Expression and query languages for GLIF

Omolola Ogunyemi, Qing Zeng, Aziz Boxwala
Decision Systems Group
Brigham and Women's Hospital, Harvard Medical School

Introduction

This document contains a draft proposal for an expression language and a query language for use in GLIF and in relevant standards produced by the HL7 Decision Support Technical Committee (DSTC).

Considerations that went into developing these specifications were:

1. Sharability of the criteria and queries
2. Use of existing industry standards where possible
3. Expressibility of the languages
4. Ease of use of the languages

The query language has been designed in the context of the guideline execution model proposed in the HL7 DSTC. The model proposes the use of a virtual electronic medical record (vEMR) that provides a standard interface to heterogeneous medical record systems. While the query language does not depend on the specific classes or tables in the vEMR it does depend on the general framework of the vEMR. The expectations of and constraints specified by the query language on the virtual EMR are described in a later section.

The expression language can be used for specifying decision criteria, and abstracting or deriving summary values. We chose an object-oriented model for the language to allow for the *flexibility* and *extensibility* that is needed for implementation in a broad range of applications. The expression language stresses list and temporal operations. This version of the language has not been developed to handle uncertainty or vagueness. Support for such reasoning can be implemented in future versions through addition of classes.

The query and expression language must share a common data model since the expression language must use the results of the query. To that end, we specify a data model consisting of basic data types and some built-in classes derived from the HL7 RIM specification.

Query Language

Virtual EMR

- The query language will be querying a vEMR.
- Given that the vEMR is still under development, we are only making certain assumptions here.
- vEMR is a set of objects representing medical data
- The set of objects might be organized as a tree or a forest. Thus, the vEMR must specify certain root objects from which traversal of the EMR objects begins.
- The attributes of each object are specified
- Each attribute can be a basic data type, built-in class or added class.

Features

- This query language is developed base on OQL and TSQL
- Basic syntax:

  query ::= select vEMR object

  from vEMR objects

  [where criteria]

- Because we allow selection from multiple objects and reference to multiple objects in criteria, *join* is implied.
- The criteria in the where clause can refer to individual attributes
- Basic operator: +, -, *, /, mod, abs
- Comparison operator: like, =, !=, >=, <=, >, <
- Logic operator: and, or, not
- Collection operator: for all, exists, unique, in, some, all, any, count, sum, min, max, avg

- Temporal operations:

  Terminology

TSQL	**GEL**	**Example**
Interval	Duration	3 days
Indeterminate interval	Duration interval	0 days ~ 1 dsys
Period	Timstamp interval	1997-years ~ 1998-years 1997-02-01 days ~ 1998-03-01 days
Datetime	Timestamp	1997-years 1997-02-01 days

Conversion Operators

Scale:

duration (1 days)

to duration (24 hours)

to duration interval (0 month to 1 month)

timestamp (1997 years)

to timestamp interval (1997-01 month ~ 1997-12 month)

Cast:

Duration (1 month)

to duration (28 days)

timestamp (1997 years)

to timestamp (1997-01 month)

Comparison Operators

	Duration	Duration interval	Timestamp	Timestamp interval
Duration	> = < / + -	Contains Precedes	Arithmetic (+, -)	> = <
Duration interval	Contains Precedes	Contains Proceeds Overlap Start End = Meets	None	Arithmetic (+, -)
Timestamp	Arithmetic (+, -)	None	> = <	Contains Precedes
Timestamp interval	> = <	Arithmetic (+, -)	Contains Precedes	Contains Precedes Overlap Start End = Meets
Number	* /	* /	None	None

Reference:

Allen, J. F. "Maintaining Knowledge about Temporal Intervals." Communications of the Association of Computing Machinery, 26, No. 11, Nov. 1983, pp. 832–843.

Allen's interval comparison operators:

a finishes b

a starts b

a during b

a overlaps b

a meets b

a equals b

a before b

Results

- Results are returned as set, bag or list of objects (see built-in datatypes) depending on the exact query syntax:
    - Bag: default return type
    - List: When the results have to be ordered
    - Set: When the resulting elements are unique
- Results can be cast to a compatible type using the cast operator ().

Expression Language

Overview

GLIF's new expression language will be strongly typed and object-oriented. In addition to basic data types and operations, it will have a set of built-in classes and class attributes and methods that can be used to create complex mathematical, logical, and temporal expressions. The language will provide a mechanism for users to extend the initial set of classes provided, by defining new classes along with their attributes and methods. This will facilitate the process of encoding and evaluating expressions that rely on classes and relationships specified in data models used by GLIF. This is especially important in enabling GLIF to successfully support different data models, as classes and relationships specified will vary from one data model to another. The syntax of the language will be similar to that of GEL, with new constructs added to aid designation of an object's attributes and methods. Some GEL built-in operators will be discarded in the new language both for simplicity and in a bid to approximate the OO property of encapsulation (e.g., GEL's built-in temporal operators will be methods of an appropriate class of the new language).

Basic Data Types

The basic data types that will be supported by the expression language are listed below:

- Integer
- Real
- String
- Extended Boolean (true, false, unknown)

Built-in Operators

Arithmetic operators

Basic arithmetic operations that will be supported on integers and real numbers include unary minus, and unary plus, addition, subtraction, multiplication, and division. Some of these operators will work with built-in temporal classes.

String operators

Concatenation operations will be supported on strings.

Type assessment operators

Unary operators that determine whether a given value is an integer, real number, string, extended boolean, null value, etc., will be provided.

Comparison Operators

The following operators will be supported for integer or real number comparisons and for lexicographic comparisons of strings: equality (=), inequality (<>), less than (<), less than or equal to (<=), greater than (>), greater than or equal to (>=)

Logical operators

Logical and, or, not, and exclusive or operators will be supported.

Complex Data Types & Built-in Classes

Complex data types will be modeled in the language as classes and include:

Temporal/time-related classes

- Absolute Date and Time (timestamp) (will have methods such as isBefore, isAfter, isEqual, extractYear, extractDay, extractMonth, isWithinPast, etc.)
- Duration (will have methods such as beforeNow, afterNow, isEqual, etc.)

Interval classes

- Absolute Date and Time (timestamp) Interval (will have methods such as overlaps, etc.)
- Duration Interval
- Numeric Interval

Collection classes

- List (will have methods such as isEmpty, length, contains, etc.)
- Set
- Bag

Utility classes

- Ratio
- Math (will have methods such as absolute value, exp, sin, cos, tan, floor, ceiling, sqrt, etc.)
- Concept

Declaration Mechanisms

The mechanism and syntax for defining a new class and methods in the new expression language have not been finalized. An approach similar to the one taken in GEL could be adopted; suggestions are welcome.

Flow-control

Constructs that allow direction of the flow of control in evaluating expressions will be provided. These include if-then statements and for loops. Note that these are only intended to be used to enhance the expressive power of the language by facilitating the creation of complicated expressions. These constructs are *not* meant to be used for invoking action steps within the GLIF model itself.

Language-based Mechanisms for Handling Uncertainty

While it would be desirable to add classes and methods that provide support for uncertainty handling – e.g., for Bayesian networks, fuzzy logic, etc., at this stage, the language will not provide support for this. We intend to introduce these however, as it may be necessary to handle

uncertainty in guideline criteria using probabilistic, pseudo-probabilistic, or non-monotonic logic approaches.

Handling Exceptions

Decisions on appropriate mechanisms for handling exceptions due to syntax errors or errors in the logic of an encoded expression will need to be determined. This is an issue that will have an impact on the execution engine.

Examples

The examples cannot be made as clear without a vEMR model. We do list our assumptions about classes in the vEMR when pertinent.

Example 1: Is a list empty

list_of_drugs is an object of class Set. It has a member function called isEmpty that returns a boolean.

```
list_of_drugs.isEmpty()
```

Example 2: Select drugs from a list that do not have interactions with patient's existing drugs

Step 1. Get patient's current drugs from vEMR

```
meds=select current_drugs.name from current_drugs in
Patient.medications where
(current_drugs.valid_time.base_time_point +
current_drugs.valid_time.duration) > now
```

Patient is the root of the Virtual EMR. Medications is a collection of objects of type Medication (from the hypertension vEMR from EON)) representing the medications that have been prescribed to the patient. This query returns the names of all the drugs the patient is currently on.

Step 2: Match patient's current medications to contraindications

```
for (d in CandidateDrugs) {
  x = d.getContraindicatedDrugs();
  if (! x.contains(meds))
    selected_med.add(d);
}
```

CandidateDrugs is a list of possible drugs that can be given to a patient eligible for this guideline. x is a temporary variable to store the contraindications for drug d in the loop. selected_med is an object of the list class. meds is the result of the query in step 1. selected_med contains the final result.

Example 3: Chronic cough in adult

```
Time.cast(Time.now()-cough.valid_time.base_time_point,
"weeks") > 3 weeks
```

The Time class is a built-in class that has functions called cast and now. The former function casts the duration to the units specified in the second argument.

Example 4: Retrieve current prescriptions for ACE inhibitors

```
(MedicationList) select last(medication) from medication in
Patient.medications where medication.name = “ACEI” and
((medication.valid_time.base_time_point +
medication.valid_time.duration) > now) sort by
medication.valid_time.base_time_point
```

This is very similar to the query in example 1. This example shows how the results of a query (List) can be cast to a new class (MedicationList, a subclass of List). It also demonstrates use of sort and the function “last”.