GELLO – Guideline Expression Language for Clinical Decision Support

Margarita Sordo, PhD
Children’s Hospital Boston
Harvard Medical School

Slides selected from GELLO – OCL Premier by Robert Dunlop, MD
Update on GELLO

GELLO v2 has been fully aligned to the latest version of the Object Constraint Language (OCL).

GELLO v2 will be balloted in September.

Some changes in LET statement DEF function and removal of Factory function for creation of complex objects.

An authoring tool is under development and will be ready probably for this months/s HL7 meeting – a complier will follow.
Overview of OCL

OCL was developed by the Object Management Group to constrain and to query UML Class Models. As an expression language, it has the following features:

- Vendor-independent
- Platform-independent
- Object-oriented
- Side-effect free
- Flexible
- Extensible
- Declarative
GELLO v2 and OCL

**GELLO v2** is a subset of OCL that is constrained to the healthcare domain. GELLO can query HL7 Class Models, including the RIM and R-MIMs. GELLO enables:

- Formal definition of constraints for HL7 Class Models
- Creation of unambiguous query expressions based on HL7 v3 RIM-based Models for the purpose of:
  - Retrieving clinical information from data sources
  - Creating and executing clinical decision support criteria
  - Reasoning about clinical data/values, including temporal reasoning
- Exploitation of the features of OCL
Contents

A brief description of GELLO v2 and OCL. It covers the following areas:

- UML and HL7 Class Models
- Setting Packages and Contexts for GELLO expressions
- Explanation of the self and def keywords
- let assignments and the in keyword
- Operations, including query operations
- Dot/period notation and the Arrow (->)
UML and HL7 Class Models

UML Class Models in HL7 v3 comprise classes and associations between classes:

- The RIM Base Classes and Associations are illustrated below
- Classes are the colored rectangles
- Associations are the red lines

May '11 GELLO 6
Specifying a GELLO Package

**Package** is used to specify the base class model being used for a set of GELLO expressions:

GELLO expressions are written within the `package... endpackage` ‘container’

HL7 v3 RIM ‘Package’
Setting the Context

Context specifies the model entity for which a GELLO expression is defined. This is usually a Class:

- ‘context Entity’ will set the focus on the Entity Class

  • NB: Context may specify other model entities such as a datatype or an operation
Setting the Context

‘context Entity’ example is illustrated below:
Using \textit{self} Keyword

\textit{self} may be used within a GELLO expression to refer to an instance of the context class:

\begin{itemize}
  \item \textit{self} would refer to the Entity Class
\end{itemize}
**def**: Expressions

**def**: is used to create new attributes or operations within class models:

This example defines an attribute for the context Entity class:
**def**: Expressions

The **name** of the new attribute follows the first colon, i.e. **allRelatedActs**

The **type** of the attribute follows the second colon, i.e. **Set( Act )** which is a set of Act Class objects

The GELLO derivation expression follows the = (equals) operator

- **NB:** The expression result must be of the same type as the new attribute
What *def:* is Doing

The result of *def:* is illustrated in the following diagram:

```
def: allRelatedActs: Set( Act )
```

NB: The result of a *def:* expression will not persist after the execution of the GELLO expression. GELLO-defined attributes and operations do not represent permanent modifications of the HL7 v3 RIM or other pre-defined HL7 class models.
**let Assignments**

*let* is used to create temporary variables within a *def*: statement:

```plaintext
package HL7_V3_RIM

def: allRelatedActs: Set( Act ) = self.playedRole.participation.act
def: age: Integer =
  let cameOfAge : Integer = 21,
  twentySixYearsLater : Integer = 26
in
cameOfAge + twentySixYearsLater

endpackage
```
let Assignments

Following execution of the `def:` expression:

```swift
def: age: Integer = 
    let cameOfAge : Integer = 21,
    twentySixYearsLater : Integer = 26
in
    cameOfAge + twentySixYearsLater
```

- The result is an `age` attribute with a value of 47
- The values of the `let` variables do not persist
**let Comma Delimiter**

**Multiple let assignments are separated by a comma delimiter:**

```
package HL7_V3_RIM
context Entity

def: allRelatedActs: Set( Act ) = self.playedRole.participation.act
def: age: Integer =
    let cameOfAge : Integer = 21,
    twentySixYearsLater : Integer = 26
in
cameOfAge + twentySixYearsLater

endpackage
```
The *in* keyword is used to delineate the beginning of the definition expression:

```java
package HL7_v3_RIM

class Entity

context Entity

def: allRelatedActs: Set( Act ) = self.playedRole.participation.act

def: age: Integer =
    let cameOfAge : Integer = 21,
        twentySixYearsLater : Integer = 26

    in cameOfAge + twentySixYearsLater

endpackage
```
Operations

Operations are used in OCL to return a value or a set of values. They may be pre-defined in the Class Model that is being used, or they may be defined in OCL using **def**: 

- The HL7 v3 RIM does not contain pre-defined operations, only attributes
- The HL7 Data Types Class Model does include pre-defined operations:

The PointInTime datatype includes the `equals()` operation. Inside the brackets is the type of any input parameter/s passed to the operation. In this example, the input would be another PointInTime. After the colon, the type of the output value/s is specified, which is a Boolean true or false value in this example. This operation will compare two time points and return true if they are equals, false if they are not.
Query Operations

Query operations are only able to return a value or set of values. They cannot be used, for example, to insert a new value into a patient electronic medical record database for example:

- All user-defined operations that are created using `def:` in GELLO are Query Operations
- The syntax of a user-defined operation is illustrated below:

```python
class Patient:
    def age(todaysDate: PointInTime): Integer =
        todaysDate - self.player -> select(oclIsTypeOf(Person)).birthTime
```

<table>
<thead>
<tr>
<th>Operation name</th>
<th>Input parameter name</th>
<th>Input parameter type</th>
<th>Operation output type</th>
</tr>
</thead>
<tbody>
<tr>
<td>def: age</td>
<td>todaysDate</td>
<td>PointInTime</td>
<td>Integer</td>
</tr>
</tbody>
</table>

Expression that derives value of the age() operation output
What *def*: is Doing

*def: age()* : *Integer* is illustrated below:
Dot/Period Notation

The Dot or Period features prominently in GELLO expressions:

▷ As illustrated in the following example:

context Entity

```
self.playedRole.participation.act
```

Dots/periods
To the Right of the Dot/Period

The Dot/Period to the right of self will specify what type of result is returned. There are three possibilities with the RIM:

- `self.attributeName` would return the attribute value
  - The type of the attribute value will be set within the class entity

- `self.operationName()` will return the result of the operation
  - RIM classes do not have operations or methods
  - Many HL7 datatype classes have operations, eg minus()

- `self.associationName` will return the set of all instances of the class at the end of the Association
  - This is an instance where the dot/period notation is used as a short-hand for the `collect()` operation, which returns a collection that is a different type from the instance on the left side of the dot/period
Arrow Notation

The Arrow (\(\rightarrow\)) also features prominently in GELLO expressions:

-enabled

As illustrated in the following example:
To the Left of the Arrow (->)

The Arrow (->) is used to apply a pre-defined OCL collection operation. To the left of an arrow, therefore, will be a collection type (Set, OrderedSet, Bag, or Sequence):

 proposé In the following example:

self.playedRole.participation.act

-> select(oclIsTypeOf(Observation))

• The collection operation select() is applied to the result of the expression

self.playedRole.participation.act, which is a collection of Act and related sub-classes

• OCL collection operations are different from user-defined operations, which are applied using the Dot/Period notation
To the Right of the Arrow (->)

After an Arrow (->) will be a pre-defined OCL or GELLO collection operation:

- In the following example:
  
  \[
  \text{self.playedRole.participation.act} \\
  \rightarrow \text{select(oclIsTypeOf(Observation))}
  \]

- The collection operation \textit{select()} loops through each object in the \textit{self.playedRole.participation.act} collection and creates a new collection.
- Each object is evaluated according to the Boolean operation within the \textit{select()} brackets, viz. \textit{oclIsTypeOf(Observation)}.
- If the object is an Observation sub-class, then it will be added to the new collection that is being created by the \textit{select()} operation.