundation, 2006.

Address for correspondence

Dr Chunlan Ma

Clinical Informatics Consultant

Chunlan.ma@oceaninformatics.biz

Heath Frankel

Product Development Manager

Heath.frankel@oceaninformatics.biz