GELLO, A Practical Implementation through the Application of Real World Examples

Peter R. Tattam, B.Sc.
Andrew McIntyre, M.B.B.S. (Hons) F.R.A.C.P.
http://www.medical-objects.com.au
peter.tattam@medical-objects.com.au

Choice of GELLO for decision support

- Project to build advanced decision support and registry reporting tools for the treatment of Lymphoma.
- GLIF was vehicle for Guidelines.
- Decision was made to use GELLO encoded logic.
- Originally envisaged that Arden would be the vehicle, however further investigation suggested that GELLO would be a better candidate to evaluate the clinical data to assist relevant decisions.
- GELLO features
  - Rich querying facilities.
  - Object oriented
  - Integrates well with HL7
- Implemented what we believe to be one of the first practical implementations of GELLO worldwide.

Working with GELLO specifications

- GELLO is a work in progress
- Developed in coordination with HL7 Decision Support Group
- Based on OCL 2.0
- Started in Dec 2003
- Most recent draft of specification dated May 2005
- Mailing list started Dec 2006 with active discussion
- Implementation raised many issues with specifications

Limitations with GELLO Language and Grammar

- Typographical errors
- Incomplete language elements
- Incorrect language elements
- Ambiguous constructs
- Discrepancies between grammar and examples used
- Semantic limitations of the language
- Typically formal grammar and actual grammar differ in practice due to implementation details
- Even so, formal grammar in specification is incorrect, incomplete and does not even parse the examples in the specification

Minor Corrections

- Had to cut & paste grammar from HTML document
- Built a tool to process the BNF into a useful form
- Found syntax errors in the BNF and corrected.
    - Misspellings
        - Fixed by inferring correct names.
    - Undefined and unused symbols using reachability analysis
        - Symbols “GELLOType”, “ReferenceToClass” undefined
            - Fixed by changing “GELLOTypes” to “GelloType”, and adding “ReferenceToClass” to point to “ReferenceToInstance”
        - Symbols <IMPLIES>, <NEW>, <ENDCONTEXT> unused
            - Fixed by including <IMPLIES> in “ConditionalExpression”, and omitting <NEW> and <ENDCONTEXT>
    - Syntax errors in terminal regular expressions
        - Fixed
    - Fixed errors in some of the terminal regular expressions
        - <DECIMAL_LITERAL> only generated numbers without digit “0”!!
        - <REAL_LITERAL> is ambiguous with <INTEGER_LITERAL>
        - <NUMBER> was removed by simplifying grammar.

Completing the Language

- Various Elements in language appear to be stubs
- Referred back to OCL to figure out what elements should look like
- Elements fleshed out
  - "CollectionLiteral" defined

```
CollectionLiteral::=  CollectionType "{" ( CollectionLiteralElement ( ","
    CollectionLiteralElement )* )? "}"
CollectionLiteralElement::= Expression ( ".." Expression )?
```

  - "TupleLiteral" defined

```
TupleLiteral::= <TUPLE> "{" TupleLiteralElement ( "," TupleLiteralElement )* "}"
TupleLiteralElement::= <ID> ":" GELLOType "=" Expression
```

  - "EnumerationType" extended

```
EnumerationType::=     <ENUM> "(" <ID> ( "," <ID> )* ")"
```

  - "CollectionType" extended

```
GELLOType::=     BasicType
               |  CollectionType "(" GELLOType ")"
               |  TupleType
               |  EnumerationType
```

Trivial Extensions to language

- Added comments
  - // A comment to end of line.
  - /* A comment which is
     more than one line.
    */
- Allow “ to be used synonymously with ‘ for strings
- Generalized parameters to standard functions to be “Expression”s rather than specific typed literals.
- Allow identifiers to be case insensitive.

Significant Enhancements to Language

- Many of the examples refer to lists of Statements rather than a single GELLO Expression or Statement.
- Based on implementation experience and recent discussions on the mailing list, a significant extension to allow for multiple declarative statements to be specified.
- These issues were resolved by the following constructs:
  - Introduction of “Block” construct.

```
GELLOExpression::=     Block
Block::=               Declarative* ExpressionNP
Declarative::=            LetStatement
                       |  ContextNavigationStatement
```

  - Redefining “IfStatement” and “ComparisonExpression” and introducing “IfExpression”

```
IfStatement::=     <IF> Expression <THEN> Block <ELSE> Block <ENDIF>
ComparisonExpression::=                                IfExpression (<EQUAL> IfExpression |
                          <NEQ> IfExpression | <LT> IfExpression  |
                          <LEQ> IfExpression | <GT>  IfExpression  |
                          <GEQ> IfExpression)*
IfExpression::=             AddExpression
                       | IfStatement
```

- Resolution of no statement separator
  - The introduction of multiple statements introduced a difficulty in the grammar in that statements do not have a terminator or separator (e.g. “;”).
  - Problem occurs when two GELLO expressions appear next to each other within the language.
  - Resolved by restricted form of Expression in the grammar
- Included the [and] operators to index into collections.
  - By reference to OCL V2.0
  - Shorthand method for the ElemAt() collection operator

Unambiguous Grammar Constructs

- Many of the constructs as defined in the original specification result in a highly ambiguous grammar.
- Constructs which look superficially correct for descriptive purposes end up generating an ambiguous grammar.
- The importance of an unambiguous grammar is two-fold
  - Being able to specify the language in a portable way to a wide range of users and implementers
  - Being able to generate practical parsers for the language
- A great deal of time was spent trying to resolve the ambiguous nature of the GELLO language as specified by the original grammar.
- The general nature of the ambiguities fell into several categories
  - Places where one construct overloaded another.
    - E.g. when a “Name” and an <ID> were derivable in the same place.
  - Places where one construct next to another resulted in an ambiguity i.e. when an “Expression” appeared next to another “Expression”. These two examples are identical syntactically but have different meaning
    - Example 1.

      Let A: Integer = C + D

      (A * 20)
    - Example 2.

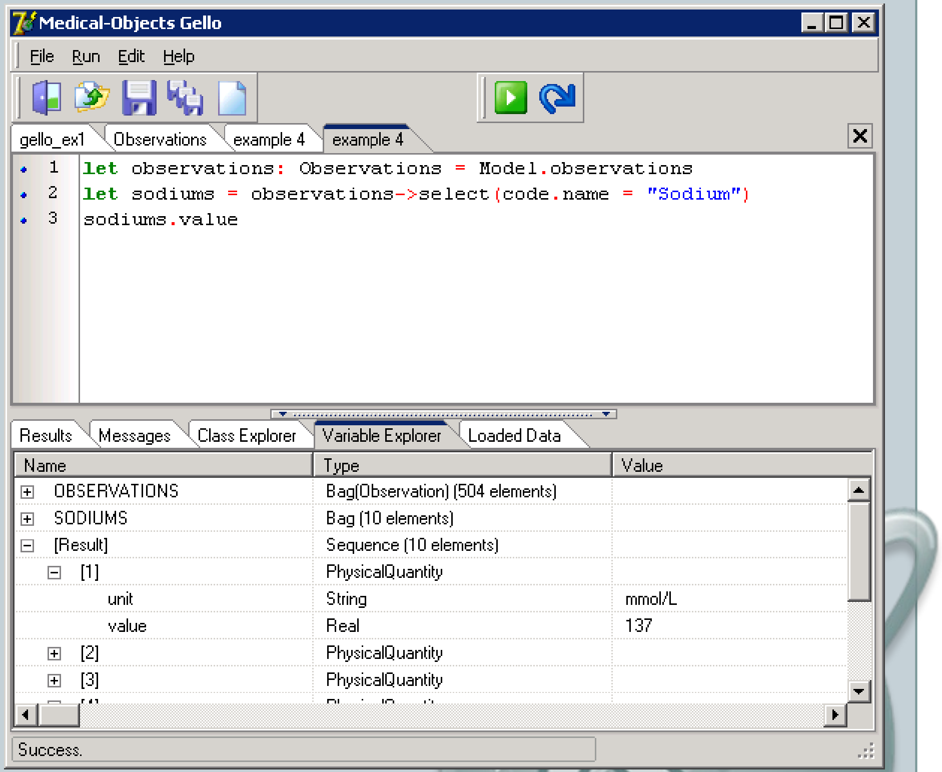
      Let A: Integer = C + D(A *20)
- The changes to resolve the ambiguities were many and varied. The more significant of these are
  - A restricted form of “Expression”, “ExpressionNP” which does not start with “(“, “+” or “-”.
  - Introducing the “Operand” construct from which variable references, attribute references, method operators and collection operators are formed.

Rich HL7 infrastructure

- The GELLO and GLIF modules have been built to operate over a Rich HL7 infrastructure developed over many years by Medical-Objects
- HL7 version 3 Data Model (RIM) is incorporated into GELLO
  - Observation
  - Patient
  - Medication
- Model data and GELLO results visible from IDE
- Can dynamically bind Model data to GELLO infrastructure
- Uses Windows COM to manage data ownership.
- Also includes SNOMED engine access.
- Supports concept of Medical Logic Modules (MLM) as in Arden

Embedded GELLO

- The GELLO as implemented by MO-GELLO has been developed as an embedded component within a GLIF and Archetypes framework.
- It was developed using a LALR(1) parser framework in conjunction with a Delphi Object Pascal HL7 framework.
- It is interpretive in nature.
- Gello expressions are compiled at run time and stored as an internal object oriented expression tree.
- Execution speed is facilitated by the use of object oriented techniques.
- There is no “byte code” to execute, all calls are made natively to the HL7 framework.
- GELLO expressions can be implemented using an embedded IDE called “Mowgli”.
- Library facilities have been developed whereby frequently used GELLO expressions can be run indirectly from within another GELLO expression



```
let observations: Observations = Model.observations
let sodiums = observations->select(code.name = "Sodium")
sodiums.value
```

Name	Type	Value
⊞ OBSERVATIONS	Bag(Observation) (504 elements)	
⊞ SODIUMS	Bag (10 elements)	
⊟ [Result]	Sequence (10 elements)	
⊟ [1]	PhysicalQuantity	
unit	String	mmol/L
value	Real	137
⊞ [2]	PhysicalQuantity	
⊞ [3]	PhysicalQuantity	

Success.

References

- The GELLO Specification
- Medical-Objects GELLO
- Original GELLO Grammar
- Revised GELLO Grammar