Data Model/Ontology:

- **Outline**

To better address the needs of data model and ontology by guideline, we will organize the data model into 3 layers:

  - Core GLIF

    Core GLIF deals with the data modeling and ontology issues that essential to the GLIF standard. It defines how medical data and concept should be represented and referenced by GLIF. It also defines the scope of data items and how the data items acquire their value. The core GLIF makes certain assumptions of the RIM and ontology though it does not require a particular RIM or ontology.

  - GLIF Reference Information Model (RIM)

    The RIM model provides a class hierarchy for medical concepts that represent different classes of medical data. The hierarchy allows us to specify the attributes of each class of medical data in an object-oriented fashion. It also makes it possible to do type check for GLIF. The RIM assumes the existence of a term dictionary that provides mapping from terms to concepts that are referred to by the class hierarchy.

    Different from core GLIF, users have an option to use the GLIF RIM model or not. However, unless different users are using the same RIM models, it will very hard for them to share with each other any reference to patient data in a structure way. Since we do not assume the every user has a RIM model for their guideline authoring and execution needs, we are developing GLIF RIM.

  - Medical Ontology

    The medical ontology for GLIF should contain a term dictionary and a medical knowledge base. How to code data is beyond the scope of a guideline format, so a user can choose to use any ontology with GLIF. However, without sharing a common term dictionary, users may not be able to share the meaning of the criteria and actions in structured way. So it is very important for users to have a common term dictionary and we found UMLS is very appropriate for that purpose.

    The knowledge base should contain a concept hierarchy that can be viewed as the instantiation of the class hierarchy in the RIM. The more medical knowledge we have, the better we can assist users in guideline authoring and execution. To acquire and represent medical knowledge is a huge research area by itself. What we want to achieve is to study what types of knowledge are needed and demonstrate how they can be used by GLIF.

- **Core GLIF**

  - **Assumption of RIM and ontology**

    RIM should have a class hierarchy that organizes medical concepts into classes. It should also provide a data model for each of the class. Ontology should have a term dictionary that map ascii strings to medical concepts. The RIM and term dictionary should refer to the same set of concepts.

  - **Data Item**

    Data Item is a symbol that can have mutable values. (Similar to variable in programming languages) A data item contains six attributes: name of the data item, owner of the data item, the RIM model it uses, concept which is referred to by the data item, the ontology it uses and the data value of the data item.

    Data Item {

        Name: String

        Owner: String

RIM Model: String

Referring Concept: code of concept

Ontology: String

Data Value: List of instances of the concept object

}

Referring to the same concept, users may choose different names for a data item. For example, in medical record, “sex” and “gender” can refer to the same concept. Also, sometimes, a user may want to use a abbreviation instead of the whole concept name.

Owner of the data item refers to for which patient a piece of data is collected. The reason to specify this is because sometimes even in one guideline, data from multiple patient will be mentioned although most guideline we have seen so far are specific for one patient. For example, clinical trial guidelines sometimes refer to a group of patients. For individual patients, we may be able to use some kind of MRN to distinguish one patient from another. For groups of patient, since we may not know or need to know each patient’s identity, we propose to use free text to describe such groups.

Each data item should refer to some concept. The concept can represent simple data types such as integer or it can be more complex types like “medication order”. The concepts will be referred to by their codes. The term dictionary in the medical ontology will determine the code of a concept.

Data value of a data item will be modeled by a list of instances of data model objects in the RIM model. The data model object of the instance is that of the concept the data item refers to. For example, a data item “albumin” may refer to “Serum Albumin Test Result”. So the data value is list of instances of “Serum Albumin Test Result” object in the RIM.

Example:

In the criteria “Albumin < 1.3”, Albumin is a Data Item. An instance of the object of albumin looks like:

```
{
        Name: Albumin
        Owner: MRN00001
        RIM Model: GLIF
        Referring Concept: C0034627 (Serum Albumin Test Result)
        Ontology: UMLS
        Data Value: {
                {BWH Serum Albumin, 1.5, mg/ml, 01/15/1999}
                {MGH Serum Albumin, 1.7, mg/ml, 04/18/1999}
                …………..
        }
}
```

- **Literal**
  - Literal is a symbol that can have fix values. (Similar to constant in programming languages) It has five attributes: name of the Literal, name of the RIM Model it uses, the concept the Literal refers to, the ontology it uses and the Data Value.

Most attributes of the literal are similar to those of the data item. However, literal can not have more than one instance and the instance can not have an owner. In some cases, a represents a concept and it will not have any instance. For example, when in the expression “diagnosis == heart disease”, heart disease should have no instance. However, in some cases, a literal represents an instance of a concept instead of the concept itself. So, in such cases, a literal can have a data value of one instance. For example, in “height > 5 foot 4 inch” 5 foot 4 inch refers to a concept “height”, but it is an instance of the concept instead of the concept itself.

```
Literal {
        Name: String
        RIM Model: String
        Referring Concept: Pointer to or code of concept
        Ontology: String
        Data Value: One or zero instance of the concept object
}
```

Example:

In the criteria “Diagnosis == CHF”, CHF is a literal and it is represented like:

```
{
    Name: CHF
    RIM Model: GLIF
    Referring Concept: C0003429 (Congestive Heart Failure)
    Ontology: UMLS
    Data Value: {}
}
```

In the criteria “Height > 5 foot 4 inch”, 5 foot 4 inch is a literal and it is represented like:

```
{
    Name: 5 foot 4 inch
    RIM Model: GLIF
    Referring Concept: C0005937 (Height)
    Ontology: UMLS
    Data Value:
            {
            {5, foot, 4, inch,…..}
            }
}
```

- **Default Model in the absence of RIM and Ontology**

  Not all terms may be mapped to concept and all concepts have data model. When a term failed to be mapped to a concept, the “Referring Concept” is automatically assigned the value “UNKOWN” and the type for the field “Data Value” is assigned “string”. When a concept does not have a data model, the type for the field “Data Value” is assigned “string”.

- **Scope of Data Item**

  There might be needs to pass data values between a sub-guideline and a guideline. What we proposed here is following the principal of some programming languages use in sharing variables and their values.

  - By default, data item are not shared between guideline and sub-guideline
  - Each sub-guideline has a parameter list.
  - For each parameter in the parameter, specify its permitted passing direction (IN, IN/OUT). IN should be the default.

  Peter have given this issue more thought and here are his opinions:

  “Block Structured:

  Under this interpretation, sub-guidelines are used to manage complexity. Only the most general of steps exist at the top level. Further refinement of the step(s) are offered as one delves deeper into the guideline.

  In this case, there is an implicit declaration of variables and little need for parameter passing. Any information gathered in a higher level would be visible to the sub-guideline. The only major difficulty arises from a Choice sub-guideline in which the preferences of the various choices are influenced by the activities in the sub-guideline. If this is truly the only scoping problem for the block approach, then one could accept the limitation of modifying the parameters that determine the choice preference and not the preferences directly.

  Procedure Call:

  Under this interpretation, sub-guidelines are used to allow the use of multiple separate guidelines, which reference each other. Thus, referencing the sub-guideline transfers control from one guideline to another.

  As an aside, this interpretation does not seem to mesh well with Samson's notion of a State and its associated Clinical guideline. (I think these were the terms he used).

  With a procedure call, there may need to be parameter passing. However, this assumes that authors are writing guidelines that they expect to be called by an external guideline. Whereas it is quite probable that guidelines will be constructed that reference each other, I am not sure that the authors will want to be bound to an explicit parameter list. Any information that needed to be passed between guidelines could be passed via the patient record. The 'curly braces' would be responsible for semantic matching (e.g., from BloodPressure in one guideline to BP in another).

  If both interpretations are intended, we might want to consider naming the second reference ExternalGuideline or something similar to indicate that it is not a smaller piece of the larger guideline, but instead a reference to someone else's guideline.”

- **Get Statement**

  We have not reached an agreement on “get” statement. The following is Qing’s opinion on why having a get statement.

  - **Consideration**: The creation/use of a variable and retrieving/assigning value to it should be separated. The reason for this is that there are times people want to use a variable without retrieving new values. For example, throughout a guideline, weight and gender may be used in multiple decision steps. Value of weight may or may not be retrieved more than once depending on the guideline, gender most likely needs to be retrieved only once.

- **Possible Way of Implementation**:
  - Upon defining a new variable, a "get" statement should be generated.
  - Each time the variable is referred to, a user can choose to get new value or not.
  - A get Statement contains a Data Item, Data Source, and Temporal Constraint.

Bob's comment on this is as follows:

" Guideline may also want specify where to get a datum (e.g. stored data) from patient, from family, derived/calculated, nurse, MD, etc; and temporal constrains , etc. – on the datum, not in the logical condition.

May want to explicitly model user interaction in terms of elements of a data entry: form, flow sheet, or an encounter screen."

Mor summarized the discussion between us:

"The get statement is used explicitly to retrieve a value from the database. This is useful if every time a variable is referenced by a criterion you can decide whether you want to get the latest query result for that variable from the database, or you can use the last value held by this variable. This is useful for comparing the new value with the previous value, without creating two variables for that purpose, or for using a variable like "risk level" that takes a lot of time to compute by several criteria without the need to recalculate "risk level".On possibility is that the author of the guideline uses the "get" statement explicitly in order to specify that he wants to get the variable reevaluated, and by default, the variable will use it's latest retrieved value (the variable will always implicitly call the get statement when it is first encountered).

Another possibility is not to put the burden of deciding where to use the get statement on the guideline author. We can think of the situation in the following way. Getting the variable reevaluated by default can always be done whenever the variable is encountered. It is an optimization decision not to reevaluate a variable, and need not concern the guideline author. When we want to use a previously computed result (like "risk level") we can use the assignment action specification to assign the computed result to a variable that is not mapped to a database query (does not have an associated get function that queries that database). Then, when this variable is used in a criterion, it holds the computed result that we wish to use."

- First Order Predicate Logic

  The use of first predicate logic, especially the use of "for all" and "there exist" for scoping has been discussed in the context of expressions. We understand that sometimes users may want to use such quantifiers. For ontology, it poses some questions. For example: should "for all medications" be viewed as one data item or multiple data items? How about "contraindication between all medications and all medical conditions"? Should contraindication be viewed as a data item? How to express that contraindication should be computed over medication and medical conditions.

  What we propose is to first only formally represent in a expression what we can evaluated. It is very difficult to evaluate contraindication in an expression and we do want our GLIF compiler and run-time engine to be able to evaluate an expression. We can model the whole phrase "contraindication between all medications and all medical conditions" as a variable name and let user figure out how to compute it. If we can not compute "latest", we should view "lastest sodium" as one data item. Otherwise, we can define "sodium" as a data item and perform the "latest" operation on it.

From the example of contraindication, we can see that to run GLIF users may always have perform computation and retrieval tasks outside the control of GLIF. These tasks need to have a well-defined interface with the GLIF steps and the “get” statement is designed for that purpose.

- **GLIF RIM**
    - Semantic Hierarchy
        - UMLS
    - Data Model for Concept Class
        - HL7 RIM, LOINC, and etc..

Example:

```
public class Term {

   public string name;

}



public class Concept extends Term {

   public string UI;

   public string Definition;

}



public class Event extends Concept {

   public Timestamp Time;

}

public class Activity extends Event {

}
```

```
public class Occupational_Activity extends Activity {

}


public class Healthcare_Activity extends Occupational_Activity {

}

public class Lab_Procedure extends Healthcare_Activity {

    public UI Analyte;

    public UI Property_Measured;

    public UI Time;

    public UI Sample;

    public UI Method;

    public UI Scale;

}

public class Serum_Albumin_Test extends Lab_Procedure {

}

public class Entity extends Concept {

}


public class Conceptual_Entity extends Entity {

}
```

```
public class Finding extends Conceptual_Entity {

}

public class Lab_or_Test_Result extends Finding {

    public UI Result_of;

    public Value Val;

    public Value_types Val_type;

    public Unit Unit;

    public Timestamp Time;


    /** @link dependency */

    /*#Lab_Procedure lnkLab_Procedure;*/

}

public class Serum_Albumin_Test_Result extends Lab_or_Test_Result {

    /** @link dependency */

    /*#Serum_Albumin_Test lnkSerum_Albumin_Test;*/

}
```

- **Medical Ontology**
    - Term Dictionary

- Medical Knowledge

  Example:

```
Woman {
    Isa: Person
    Gender: Female
     ………
}

Serum Albumin Test Result {
            ……
            Normal Range: {0, 10}
            …….
}
```