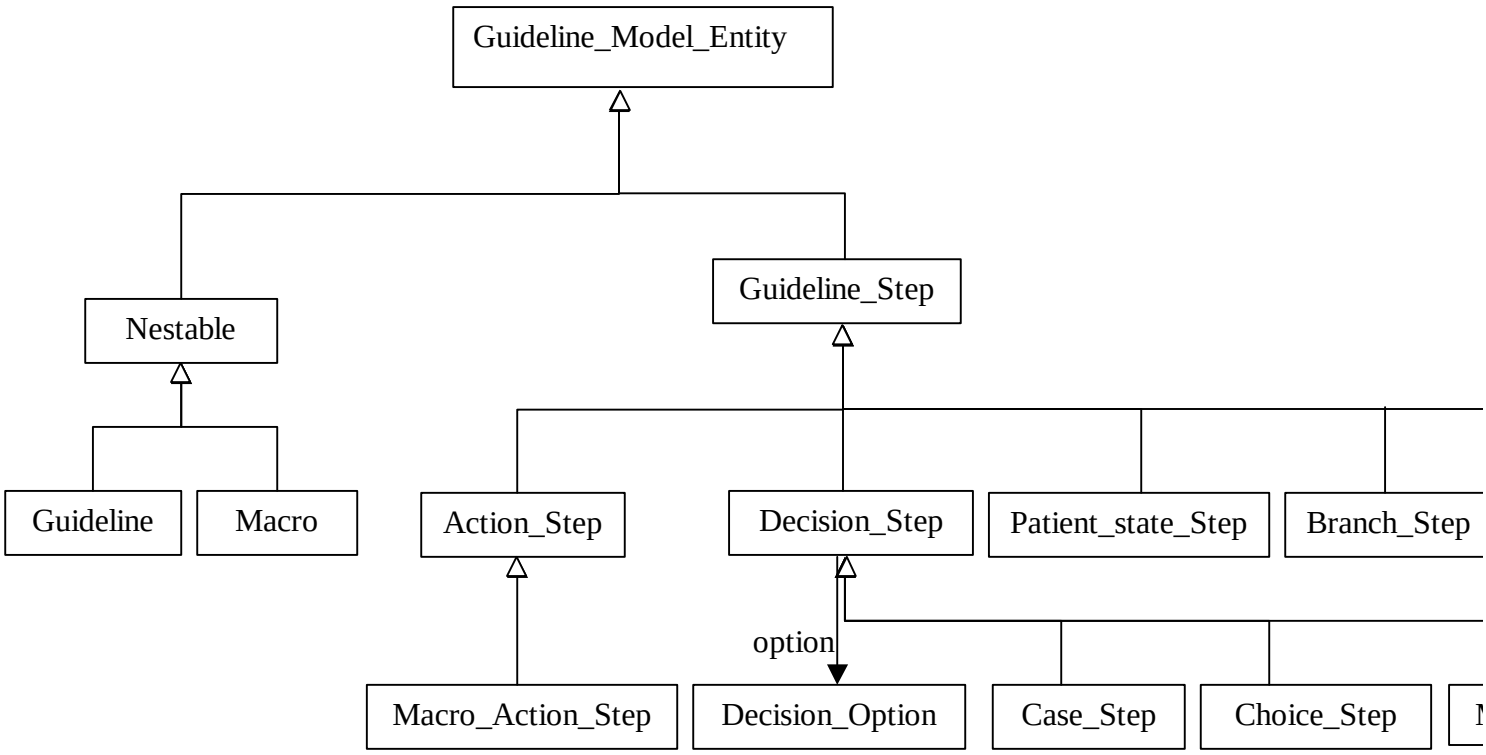
1. Introduction

1.1 Purpose of document

1.2 What is GLIF?

GLIF3 is a methodology that enables modeling and representation of guidelines at three levels of abstraction: a conceptual flowchart that is easy to author and comprehend, a computable specification that can be verified for logical consistency and completeness, and an implementable specification that can be incorporated into particular institutional information systems.

The GLIF3 model is object-oriented. It consists of classes, their attributes, and the relationships among the classes, which are necessary to model clinical guidelines. The model is described using Unified Modeling Language (UML) class diagrams [Object Management Group, 1999 #46]. Additional constraints on represented concepts are being specified in the Object Constraint Language (OCL), a part of the UML standard.


Guideline_Model_Entity
Nestable
Guideline_Step
Guideline
Macro
Action_Step
Decision_Step
Patient_state_Step
Branch_Step
option
Macro_Action_Step
Decision_Option
Case_Step
Choice_Step

2. Overview of GLIF

2.1 Scope of GLIF

2.2 Bird's eye view of GLIF

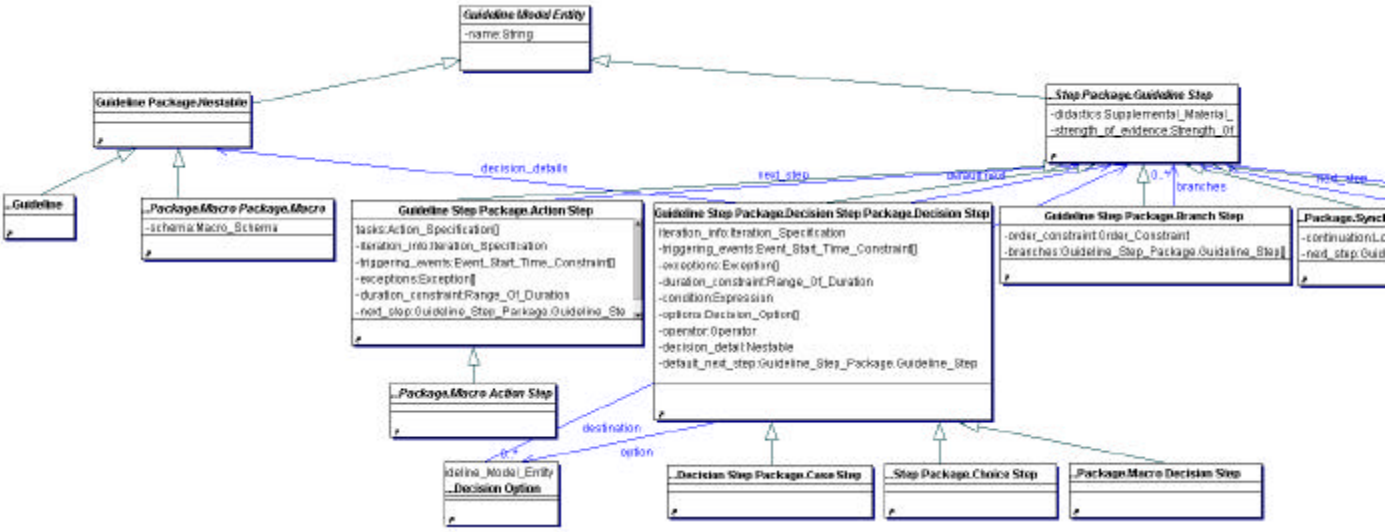
Subpackages

package Data
package Guideline_Package
package Guideline_Step_Package
package Expression_Package
package Temporal_Expression_Package
package Supplemental_Material_Package
package Action_Specification_Package
package Iteration_Package
package EventsAndExCeptions
package Nesting
package Views_Package

Classes

class Guideline_Model_Entity

Class Diagram



2.3 Understanding the domain ontology

Domain Ontology

Logical expressions (criteria) and action specifications reference medical terms. These terms are formally defined in the medical domain ontology. The support of the ontological needs for guideline modeling is separated into three layers. The first layer, **Core GLIF**, is part of the GLIF specification language. It defines a standard interface to medical data and concepts, and to the relationships among them.

The second layer, **Reference Information Model (RIM)**, is essential for guideline execution and data sharing among different applications and different institutions. It defines the basic data model for representing medical information needed in specifying protocols and guidelines. It includes high level classification concepts, such as drugs and observations about a patient, and attributes, such as units of a measurement and dosage for a drug, that medical concepts and medical data may have.

The third layer, **Medical Knowledge layer**, contains a term dictionary and a medical knowledge base. Although a medical knowledge base is very desirable, we can only make limited assumptions about its existence because of its huge scope

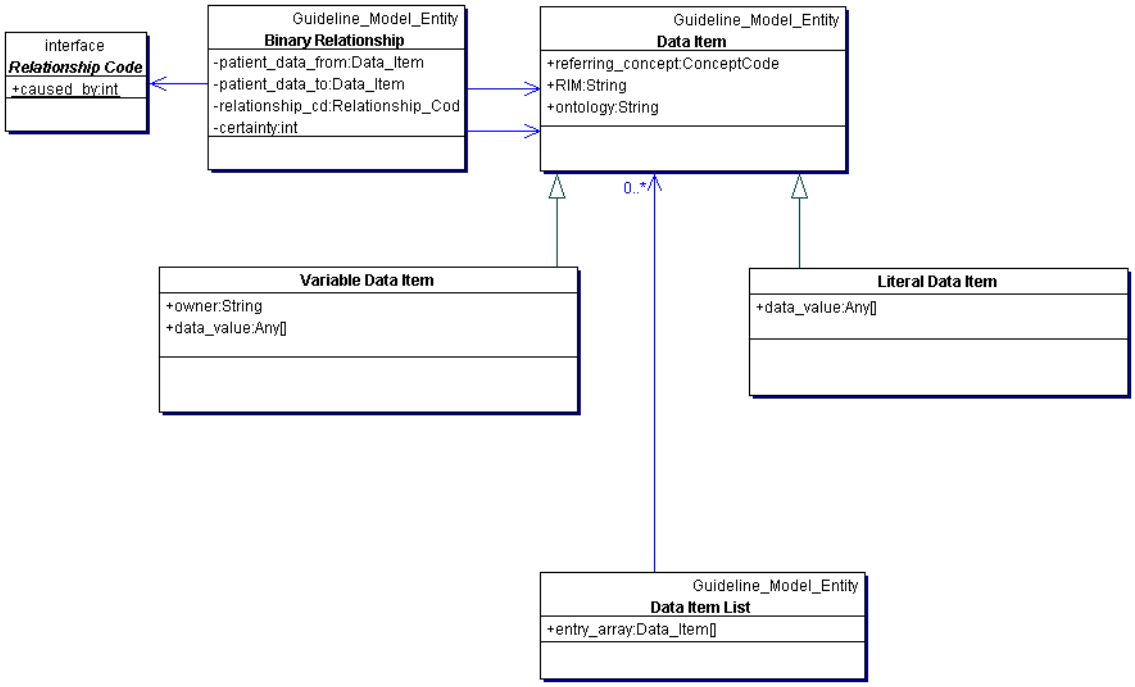
When all three layers are involved, they work closely together: Core GLIF relies on the RIM to supply the attributes of the medical concepts and to represent data values. Core GLIF relies on the Medical Knowledge layer for encoding specific medical concepts.

In the three-layered medical ontology, users have the freedom to choose a particular RIM and a particular medical knowledge layer that fits their needs.

See also sections 4.2 and 4.3 for specifying decision criteria and patient data.

Package *Core_GLIF*

Class Diagram



Core GLIF defines how medical data and concepts should be referenced by GLIF. It also defines the scope of data items and how the data items acquire their values. Core GLIF is part of the GLIF specification.

Core GLIF views all medical terms and concepts as data items. Data items are symbols that may have values. GLIF need to employ data items to describe clinical decisions and actions. Data items represent data that need to be acquired from external sources, such as patient's weight, or data that are computed or assigned by guidelines, such as risk level and creatinine clearance. The data item values conform to the data objects whose structure is defined by the RIM layer of the medical ontology.

Each data item has a name, a RIM identifier indicating the RIM it uses, a Medical Knowledge identifier indicating the Medical Knowledge ontology it uses, and the code of a concept to which the data item refers, that should be also defined by the third layer in order to enable mapping to an EPR. By specifying the RIM and Medical Knowledge ontology at the level of an individual data item, we allow one guideline to refer to multiple RIMs and Medical Knowledge ontologies. For ease of use, a default RIM and Medical Knowledge ontology can be defined.

Core GLIF distinguishes between two types of data items: literals (constants) and variables. A **variable data item** can have mutable values. A variable data item has an owner and a data value attribute, which is modeled as a list of instances of a data object defined by the RIM layer. Patient's height, weight, gender, and age are variable data items. Owner indicates who the patient is when patient data or other data specific to a particular patient. 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. When no owner is specified, this attribute will take the default value of "unknown". Otherwise the value of the attribute will be the identifier of the owner.

A **literal data item** is a data item that has a fixed value. It is similar to a constant in programming languages. Unlike a variable data item, a literal does not have an owner and its data value is modeled by an empty list or a list of exactly one instance of the data object defined by the RIM layer. Congestive heart failure, female, smoker, and TSH test order are all examples of literal data items.

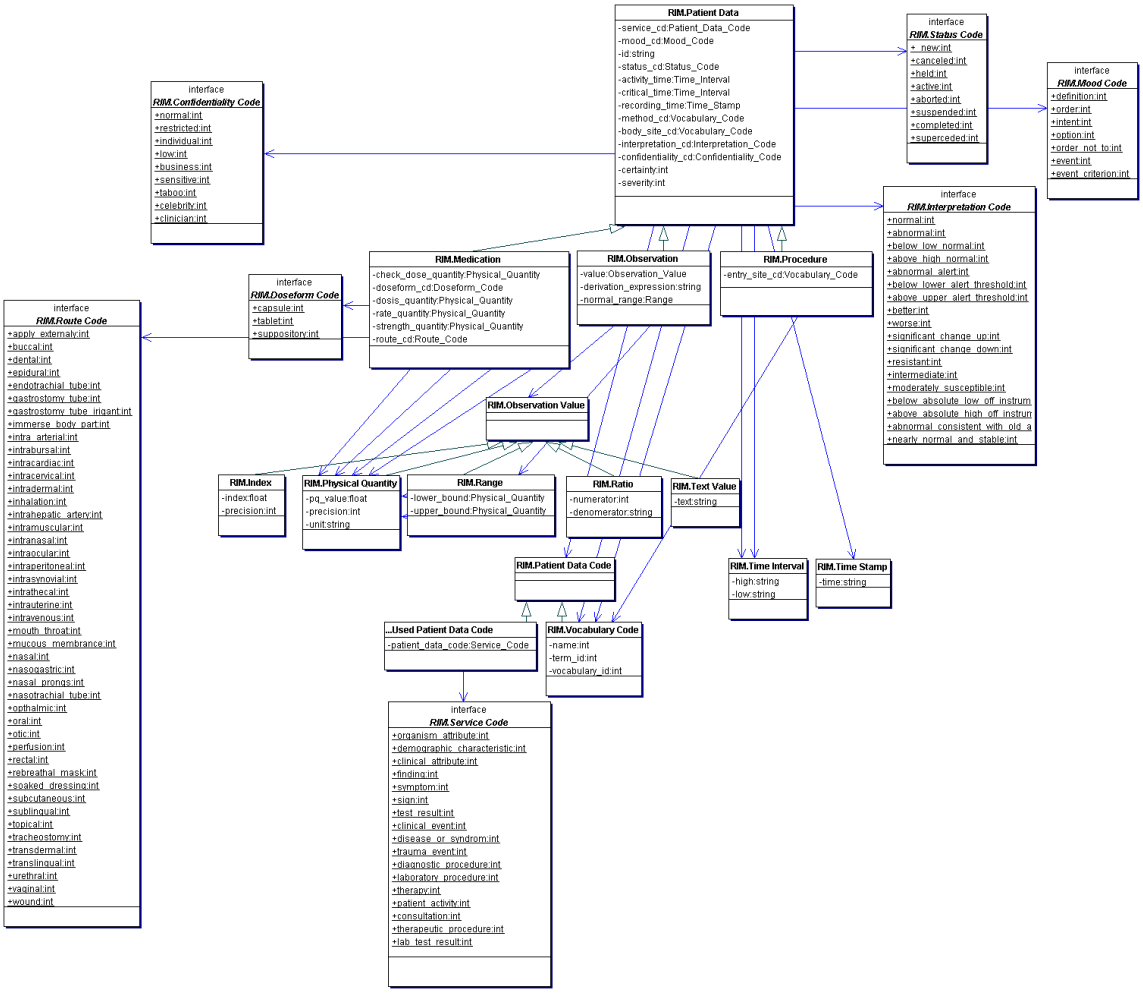
The values of variable data items can be assigned at execution time and the values of literal data items can be assigned at authoring or execution time.

A **binary relationship** is a relationship between two data items. Each binary relationship has an associated code (e.g., “caused_by”) and a certainty attribute that expresses how sure we are that the relationship holds.

Core GLIF does not require the existence of RIM and Medical Knowledge layers. When RIM and Medical Knowledge are not available, the data rep-resentation is handled by a default model. However, we do expect users to employ RIM and Medical Knowledge layers for level B and C guideline representation.

When users simply want to create a human-readable guideline, a RIM or term dictionary might not be needed. Even when a RIM and a term dictionary are provided, not all terms may be mapped to concepts nor all concepts have a data model. When both RIM and Medical Knowledge ontology are absent, the RIM and Medical Knowledge ontology identifier fields in Core GLIF data items are marked as “UNKOWN”. When a term fails to be mapped to a concept, the referring concept is automatically as-signed the value “UNKOWN”. When a concept or term does not have a data model specified by the RIM, the type for each instance of data value is assigned a type “string”.

Package *RIM*


RIM.Patient Data
-service_cd:Patient_Data_Code
-mood_cd:Mood_Code
-id:string
-status_cd:Status_Code
-activity_time:Time_Interval
-critical_time:Time_Interval
-recording_time:Time_Stamp
-method_cd:Vocabulary_Code
-body_site_cd:Vocabulary_Code
-interpretation_cd:Interpretation_Code
-confidentiality_cd:Confidentiality_Code
-certainty:int
-severity:int
interface
RIM.Status Code
+_new:int
+canceled:int
+held:int
+active:int
+aborted:int
+suspended:int
+completed:int
+superceded:int
interface
RIM.Mood Code
+definition:int
+order:int
+intent:int
+option:int
+order_not_to:int
+event:int
+event_criterion:int
interface
RIM.Confidentiality Code
+normal:int
+restricted:int
+individual:int
+low:int
+business:int
+sensitive:int
+taboo:int
+celebrity:int
+clinician:int
interface
RIM.Interpretation Code
+normal:int
+abnormal:int
+below_low_normal:int
+above_high_normal:int
+abnormal_alert:int
+below_lower_alert_threshold:int
+above_upper_alert_threshold:int
+better:int
+worse:int
+significant_change_up:int
+significant_change_down:int
+resistant:int
+intermediate:int
+moderately_susceptible:int
+below_absolute_low_off_instrum
+above_absolute_high_off_instrum
+abnormal_consistent_with_old_a
+nearly_normal_and_stable:int
RIM.Medication
-check_dose_quantity:Physical_Quantity
-doseform_cd:Doseform_Code
-dosis_quantity:Physical_Quantity
-rate_quantity:Physical_Quantity
-strength_quantity:Physical_Quantity
-route_cd:Route_Code
RIM.Observation
-value:Observation_Value
-derivation_expression:string
-normal_range:Range
RIM.Procedure
-entry_site_cd:Vocabulary_Code
interface
RIM.Doseform Code
+capsule:int
+tablet:int
+suppository:int
interface
RIM.Route Code
+apply_externaly:int
+buccal:int
+dental:int
+epidural:int
+endotrachial_tube:int
+gastrostomy_tube:int
+gastrostomy_tube_irrigant:int
+immerse_body_part:int
+intra_arterial:int
+intrabursal:int
+intracardiac:int
+intracervical:int
+intradermal:int
+inhalation:int
+intrahepatic_artery:int
+intramuscular:int
+intranasal:int
+intraocular:int
+intraperitoneal:int
+intrasynovial:int
+intrathecal:int
+intrauterine:int
+intravenous:int
+mouth_throat:int
+mucous_membrance:int
+nasal:int
+nasogastric:int
+nasal_prongs:int
+nasotrachial_tube:int
+opthalmic:int
+oral:int
+otic:int
+perfusion:int
+rectal:int
+rebreathal_mask:int
+soaked_dressing:int
+subcutaneous:int
+sublingual:int
+topical:int
+tracheostomy:int
+transdermal:int
+translingual:int
+urethral:int
+vaginal:int
+wound:int
RIM.Observation Value
RIM.Index
-index:float
-precision:int
RIM.Physical Quantity
-pq_value:float
-precision:int
-unit:string
RIM.Range
-lower_bound:Physical_Quantity
-upper_bound:Physical_Quantity
RIM.Ratio
-numerator:int
-denomerator:string
RIM.Text Value
-text:string
RIM.Patient Data Code
RIM.Time Interval
-high:string
-low:string
RIM.Time Stamp
-time:string
...Used Patient Data Code
-patient_data_code:Service_Code
RIM.Vocabulary Code
-name:int
-term_id:int
-vocabulary_id:int
interface
RIM.Service Code
+organism_attribute:int
+demographic_characteristic:int
+clinical_attribute:int
+finding:int
+symptom:int
+sign:int
+test_result:int
+clinical_event:int
+disease_or_syndrom:int
+trauma_event:int
+diagnostic_procedure:int
+laboratory_procedure:int
+therapy:int
+patient_activity:int
+consultation:int
+therapeutic_procedure:int
+lab_test_result:int

For a RIM to be acceptable for use by GLIF, it should have a class hierarchy that organizes medical concepts into classes. For each class, the RIM should provide a data model that defines the attributes or fields of the data objects that belong to that class.

The Health Level 7 (HL7) RIM and foundational models in SNOMED-RT are examples of RIMs.

The class hierarchy allows the medical concepts to be assigned to appropriate classes, although it does not necessarily perform the actual classification for concepts. For example, with a class hierarchy, one can classify a digoxin order as a medication order and a sodium test as an observation event.

Developing a RIM requires much effort. We intend to provide a default GLIF RIM with basic data types. In our study, we found both the HL7 Unified Medical Language System (UMLS) semantic net promising. Some adjustments were done. We decided to use USAM to model patient data concepts. We still haven't decided whether we would use USAM to model all Medical knowldge concepts (e.g., contraindications).

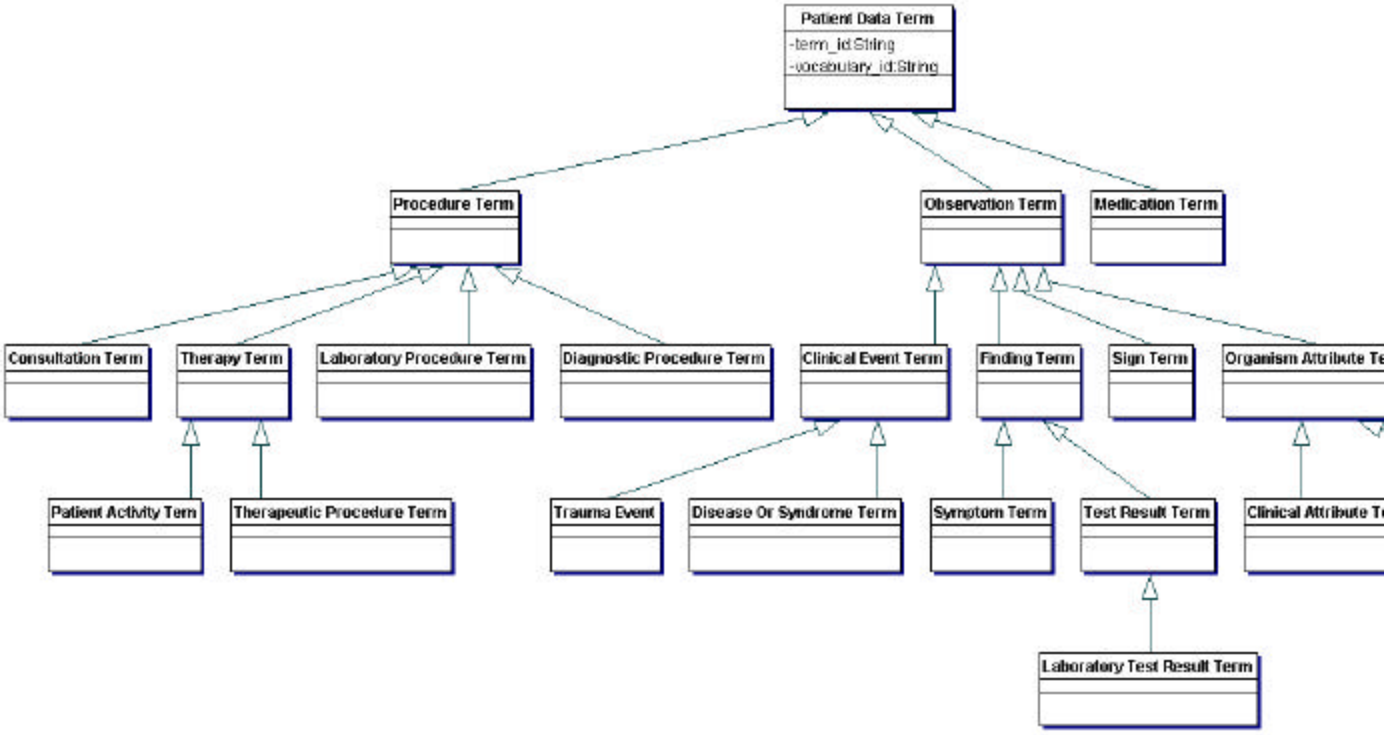
We are not utilizing all of the classes and attributes that are defined in USAM, since we have a different approach for modeling them in GLIF. Specifically, we are only using the service (patient data, medication, observation, and procedure classes). Also, we are not representing the following attributes of the service (patient data) class: max_repeat_number, interruptible_ind, substitution_cd, priority_cd, and orderable_indication.

We added two attributes to the Patinet_Data (Serive) class: severity, and certainty that we found lacking from USAM's Service class.

We haven’t yet decided how we will represent service relationships. The current version represents relationships as a pointer from one service to another. We are considering representing relationships as objects that point to the two services involved in the relationship.

Theoretically, a RIM does not require the existence of a term dictionary. However, it is likely that some kind of term dictionary will be used with the RIM. When a term dictionary is used, it is assumed that the RIM hierarchy can be applied to the concepts in the term dictionary.

Package *Third*



The Medical Knowledge ontology should provide a term dictionary that maps text strings to medical concepts. When a term dictionary is used, it is assumed that the RIM hierarchy can be applied to the concepts in the term dictionary.

The medical knowledge ontology that is defined here works with the USAM RIM. Each concept (patient data term) has an id and the id of the vocabulary from which it came. The patient data terms are classified into procedures, medications, and observations, that correspond to USAM's services. These classes are further specialized. A specific patient data term is defined as a subclass of one of the medical knowledge ontology classes.

2.4 Levels of abstraction

GLIF3 supports modeling and representation of clinical guidelines in three levels of abstraction. When a guideline is first authored, a conceptual level or representation is created. The guideline author can concentrate on conceptualizing a guideline as a flowchart of temporally ordered steps. These steps represent clinical decisions and actions or patient states. Concurrency is modeled using branch and synchronization steps. At this level of abstraction, the author is not concerned with formally specifying details, such as decision criteria, relevant patient data, and iteration information that must be provided to make the specification computable. This is exactly the purpose of the computable level of abstraction. We are intending to create tools that will aid in validation and simulation of guidelines that are specified at the computable level. Before guidelines can be incorporated into institutional information systems, mapping to institutional procedures and electronic medical records should be made. This mapping information is represented in the implmentable level. The implementable level has not yet been defined by InterMed.

The different levels of abstraction are specified by different attributes of the GLIF classes. For example, a decision criteria (`Criterion`) has a *name* attribute that gives an English sentence that describes the criterion in free text (*e.g.*, significant sciatica present for more than 4 weeks) and also a *specification* attribute that formally defines the criterion using a superset of Arden Syntax (e.g., `Lower back pain.instance. Associated symptoms == sciatica and Lower back pain.instance. Associated symptoms.instance.duration > 4 weeks`). The name attribute is at level A, while the specification attribute is at level B. When an author authors the guideline he can first specify the level A attributes, and can later specify the level B attributes. All GLIF classes have at least one level A attribute that lets the author define a free text description.

3. Creating a guideline

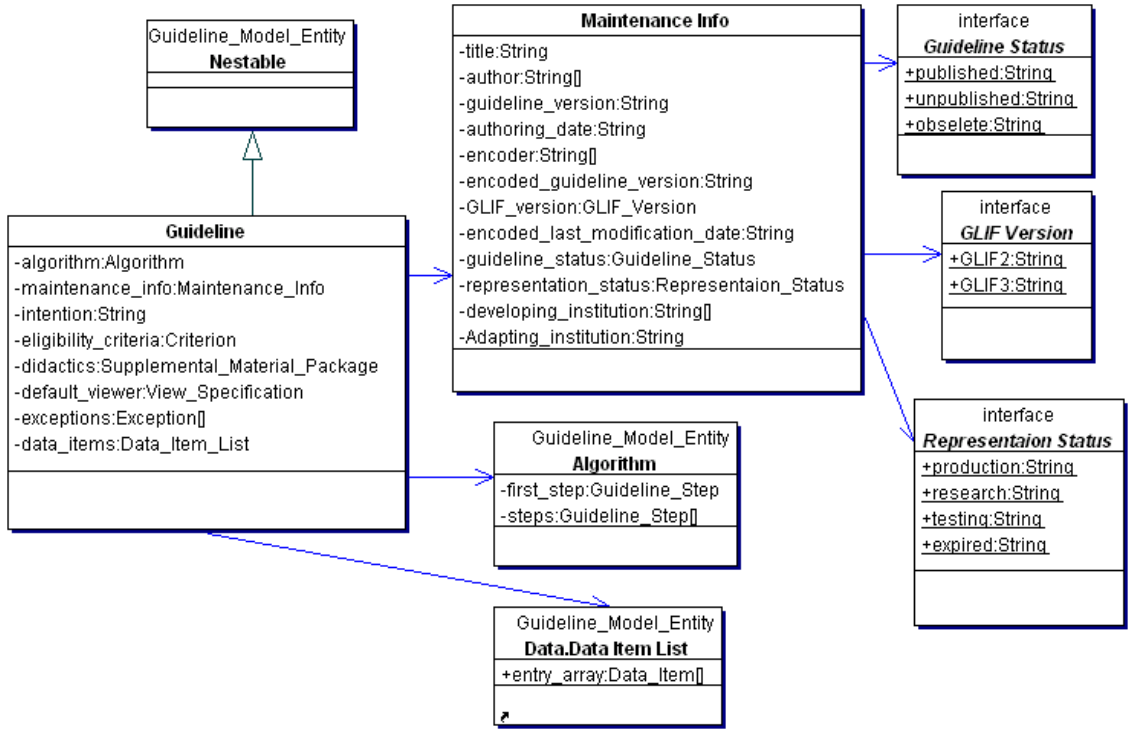


Figure 1. The Guideline Package

Guideline
[A]name: String
[A]algorithm: Algorithm
[Ao]maintenance_info: Maintenance_info
[Ao]intention: String
[A]eligibility_criteria: Criterion
[A]didactics: Supplemental_Material_Package
[A]default_viewer: View_Specificaiton
[A]exceptions: Exception[]
[B]data_items: Data_Item_List

Figure 2. The Guideline class. The abstraction levels of an attribute are shown to its left. An [o] indicates optional attributes.

The guideline class is used to model clinical guidelines and subguidelines. A guideline has an algorithm that is a flowchart of temporally ordered steps. These steps represent clinical decision, action, and patient state steps. Concurrency is modeled using branch and synchronization steps. GLIF's guideline class also specifies maintenance information (author, status, modification date, and version), the intention of the guideline, eligibility criteria, didactics, and the set of exceptions

that upon their occurrence a new next step is entered. The guideline also defines patient data items that are accessed by it. For each guideline default viewers may be specified. Since different users may be interested in different parts of a large, complex guideline, differential display capability is supported. This capability is provided through the use of filters that collapse segments of the guideline into a default view of the guideline customized to a given user, situation, etc. An example of a GLIF3-encoded guideline that was authored using the Protégé authoring tools is shown in Figure 3.

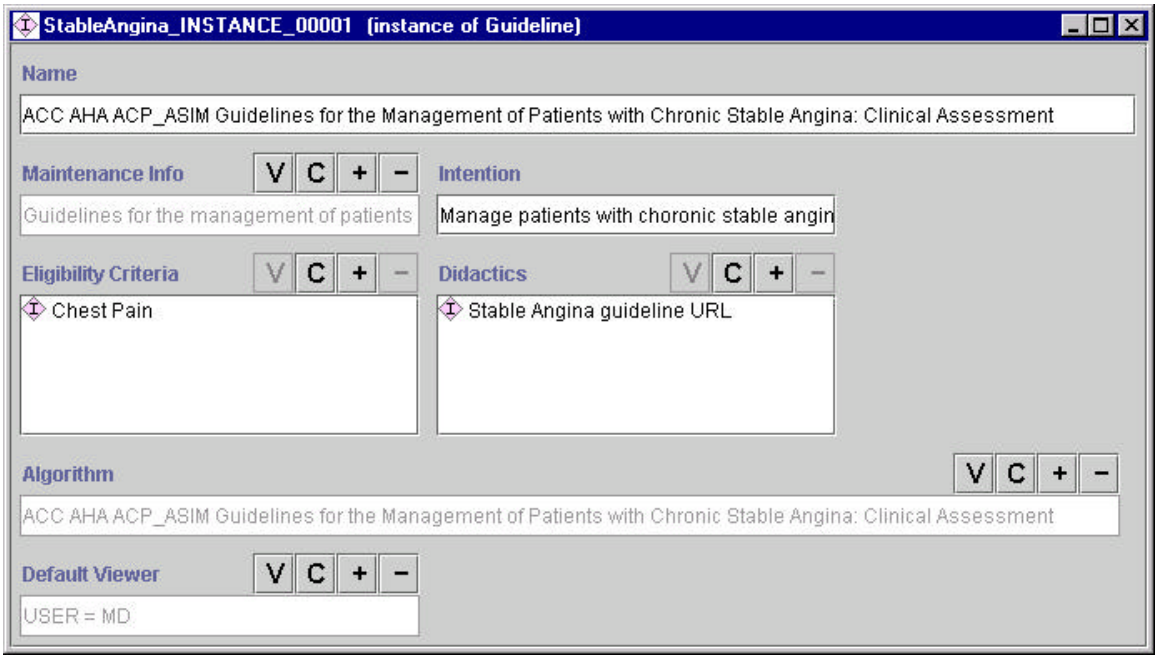


Figure 3. The Stable Angina guideline that was encoded in GLIF using the Protégé authoring tool

3.1 Header information

Maintenance information related to guidelines is represented by the Maintenance_Info class. An example is shown in Figure 5.

Maintenance_Info
[A]title:String [Ao]author: String[] [Ao]guideline_version: String [A]authoring_date:String [A]encoder: String[] [A]encoded_guideline_version: String [A]GLIF_version: float [A]encoding_last_modification_date:String [Ao]guideline_status: {published, unpublished, obsolete} [Ao]representation_status: {production, research, testing, expired} [A]developing_institution: String[] [Ao]adapting_institution: String

Figure 4. The Maintenance_Info class. The abstraction levels of an attribute are shown to its right. An [o] indicates optional attributes.

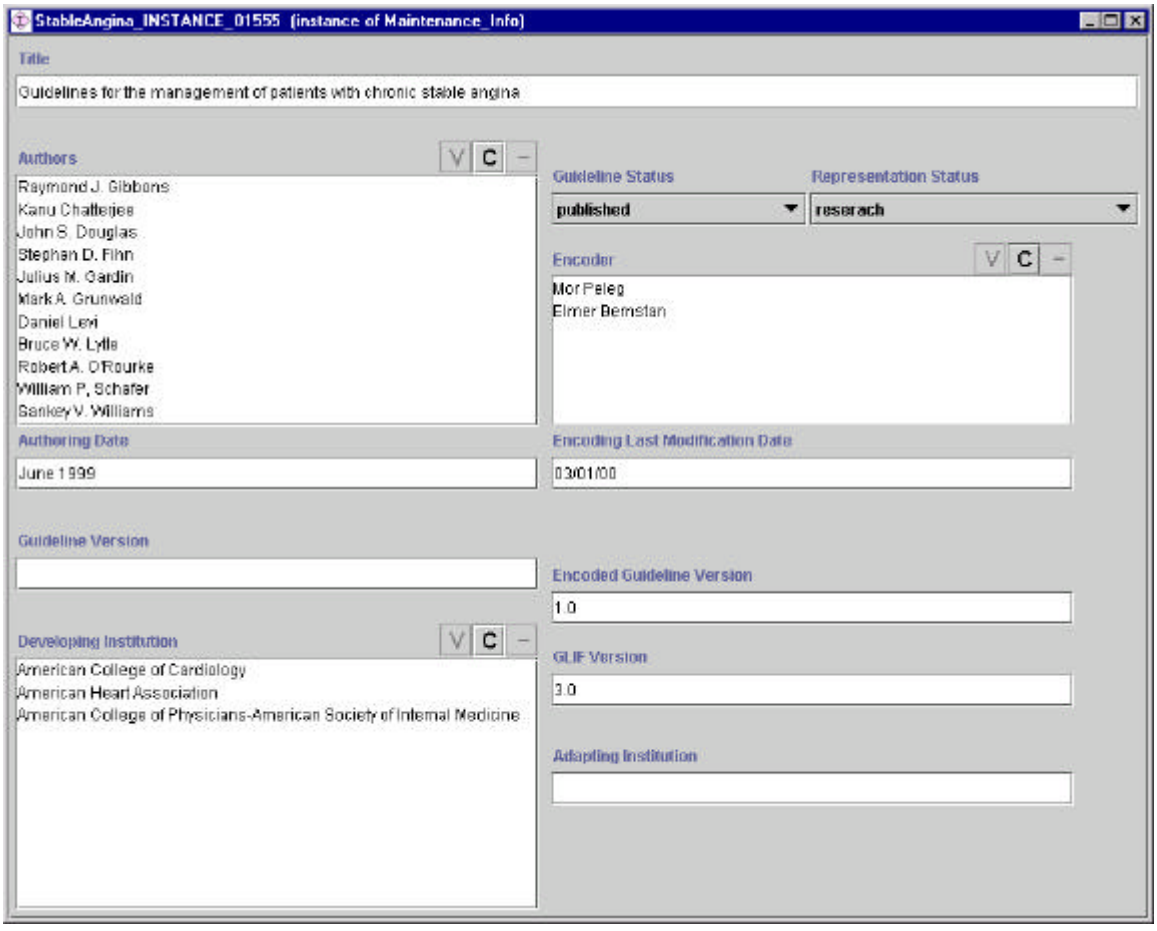


Figure 5. Maintenance information of the stable angina guideline shown in Figure 3

3.2 *Building the flowchart*

The flowchart is an instance of the Algorithm class. It may contain up to 5 types of guideline steps, as explained above. Examples of algorithms can be seen in Figure 7 and Figure 8.

Algorithm
^{Ao}name:String ^{A}first_step: Guideline_Step ^{A}steps:Guideline_Step[]

Figure 6. The Algorithm class. The abstraction levels of an attribute are shown to its right. An ^{o} indicates optional attributes.

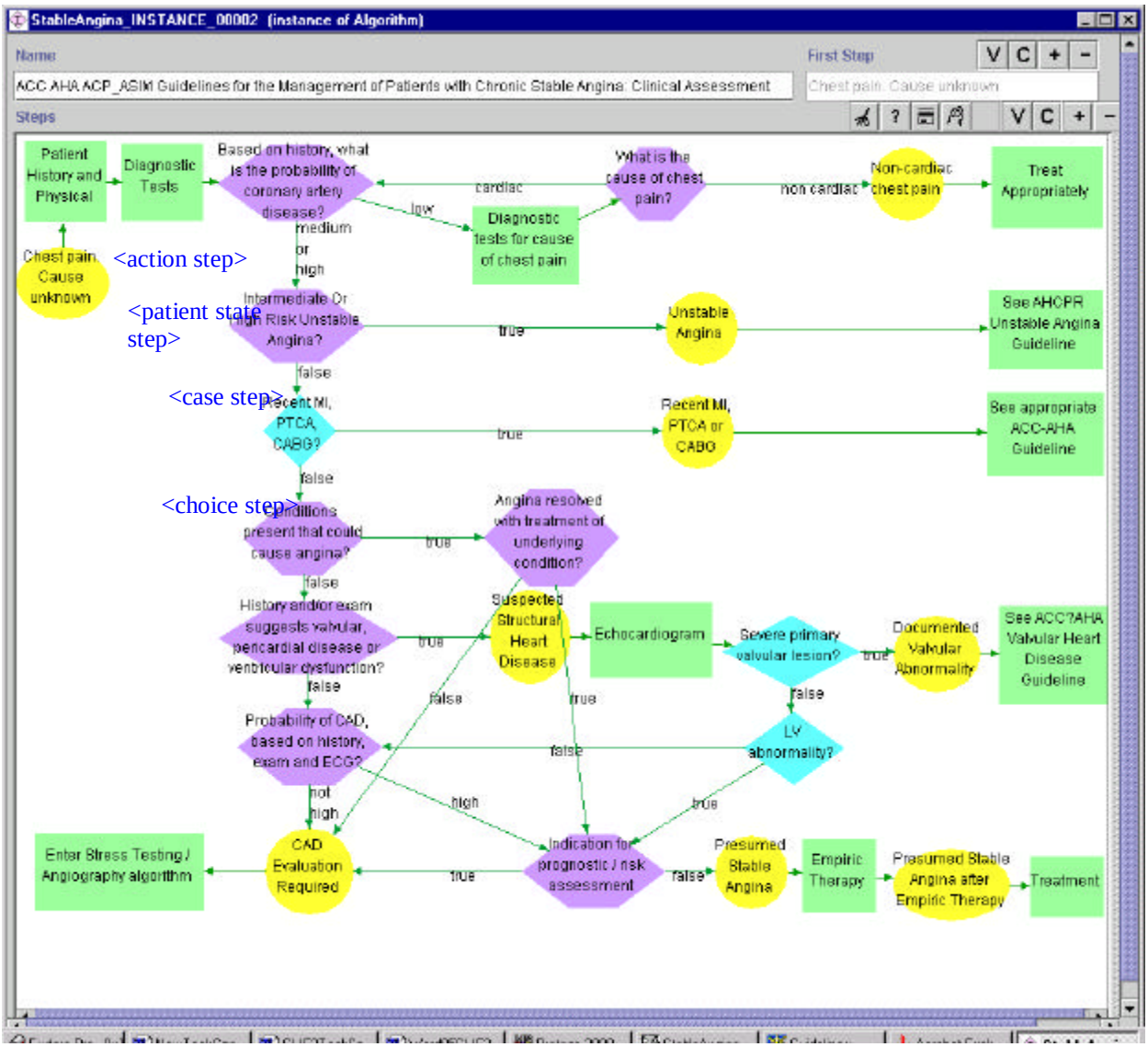


Figure 7. The top-level algorithm for the stable angina guideline shown in Figure 3

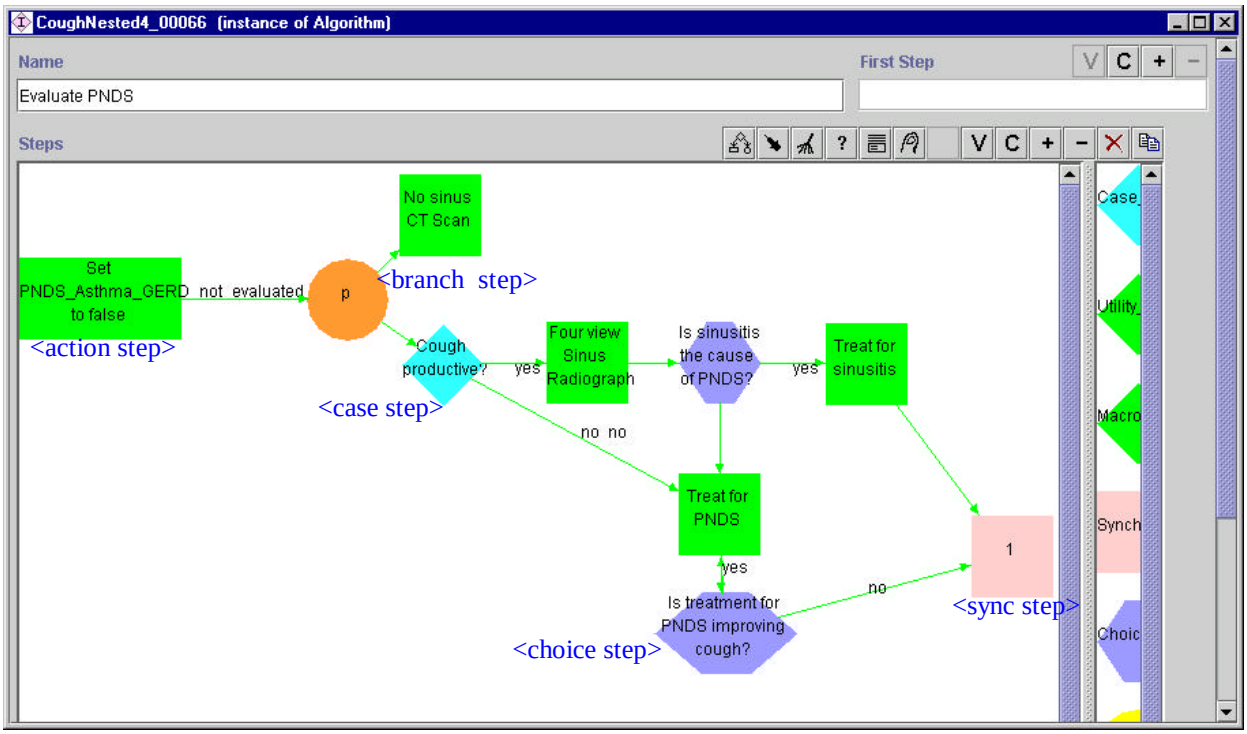


Figure 8. The algorithm for evaluating PNDS as the cause of chronic cough in immunocompetent adults.

The nodes in a guideline are Guideline_Steps. Each step has a name and didactics. A guideline step is an abstract class. There are different sub-classes of guideline steps. Medical actions and decisions are represented by the Action_Step and Decision_Step. Patient states are represented by the Pateint_State_Step. Concurrency is represented by the use of Branch and Synchronization steps. The hierarchy of guideline steps is shown in Figure 9.

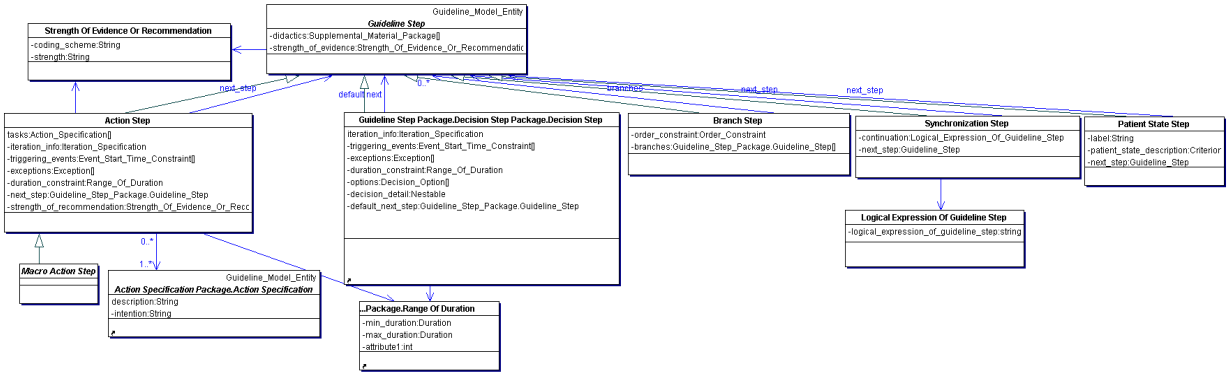


Figure 9. The guideline step package

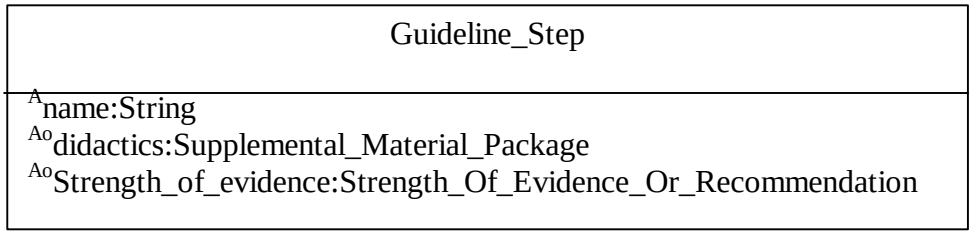


Figure 10. The Guideline_Step class

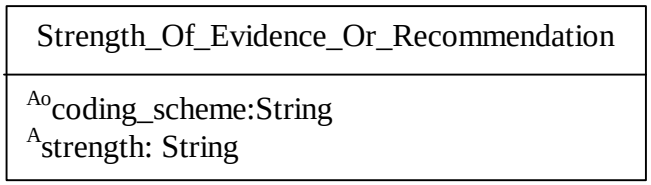


Figure 11. The Strength_Of_Evidence_Or_Recommendation class

3.3 Action Steps

Action Steps specify clinical actions that are to be performed in the patient-care process. An action step specifies a set of tasks (Action_Specifications) that need to be performed. An action step has only one next-step. The action step has attributes that specify its iteration information, duration, triggering events, and associated exceptions. Action Steps are nested by including a Subguideline_Action type of task in the step. The Subguideline_Action task has a (sub)guideline attribute that contains the nested subguideline. When a guideline step has finished its execution and the control flow is about to pass to the next step, then, if the next step has associated triggering events, then this next step is executed only after one of its triggering event occurred. An example of an action step is shown in Figure 13.

When an action step finishes its execution and the control flow is about to pass to the next step, then, if the next step has associated triggering events, then this next step is executed only after one of its triggering event occurred.

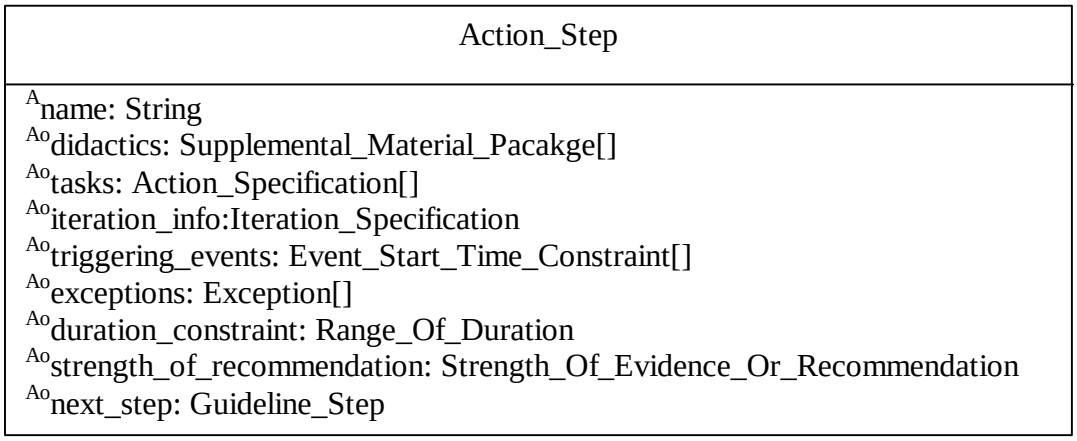


Figure 12. Action_Step class

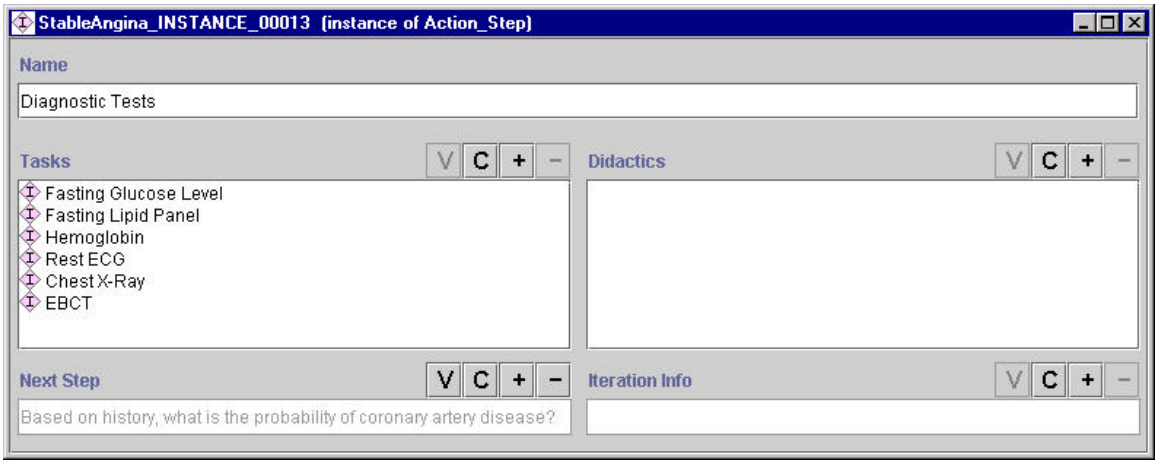


Figure 13. The details of the *Diagnostic Tests* action step shown in Figure 7

Range_Of_Duration
[A]min_duration: Duration [A]max_duration: Duration

Figure 14. The range of duration class.

3.4 Decision Steps

Decision_Step
[A]name: String [Ao]didactics: Supplemental_Material[] [Ao]iteration_info:Iteration_Specification [Ao]triggering_events: Event_Start_Time_Constraint[] [Ao]exceptions: Exception[] [Ao]duration_constraint: Time_Interval_Expression [A]options: Decision_Option[] [Ao]decision_detail: Nestable [Ao]strength_of_recommendation: Strength_Of_Evidence_Or_Recommendation [A]default_next_step: Guideline_Step

Figure 15. The Decision Step class

Decision steps direct flow from one guideline step to another. A decision step may link a guideline step to any other guideline step. A decision is linked to several decision options. If there is a match to one of the decision options, then the control flows to that decision option. If there is no match, then the control flows to the default destination guideline step. If more than one decision option is appropriate, then only one is chosen, either by the user, in the case of a choice step, or arbitrarily, in the case of a case step.

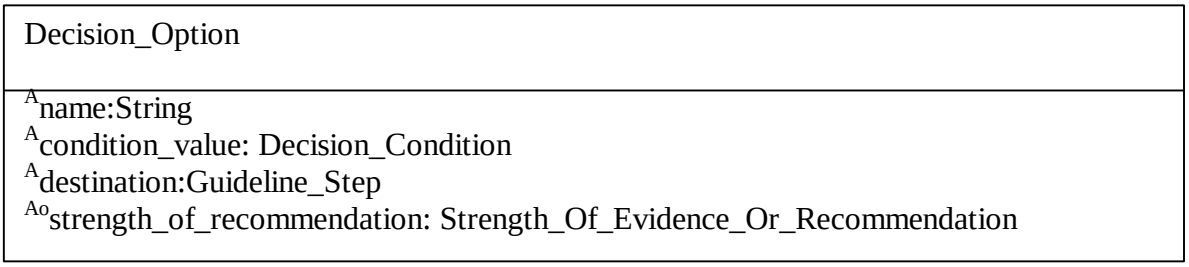


Figure 16. The Decision Option class

When a decision step finishes its execution and the control flow is about to pass to the next step, then, if the next step has associated triggering events, then this next step is executed only after one of its triggering event occurred.

Decision Steps are nested by specifying a (sub)guideline in the decision_detail attribute of the step. This subguideline is executed before the decision criterion for that step is evaluated. The subguideline would modify or create new variables and assign them values. The use of these variables in the decision criteria makes the decision nested.

The decision step is an abstract class. GLIF3 provides a flexible decision model through a hierarchy of decision step classes. This decision hierarchy distinguishes between decision steps that can be automated (*case steps*) and ones that have to be made by a physician or other health worker and cannot be automated (*choice steps*), as shown in Figure 25. The decision hierarchy can be extended in the future to model decisions that consider uncertainty or patient preferences. The hierarchy might be extended to support different decision models.

3.5 Branch Steps

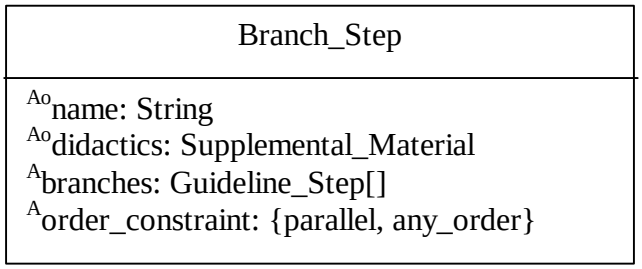


Figure 17. The Branch Step class

The branch step is used to model concurrent guideline steps. Branch steps direct flow to multiple guideline steps. All of these guideline steps must occur in parallel. A branch step may link a guideline step to any other guideline step. An example of a branch step is shown in Figure 18.

The order constraint “one_of” that was allowed in GLIF2 was removed so that the branch step would not semantically overlap the case step.

The selection method attribute that previously characterized the branch step was removed so that the branch step would not semantically overlap the synchronization step.

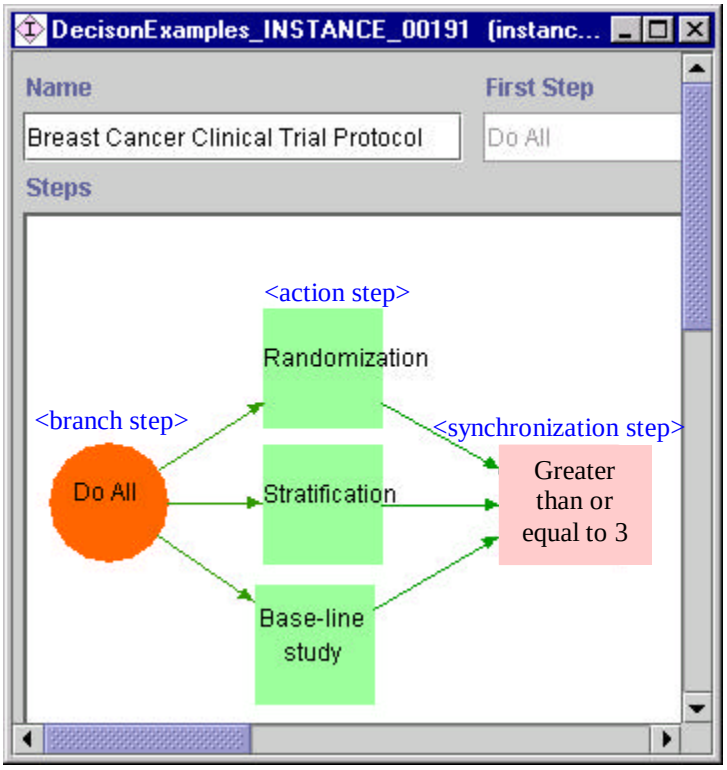


Figure 18. An example of branch and synchronization steps

3.6 Synchronization Steps

Synchronization_Step
[Ao]name: String [Ao]didactics: Supplemental_Material [Ao]next_Step: Guideline_Step [A]continuation: Logical_Expression_Of_Guideline_Step

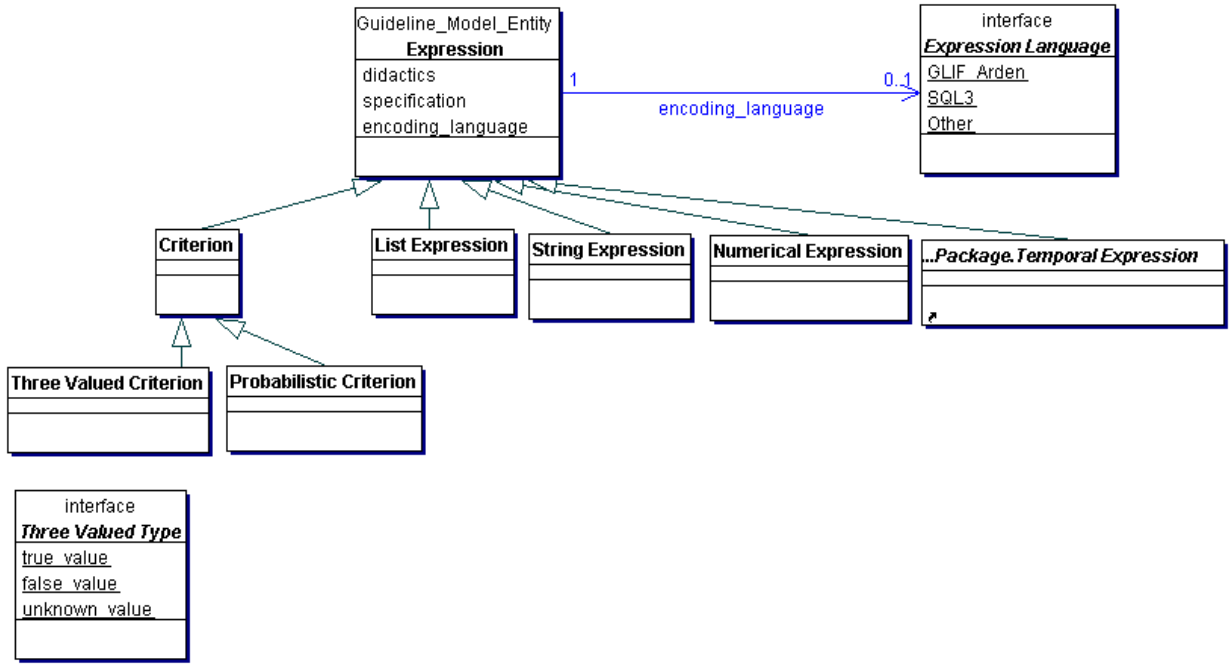
Figure 19. The Synchronization Step class

Synchronization steps are used in conjunction with branch steps. When multiple guideline steps follow a branch step, the flow of control must eventually converge in a single step. Each branch may lead to a series of steps, resulting in a set of branching paths. The step at which the paths converge is the synchronization step. When the flow of control reaches the synchronization step a continuation attribute specifies whether all, some or one of the preceding steps must have been completed before control can move to the next step. The continuation is expressed as a logical expression of a guideline steps (e.g., ((Step_A or Step_B) and step_C); greater than or equal to 4). The syntax follows.

Logical_expression_of_guideline_steps: **Guideline_Step** |
(Logical_expression_of_guideline_steps**)** | **not** Logical_expression_of_guideline_steps |
Logical_expression_of_guideline_steps **and** Logical_expression_of_guideline_steps |
Logical_expression_of_guideline_steps **or** Logical_expression_of_guideline_steps | **greater than or equal to** Integer

3.7 First look at expressions

Package *Expression_Package*



The Expression class is a parent Class for all expressions: criterion, temporal expression, list expression, numerical expression and string expression. Different expression languages can be used. Currently, BNF grammar is given for a modification of the Arden Syntax logic grammar, called GLIF_ARDEN. While this gives guideline users the flexibility to specify criteria in a language of their choosing, it is possible that the resulting guidelines will not be sharable because constructs supported by one language (e.g., temporal constructs) cannot be expressed in another language.

Criterion: There are two types of criteria: three-valued, and probabilistic. Three-valued criteria evaluate to true, false, or unknown. Probabilistic criteria are used to add conditional probabilities or Bayesian network information. With probabilistic criteria, the dependence probabilities among all variables in a specification have to be known in order for the criterion to be evaluated. The problem is that the number of conditional probabilities that need to be specified increases exponentially as the number of variables used in a specification increase, and the dependence relationships among some variables may not be known. Users of probabilistic criteria need to provide the conditional probabilities needed for evaluating these criteria, and this means that use will probably be limited to advanced users of GLIF.

List_Expression: an expression involving comma-separated strings, numbers, variable names, or other literals

Probabilistic_Criterion: To add conditional probabilities or Bayesian network info

String_Expression: an expression involving strings

Considerations:
There is no class Two-Valued_Criterion because expressions involving boolean/two-valued criteria can be handled using three-valued criteria.
The attribute, Expression.encoding_language was added to provide a means of using different expression languages (besides GLIF_ARDEN) in GLIF. While this gives guideline users the flexibility to specify criteria in a language of their choosing, it is possible that the resulting guidelines will not be sharable because constructs supported by one language (e.g., temporal constructs) cannot be expressed in another language. While flexibility is an admirable goal, more consideration should be given to the long-term consequences of allowing different expression languages if the goal of making guidelines sharable is to be achieved.
With probabilistic criteria, the dependence probabilities among all variables in a specification have to be known in order for the criterion to be evaluated. The problem is that the number of conditional probabilities that need to be specified increases exponentially as the number of variables used in a specification increase, and the dependence relationships among some variables may not be known. Users of probabilistic criteria need to provide the conditional probabilities needed for evaluating these criteria, and this means that use will probably be limited to **advanced** users of GLIF.
Evaluation of the structured string in an expression specification is performed within the parser for the grammar corresponding to Expression.encoding_language. Evaluation occurs after the entire string has been parsed. A table/database that stores variable names and values is needed in order to perform the evaluations. For probabilistic criteria, this table/database would have to include the conditional probabilities needed for evaluating the expression.
Partial specification of GLIF_ARDEN grammar in Backus Naur Form (BNF): (The grammar has been expanded to include almost all of the operators that Arden Syntax allows – 01/06/2000)
A modification of the Arden syntax is used, and not Arden syntax itself, since we wanted to be able to express certain operators, which are not supported by Arden syntax, but are needed for specifying criteria. These operators are "is a", "overlaps", "xor", "from now", "is unknown" and "at least k of ...". GLIF made minor changes to the syntax of some operators to allow a more natural way of expressing them; the operator "occur/occurs/occurred at" has the same meaning as "occur/occurs/occurred equals" and the operators "in", "not in" has the same meaning as Arden's "is in", "is not in". All of these operators, including the original Arden syntax format, can be used. All of Arden's logic grammar is supported by GLIF.

Following is the expression syntax.

BNF for ExprLang.jj

NON-TERMINALS

CompilationUnit ::= (StatementOrExpression (<EOL>)+)* <EOF>
StatementOrExpression ::= Assignment
| FunctionStatement
| IfStatement
| ConcludeStatement

| Expression
Statement ::= Assignment
| FunctionStatement
| IfStatement
| ConcludeStatement
Assignment ::= Id <ASSIGN> Expression <SEMICOLON>
FunctionStatement ::= Function <SEMICOLON>
IfStatement ::= <IF> Expression <THEN> Statement (<ELSE> Statement)? <ENDIF> <SEMICOLON>
ConcludeStatement ::= <CONCLUDE> Expression <SEMICOLON>
Expression ::= ConditionalExpression
ConditionalExpression ::= WhereExpression
WhereExpression ::= OrExpression (<WHERE> OrExpression (<ORDER_BY> OrExpression)?)?
OrExpression ::= ConditionalAndExpression (<OR> ConditionalAndExpression | <XOR>
ConditionalAndExpression)*
ConditionalAndExpression ::= ComparisonExpression (<AND> ComparisonExpression)*
ComparisonExpression ::= ConcatExpression (<EQUAL> ConcatExpression | <NOTEQUAL>
ConcatExpression | <LT> ConcatExpression | <LEQUAL> ConcatExpression | <GT> ConcatExpression |
<GEQUAL> ConcatExpression | <IS_WITHIN> ConcatExpression (<TO> ConcatExpression |
<PRECEDING> ConcatExpression | <FOLLOWING> ConcatExpression | <SURROUNDING>
ConcatExpression | <PAST> ConcatExpression | <SAME_DAY_AS> ConcatExpression) |
<IS_BEFORE> ConcatExpression | <IS_AFTER> ConcatExpression | <IS_IN> ConcatExpression |
<OVERLAPS> ConcatExpression)*
ConcatExpression ::= AddExpression (<CONCAT> AddExpression)*
AddExpression ::= MultiplyExpression (<MINUS> MultiplyExpression | <PLUS> MultiplyExpression)*
MultiplyExpression ::= PowerExpression (<TIMES> PowerExpression | <DIVIDE> PowerExpression)*
PowerExpression ::= B4AfterExpression (<POWER> PowerExpression)*
B4AfterExpression ::= UnaryExpression (<BEFORE> UnaryExpression | <AFTER> UnaryExpression)*
UnaryExpression ::= <MINUS> UnaryExpression
| <PLUS> UnaryExpression
| <NOT> UnaryExpression
| <FIRST> UnaryExpression
| <LAST> UnaryExpression
| <EARLIEST> UnaryExpression
| <LATEST> UnaryExpression
| <NEAREST> UnaryExpression
| <ANY_OF> UnaryExpression
| <ALL_OF> UnaryExpression
| <TIME_OF> UnaryExpression
| <IS_NULL> UnaryExpression
| <IS_BOOLEAN> UnaryExpression
| <IS_UNKNOWN> UnaryExpression
| <IS_NUMBER> UnaryExpression
| <IS_TIME> UnaryExpression
| <IS_DURATION> UnaryExpression
| <IS_STRING> UnaryExpression
| <IS_LIST> UnaryExpression
| Duration <AGO>
| Duration <FROM_NOW>
| PrimaryExpression
PrimaryExpression ::= Literal
| Interval
| List
| Function
| Id ("." Id)*
| "(" Expression ")"
| <TRUE>

| <FALSE>
| <UNKNOWN>
Id ::= <ID>
Duration ::= <NUMBER> (<YEAR> | <MONTH> | <WEEK> | <DAY> | <HOUR> | <MINUTE> | <SECOND>)
Function ::= Id "(" (ArgumentList)? ")"
ArgumentList ::= Expression ("," Expression)*
Interval ::= <INTERVAL> ("(" | "[") (((<NUMBER> | <INFINITY>) "," (<NUMBER> | <INFINITY>) (")" | "]")) | ((<DATE> "," <DATE>) (")" | "]")) | (Duration "," Duration (")" | "]")))
List ::= <LIST> ("(" ")" | "(" Literal (<COMMA> Literal)* ")")
Literal ::= (<STRING> Id)
| (<STRING>)
| Duration
| (<NUMBER> Id)
| (<NUMBER>)
| (<DATE>)

Tokens/Terminals

```
TOKEN : /* RESERVED WORDS */
{
  < BOOLEAN: "boolean" >
| < DURATION: "duration" >
| < FALSE: "false" >
| < NULL: "null" >
| < RES_NUMBER: "number" >
| < RES_STRING: "string" >
| < TIME: "time" >
| < TRUE: "true" >
| < UNKNOWN: "unknown" >
| < INFINITY: "infinity" >
}

TOKEN : /* OPERATORS */
{
  < COMMA: "," >
| < WHERE: "where" >
| < OR: ("|" | "or") >
| < XOR: ("*|" | "xor") >
| < AND: ("&" | "and") >
| < NOT: ("!" | "not") >
| < EQUAL: ("==" | "=") > // here to next comment -- same precedence
| < NOTEQUAL: ("!=" | "<>" ) >
| < LT: "<" >
| < LEQUAL: "<=" >
| < GT: ">" >
| < GEQUAL: ">=" >
| < IS_WITHIN: "is within" >
| < TO: "to" >
| < PRECEDING: "preceding" >
| < FOLLOWING: "following" >
| < SURROUNDING: "surrounding" >
| < PAST: "past" >
```

```
| < SAME_DAY_AS: "same day as" >
| < BEFORE: "before" >
| < AFTER: "after" >
| < IS_BEFORE: "is " >
| < IS_AFTER: "is " >
| < IS_IN: ("is ")? "in" > // end same precedence
| < IS_NULL: "is " < NULL > >
| < IS_BOOLEAN: "is " >
| < IS_UNKNOWN: "is " < UNKNOWN > >
| < IS_NUMBER: "is " < NUMBER > >
| < IS_TIME: "is " < TIME > >
| < IS_DURATION: "is " < DURATION > >
| < IS_STRING: "is " < STRING > >
| < IS_LIST: "is " < LIST > > | < CONCAT: "||" >
| < PLUS: "+" >
| < MINUS: "-" >
| < TIMES: "*" >
| < DIVIDE: "/" >
| < POWER: ("**" | "^") >
| < AGO: "ago" >
| < FROM_NOW: "from now" >
| < YEAR: "years" | "year" >
| < MONTH: "months" | "month" >
| < WEEK: "weeks" | "week" >
| < DAY: "days" | "day" >
| < HOUR: "hours" | "hour" >
| < MINUTE: "minutes" | "minute" >
| < SECOND: "seconds" | "second" >
| < ANY_OF: "any of" >
| < ALL_OF: "all of" >
| < LAST: "last" >
| < FIRST: "first" >
| < LATEST: "latest" >
| < EARLIEST: "earliest" >
| < NEAREST: "nearest" >
| < FROM: "from" >
| < INTERVAL: "interval" >
| < LIST: "list" >
| < TIME_OF: < TIME > (" ") "of" >
| < ORDER_BY: "order by" >
| < AT_LEAST: "at least" >
| < OF: "of" >
| < EVERY: "every" >
| < FOR: "for" >
| < OVERLAPS: "overlaps" >
}

TOKEN : /* STATEMENTS */
{
  < IF: "if" >
| < THEN: "then" >
| < ELSE: "else" >
| < ENDIF: "endif" >
| < CONCLUDE: "conclude" >
| < ASSIGN: ":=" >
}
```

```
TOKEN : /* IDENTIFIERS -- VARIABLES OR FUNCTION NAMES */
{
  < ID: ["a"-"z","A"-"Z"] (["a"-"z","A"-"Z","0"-"9"])* >
}

TOKEN : /* LITERALS */
{
  < STRING: "\""(~["\"", "\n", "\r"])*"\"" >
| < NUMBER:
            (["0"-"9"])* "." (["0"-"9"])+ (<EXPONENT>)? (["l","L","f","F"])?
           |(["0"-"9"])+ <EXPONENT> (["l","L","f","F"])?
           |(["0"-"9"])+ >
| < EXPONENT: ["e","E"] (["+","-"])? (["0"-"9"])+ >
| < DATE: ["0"-"9"]["0"-"9"]["0"-"9"]["0"-"9"]"-"["0"-"9"]["0"-"9"]"-"["0"-"9"]["0"-"9"]
             ("T"["0"-"9"]["0"-"9"](":"["0"-"9"]["0"-"9"](":"["0"-"9"]["0"-"9"]("."(["0"-"9"])+ )?("Z" | <DIFF>)?)?)?)? >
| < DIFF: "+"["0"-"9"]["0"-"9"]":"["0"-"9"]["0"-"9"]
            |"-"["0"-"9"]["0"-"9"]":"["0"-"9"]["0"-"9"] >
 }
```

Operators that exist in GLIF_Arden but not in Arden Syntax:

Unary: from now, is unknown

Binary: in, not in (same as Arden's is in, is not in), overlaps, xor, |* , is a, is-a, occur/occurs/occurred at, at least...of

from now

In Arden, you can refer to the time of an event/occurrence in the past by saying "two days ago". But there is no similar syntax for referring to the time of a future event. "from now" was added so that we can say "[do x] two days from now".

is unknown

Testing if something is null is not the same thing as testing if it is unknown. If I have a data item that has not been initialized or assigned a value, it evaluates to null (e.g. if someone attempts to use the value of a data item without first getting it from the patient record or a physician). This is something that can be tested by Arden. If we want to note that I don't know whether the result of a logical expression is true or false then we can assign the value "unknown" to a variable representing the results of the expression. This variable has a value ("unknown") and is therefore not null.

in, not in

“in” and “not in” mean the same as Arden's “is in”, “is not in”, respectively.

overlaps

“overlaps” is used for comparing intervals (time or other intervals). So for example, [3:5] overlaps [2:4] would evaluate to true but [3:5) overlaps [5:9] would evaluate to false.

xor, |*

A xor B means ((A or B) and not (A and B))

|* is a synonym for exclusive or (xor).

is a, is-a
"is-a" is a relationship that we probably should not model as an operator. The reason for that is that we cannot evaluate it without support from the third layer of the data model/domain ontology. In fact, different third layers might choose to interpret "is-a" differently, depending on their view of the knowledge.

at least...of
The "at least...of" operator allows us to express very basic existential/universal quantification (i.e., "at least 1 of ... " is equivalent to "there exists ..." and "not (at least 1 of (not...))" is equivalent to "for all ..."). It also allows expressing "k of n" criteria.

occur/occurs/occurred at
"occur/occurs/occurred **at**" is synonymous with Arden's "occur/occurs/occurred **equal**" operator and would be evaluated exactly the same way. It just seemed like it would be clearer to use it in some situations.

Following are examples of criteria.

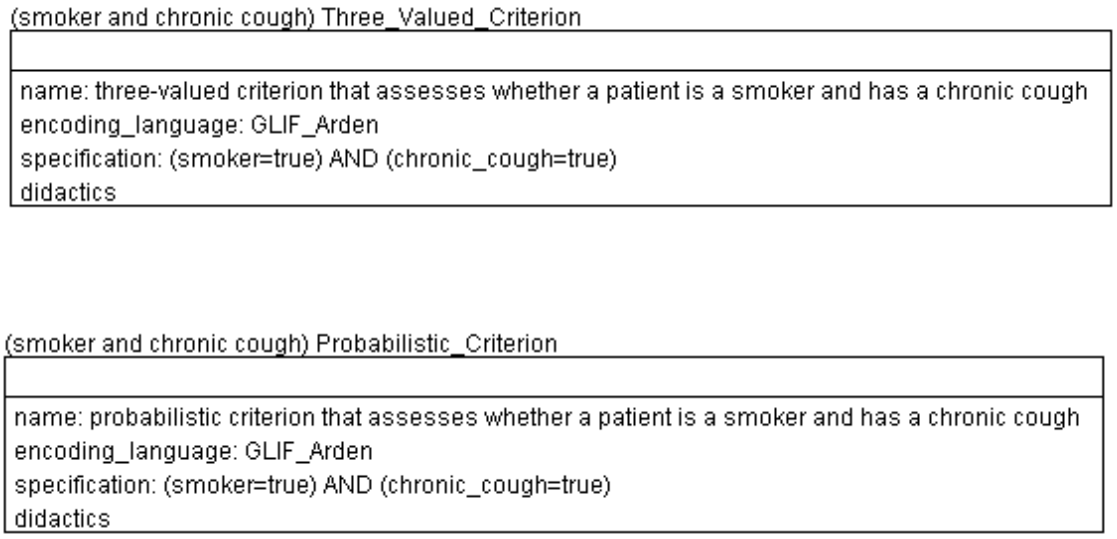


Figure 20. Examples of criteria.

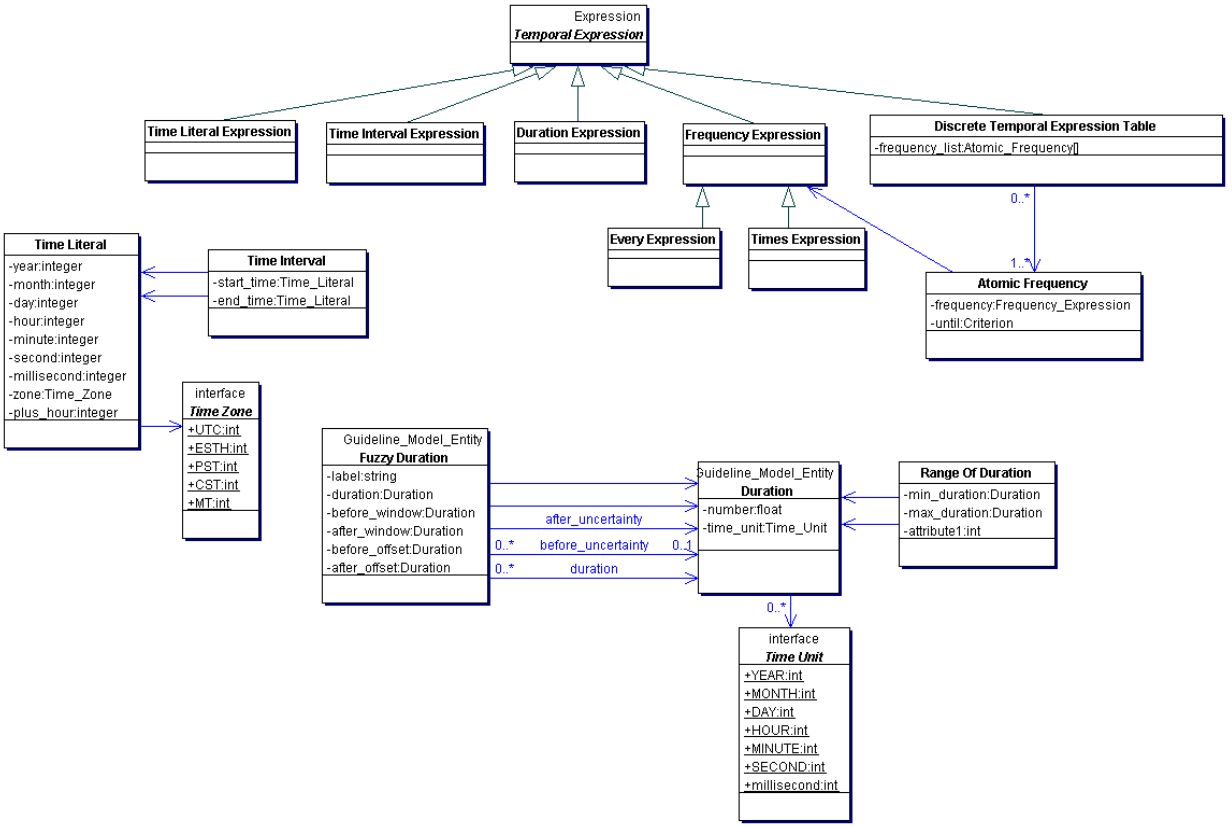
The three-valued_criterion specification: (smoker = true) AND (chronic_cough = true) evaluates to true if a patient is a smoker and has a chronic cough. It evaluates to false if the patient is not a smoker, or does not have a chronic cough, or both. It evaluates to unknown if it is unclear that the patient is a smoker or has a chronic cough, or both.
As a specific example, if we don't know whether or not the patient is a smoker and we know that the patient has a chronic cough, the criterion evaluates to unknown.

To evaluate the probabilistic_criterion specification: (smoker = true) AND (chronic_cough = true), we need to know the probability that a patient is a smoker, P(smoker=true), the probability that a patient has a chronic cough, P(chronic_cough=true), and the probability that a patient has a chronic cough given that the patient is a smoker, P(chronic_cough=true| smoker=true). This last probability expresses the dependence relationship between a patient being a smoker and the patient having a chronic cough. Given these three probabilities, the specification evaluates to a real number in the interval [0,1]: P((smoker = true) AND (chronic_cough = true)) = P(smoker=true)* P(chronic_cough=true| smoker=true)
If there is a 70% probability that the patient is a smoker and we know that there is a 95% probability of a patient having a chronic cough given that s/he is a smoker, then, P((smoker = true) AND (chronic_cough = true)) = .7 * .95 = 0.665

Package *Temporal_Expression_Package*

Different types of temporal expressions are possible. These are: time literal expression, time interval expression, duration expression, frequency expression, and discrete temporal expression table.

Class Diagram



Class Detail

TemporalExpression:
TimeLiteral | Duration | TimeInterval | Frequency | FrequencySet | Duration UnaryOperator | UnaryOperator Number of Identifier | UnaryOperator Identifier | TemporalExpression BinaryOperator TemporalExpression | Identifier BinaryOperator TemporalExpression | TemporalExpression BinaryOperator Identifier | TimeInterval lasting Duration to Duration

An *atomic frequency* specifies the frequency at which something should occur and the duration after which all iterations should end
A *discrete temporal expression table* specifies a table who consist of rows of pairs of frequencies and durations (the pairs are Atomic frequencies). For example, see patient every 5 weeks for 5 months, then, every 2 weeks for 1 month, and then every week for 1 month. The order of rows is important. The rows are to be executed from the top of the table to its bottom.

An *every expression* specifies that something should occur every fuzzy duration with an allowed before and after offset (e.g., do something every 8 hours).

A *times expression* specifies that something should occur a specified number of times within a specified interval (the per attribute) (e.g., 3 times a day).) This class should not be used during execution. It should be mapped, or refined to a tight temporal expression.

A *fuzzy duration* is a duration that has an asociated before and after uncertainty. Any time point within (duration-berfore_uncertainty, duration+after_uncertainty) is considered to be within the duration of the fuzzy duration. If the occurrence happened in the interval (every.duration - every.before_window - allowed_before_offset, every.duration - every.before_window) then the iteration point should be reset to the time of the event occurrence. An example that uses fuzzy duration is: the patient should come back to the clinic after 2 weeks, with a before and after uncertainties of 2 days).

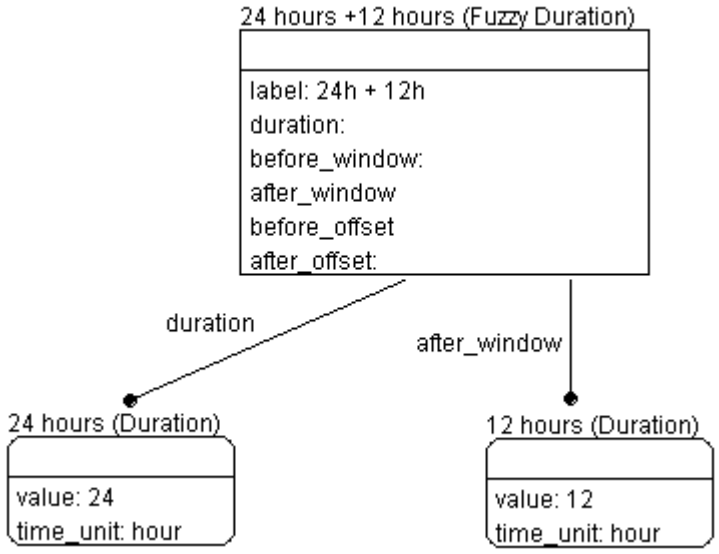


Figure 21(a) An example of a fuzzy duration. A pill has to be taken every 24 hours. If the person forgot to take the pill, she may take it within 12 hours, otherwise, the dose is skipped.
(b) An example of offsets. A pill should be taken every 4-5 hours. The next pill should be taken 4-5 hours after the previous pill was taken.

A *time-literal_expression* involves a specific instance in time (expressed as yyyy-mm-ddThh:mm:ss(Z|+/-hh) based on Arden syntax notation.

A *time zone* can be CST, EST, MT, PST, or UTC.

Temporal expressions which are a function of the number of the iteration, or the time that passed from the first iteration, or the time that passed from some milestone, are represented only as discrete set temporal criterion, and not as a function such as 2i or 2(10-t).

In order for GLIF to include temporal expressions, there has to be a notion of time-stamped events or episodes and time-stamped clinical data/patient-related concepts and this currently does not exist. This could be captured by having variables that have associated start and end times to represent different events/episodes, etc. An instantaneous event would have equal start and end times and an episode with a duration would have an end time greater than the start time. Currently there is no means of analyzing temporal trends in GLIF, and it is probably something we want to support in future versions of GLIF. To facilitate analysis of temporal trends in the future, we could introduce (now) the Arden method of creating a list that describes an event or episode's history with associated start and end time-stamps (e.g., a list of blood pressure readings for a patient at different instances in time where start_time=end_time for each reading).

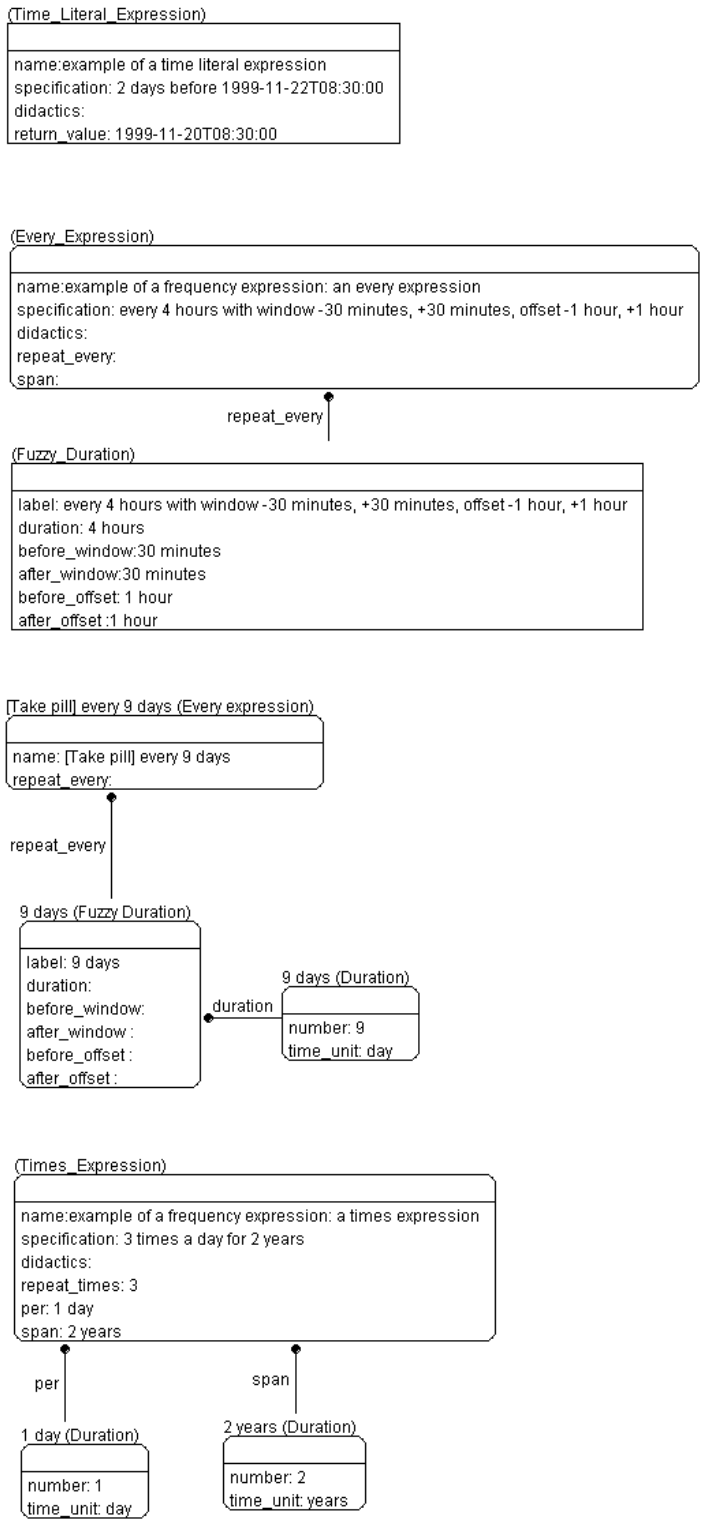


Figure 22. Examples of temporal expressions. (a) The time-literal expression specification: 2 days BEFORE 1999-11-22T08:30:00 produces the result: 1999-11-20T08:30:00 which is of type time-literal. (b, c) The every_expression specification: every 4 hours with window -30 minutes, +30 minutes, offset -1 hour, +1 hour (d) The times_expression specification: 3 times a day for 2 years

3.8 Documenting the guideline

Package *Supplemental_Material_Package*

Class Diagram

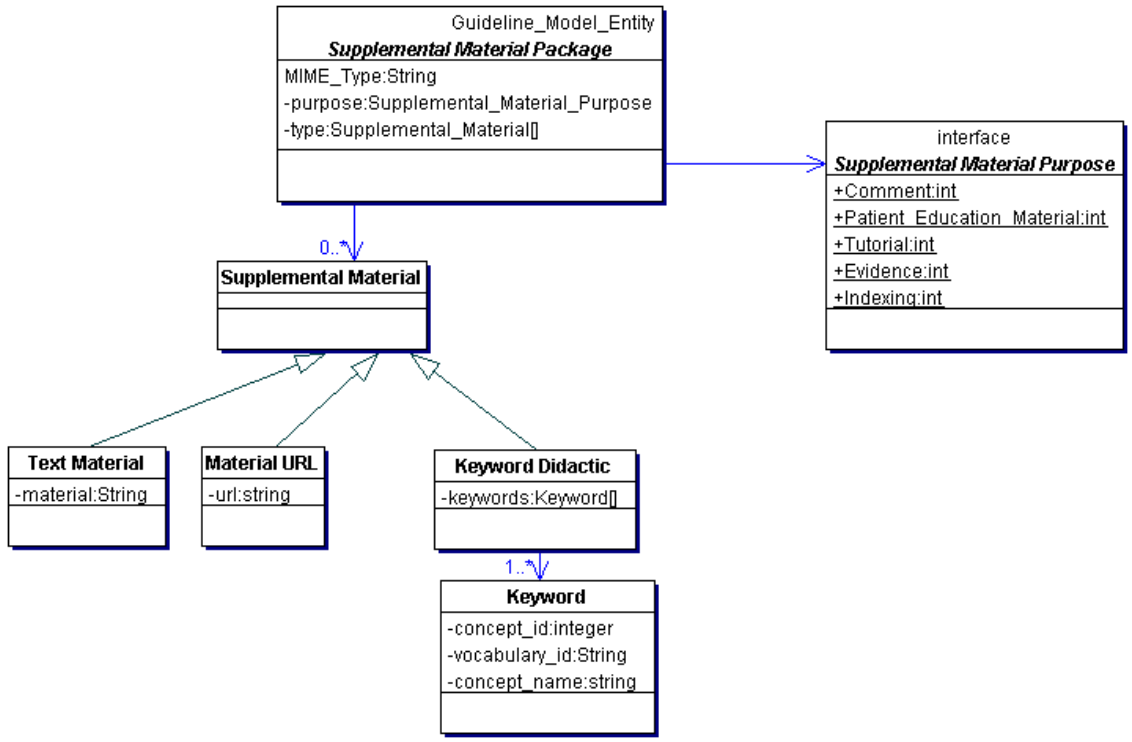


Figure 23. The supplemental material package class diagram.

Supplemental material include text material, URL material, and keyword didactics.
The Supplemental_Material_Package class provides background or supporting information.

Multipurpose Internet Mail Extensions (MIME) extends the format of Internet mail to allow non-US-ASCII textual messages, non-textual messages, multipart message bodies, and non-US-ASCII information in message headers
Level: A Examples of MIME_Types are: text/plain, text/html, image/gif, image/jpeg, mov/qt, charset=us-ascii

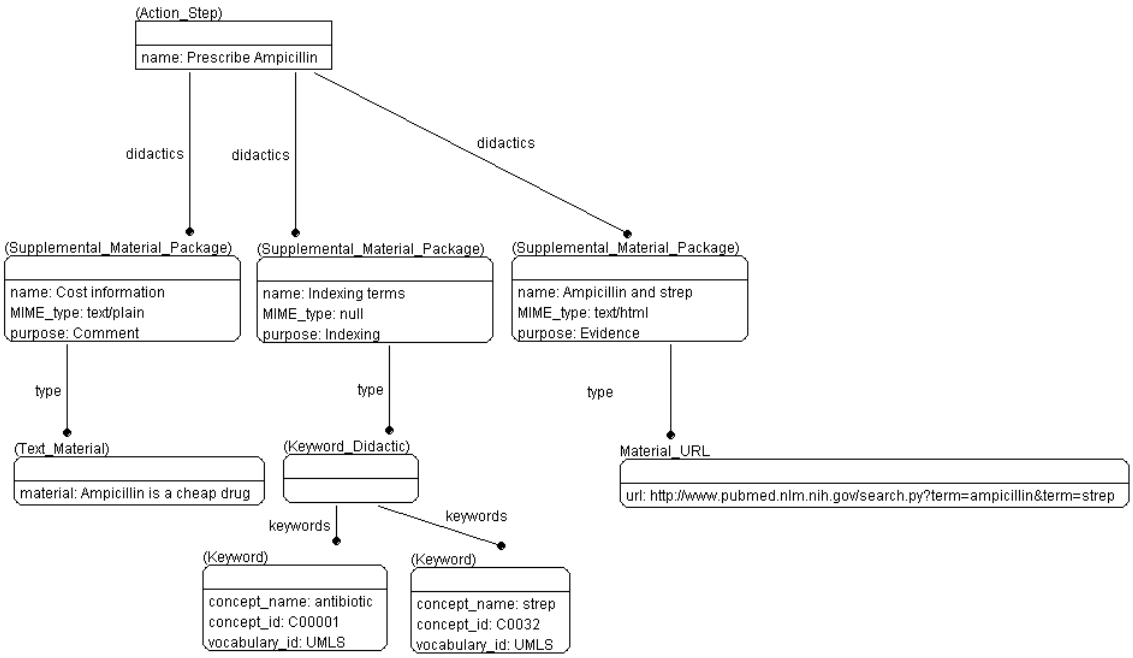


Figure 24. An example of supplemental material packages.

4. Specifying decisions

4.1 Different types of decision steps

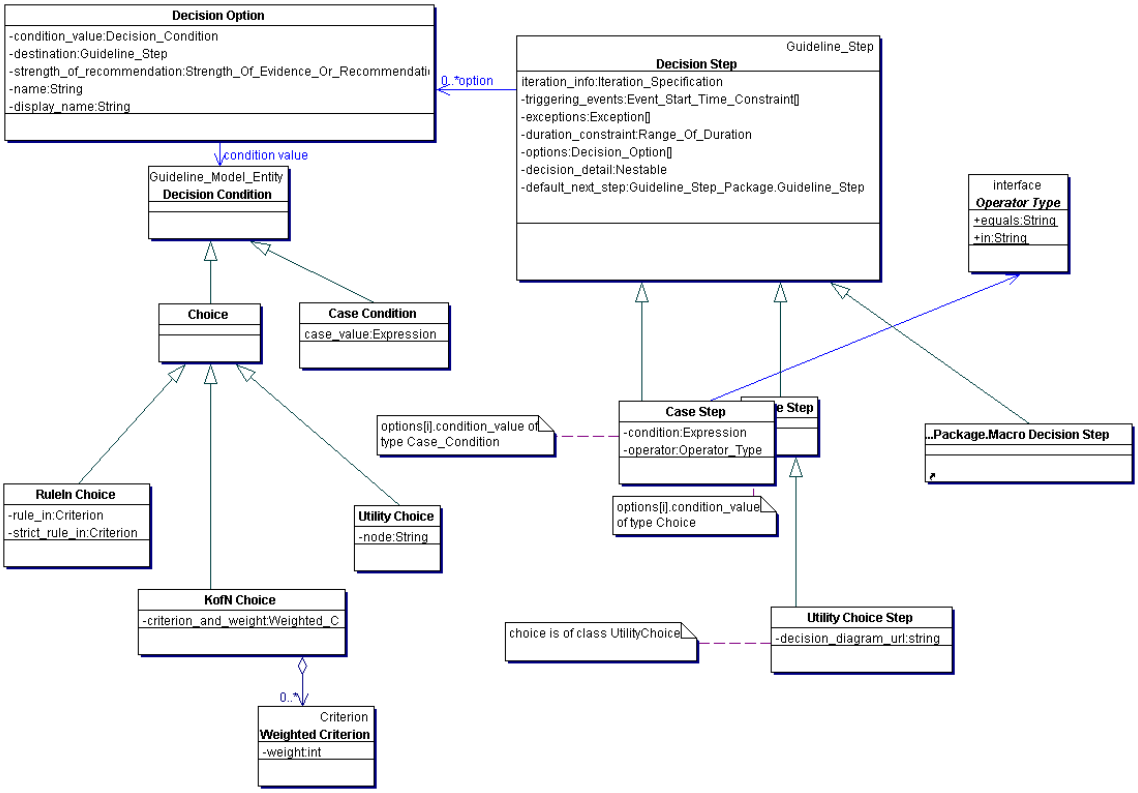


Figure 25. The decision step hierarchy

4.1.1 Case Steps

GLIF2' s conditional step used a Boolean model. This made it cumbersome and error-prone to represent criteria that do not have a true-or-false outcome (e.g., what is the patient's age category: neonate, infant, toddler, child, adolescence, adult, elderly). Therefore, the case step replaced the conditional step by allowing a conditional choice to be made among several alternative guideline steps. Case step options should be mutually exclusive. If they are not, then exactly one will be chosen. Which option is chosen is not defined. The structure of a case step is shown in .

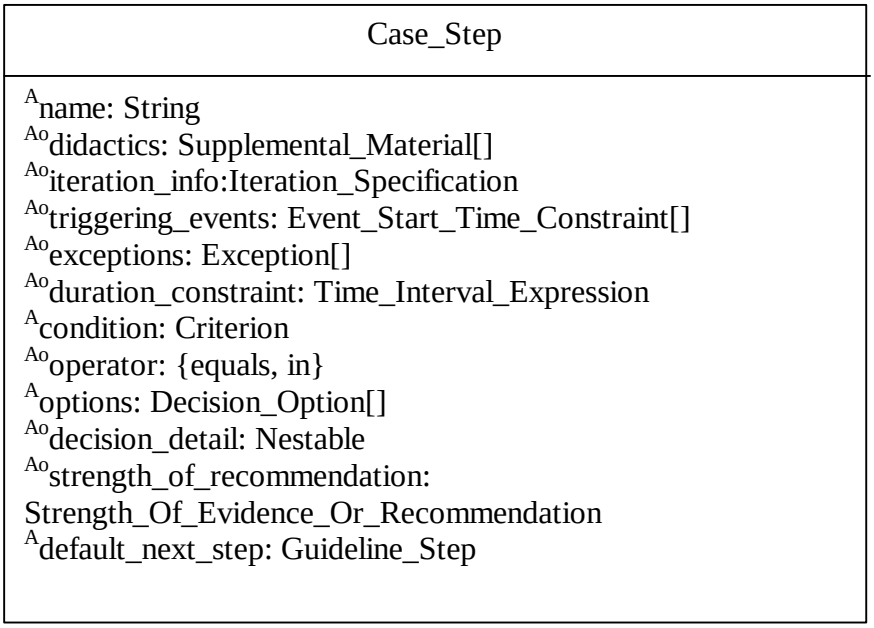


Figure 26. Case Step class diagram

A case step contains a condition, which is an expression. The condition's value determines the control flow to one of a set of possible guideline steps, which are specified by the options of the decision step. The structure of the option class is shown in Figure 16. The condition is compared, using an operator, to the options.condition_value. If the condition matches one of the decision options' condition_value then the control can flow, in the case of a choice step, and must flow, in the case of a case step, to the guideline step that is specified by that decision option's destination. If the condition does not match any of the set of values specified by the decision options or, if available data do not allow evaluation of the condition, then the control flows to the default destination guideline step.

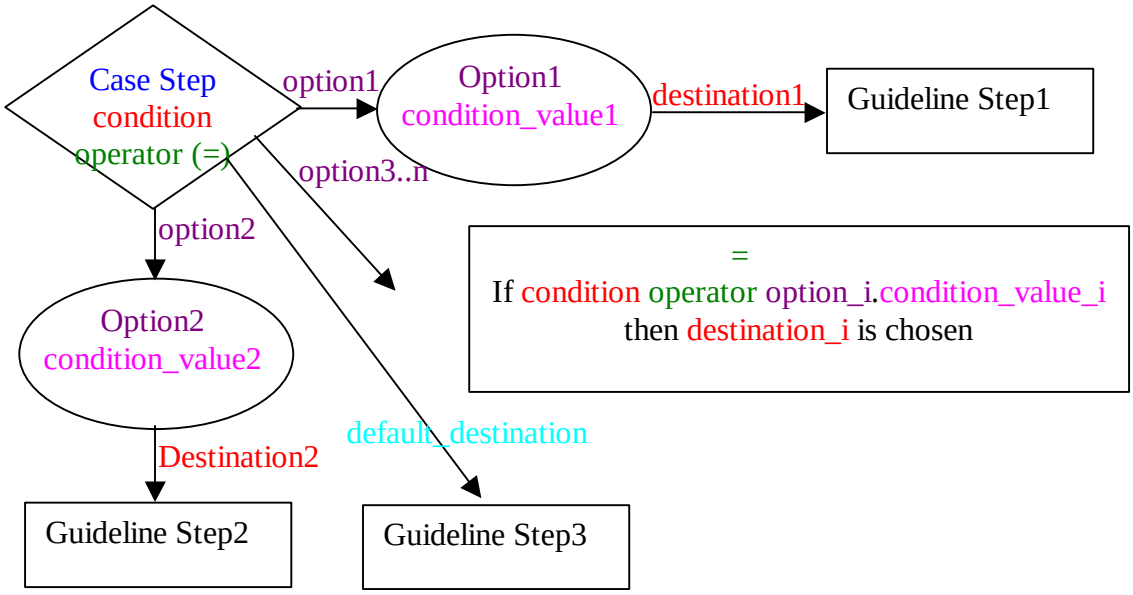


Figure 27. The way case steps are modeled in GLIF3

Decision_Option
[A]name:String [A]condition_value: Decision_Condition [A]destination:Guideline_Step [Ao]strength_of_recommendation: Strength_Of_Evidence_Or_Recommendation

Figure 28. The Decision Option class

Note that the decision options are not guideline steps, and are therefore not graphically depicted as flowchart nodes. Instead, the graphic authoring tool display of the algorithm (flowchart) shown in **Figure 27** is shown in

Figure 29.

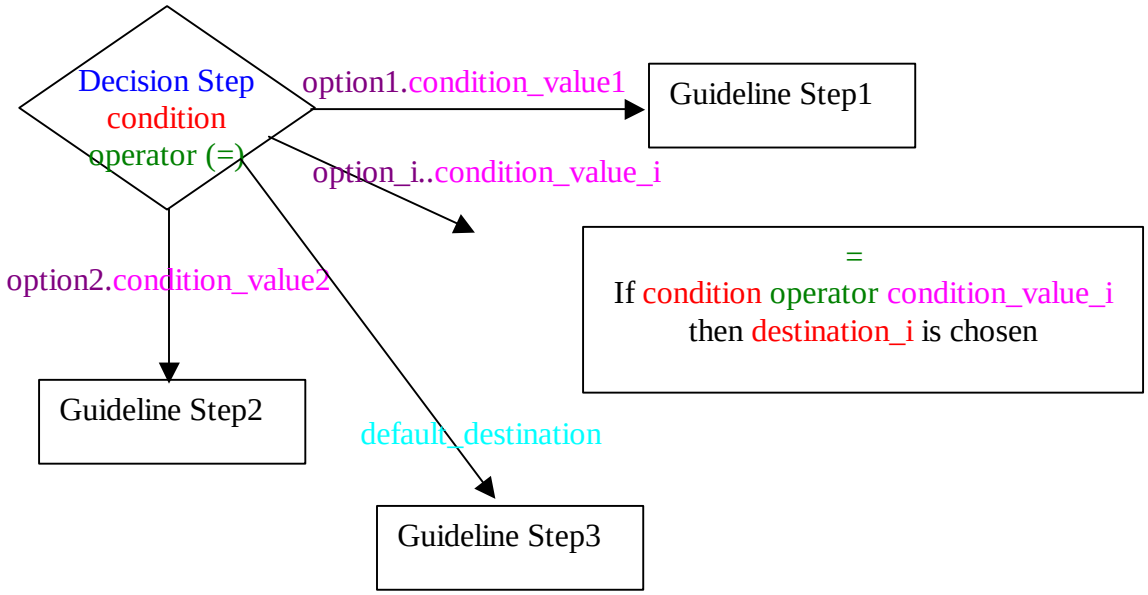


Figure 29. The way decision steps are graphically displayed by GLIF authoring tools

An example of a case step is shown in Figure 30 through Figure 34.

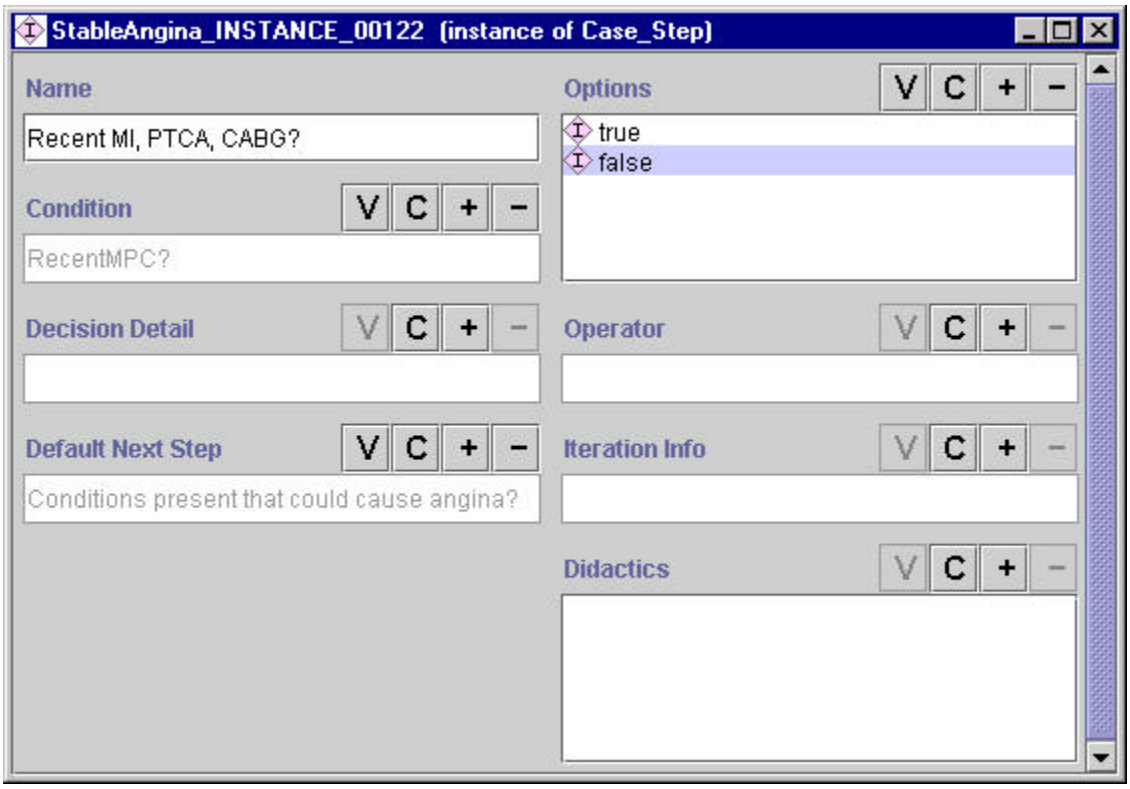


Figure 30. An example of a case step. This is one of the case steps shown in Figure 7.

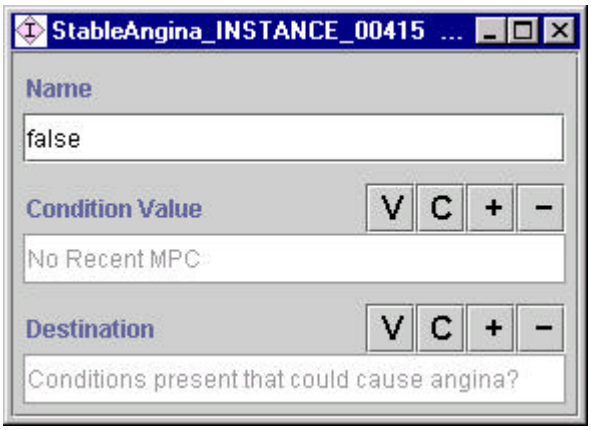


Figure 31. The details of the "false" option shown in Figure 30.

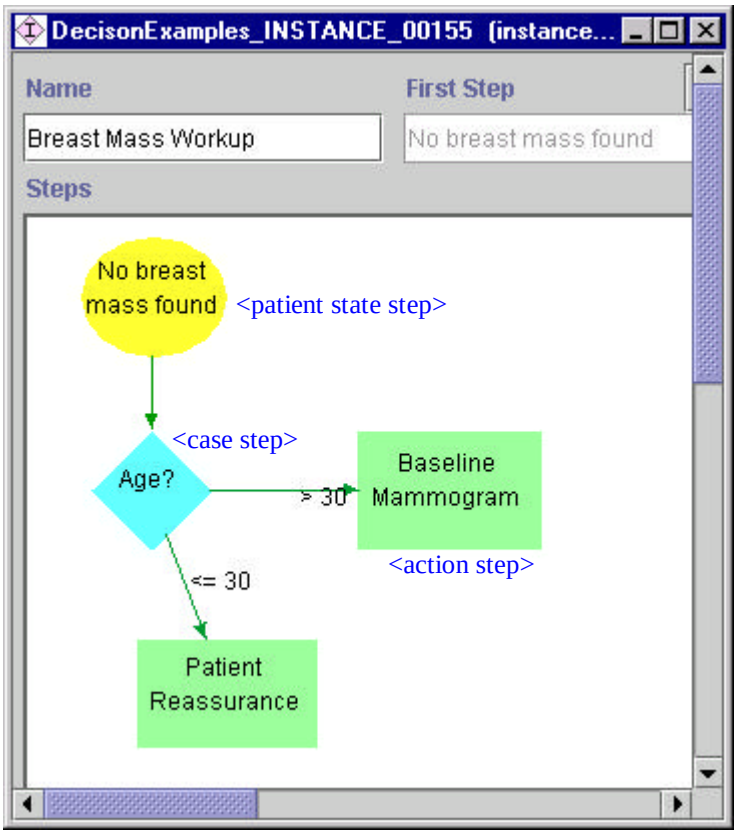


Figure 32. Case step used in the Breast Mass Workup algorithm. Only part of the algorithm is shown

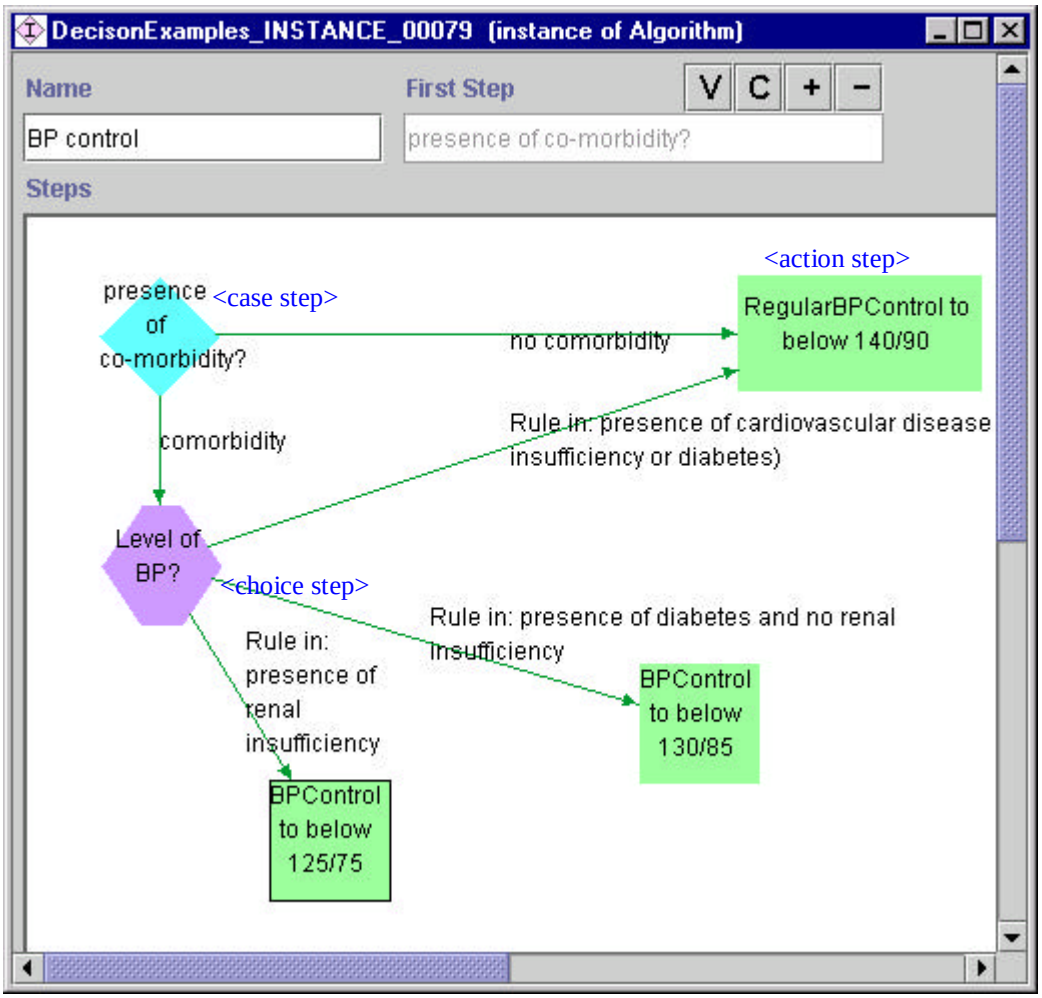


Figure 33. Case step used in the BP Control algorithm

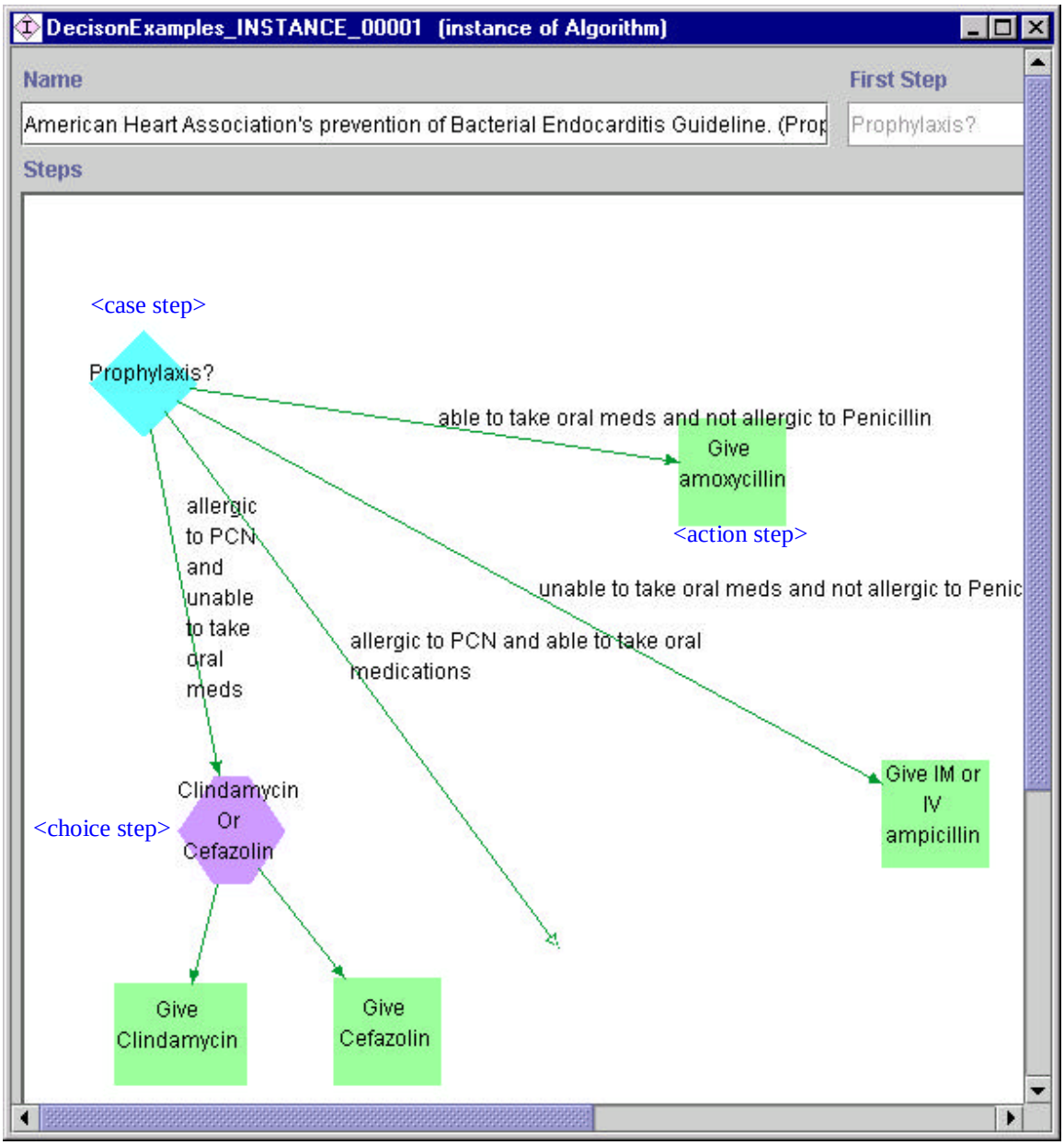


Figure 34. Case step used in the prevention of bacterial endocarditis algoeithm

The case step is a sub class of the decision step. The attributes of a case step are the same as those of a decision step.

The value of the decision option is a decision condition, which can be a choice or a case-condition. Decision conditions are classified as either case destinations or choices depending on whether the decision step that contains it is a case step or a choice step, respectively. The options' condition value attribute of a case step's options must be of type Case_Condition.

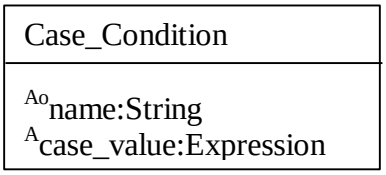


Figure 35. The Case Condition class

The data type of the Case_Step.condition and the case_value expressions have to match. For example, as shown in Figure 32, the expression "Age" is compared with tow options: ">30y" and "< 30y".

	Case_Step.condition	operator	case_value
Expression	Age	in	> 30 y
Data type	Numeric interval		Numeric interval
Actual value of the expression	[65y, 66y)		(30y, infinity)

The expression Age in >30 y is a three-valued-criterion (evaluates to True, False, or Unknown). In our example, [65y, 66y) in (30y, infinity) is evaluated to True.

The condition is the expression whose value is compared to the different decision options.
A comparison operator is used to compare the case expression to the different case values of the case step's destinations. If an operator is not specified, then the "equals" operator is used.

The operator was added so that different comparisons would be possible. For example the operator may be "equals", or may be "in" to represent a comparison to an interval (age in (30,40) as opposed to dose equals "30").

4.1.2 Choice Steps

Choice steps represent a non-deterministic decision between guideline steps. The different decision options are not necessarily mutually exclusive. Even if the decision options are mutually exclusive, the decision has to be made by a person, and cannot be made automatically. Examples of choice steps are shown in Figure 33 and Figure 34.

In a RuleInChoice, when a decision option's condition_value's rule_in and strict_rule_in match the choice step's condition, then control may flow to the destination specified by that decision option.

If the criterion matches more than one of a set of values specified by the choices, a person will have to determine the next guideline step at run-time. Only one guideline step will be chosen to be the next step to be executed.

Ranking the decision options depends on the class of choices. Each option contains a degree of preference that may be modeled differently for the different types of choices. The degree of preference will determine how the choices will be ranked. This will assist the user in choosing among the different options.

The choice step is a sub class of the decision step. The attributes of a choice step are the same as those of a decision step. The options.condition_value of a Choice_Step must be of class Choice.

4.1.2.1 Utility_Choice_Step

Utility Choice step is a subclass of the Choice step. It represents a choice step that uses the Utility theory in deciding among several options. It contains a pointer to the actual algorithm used to evaluate the choices. This may either be a decision analysis tree or an influence diagram.

The utility choice step has the same attributes as the decision step, but adds the decision_diagram attribute. The options.condition_value of the utility choice step must be of class UtilityChoice.

Utility Choice Step Class
Same attributes as decision step [B]decision_diagram_url:String

Figure 36. The Utility Choice Step class

There are 3 subclasses of the Choice class: RuleIn Choice, KofN Choice, and Utility Choice.

4.1.2.2 Rule In Choice

RuleIn Choice
[Ao]name: String [A]rule_in: Criterion [Ao]strict_rule_in: Criterion

Figure 37. The RuleIn Choice class

An example of a RuleIn Choice is shown in Figure 33.
The rule in criterion is a criterion, that if is evaluated to true, rules in favor of taking this choice option. This value plays a role in ranking this choice amongst the other choices of the choice step.
The strict rule in criterion is a criterion, that if is evaluated to true, rules in favor of taking this choice option. This value plays a role in ranking this choice as best choice if the value is true or worst choice if the value is false.
It is possible to add two more attributes: rule_out and strict_rule_out, but we can always use the "not" operator (e.g., rule_in: not allergic to penicillin or rule_out: allergic to penicillin), and so the rule_out and strict_rule_out attributes are not required.

4.1.2.3 K Of N Choice

KofNChoices contain an array of criteria, each associated with a weight. The weighted criteria for each of the options will determine how an option will be ranked amongst the choices presented to a user at run-time. The sum of the weights for each criterion in a choice has to equal 1. The higher the value of a choice (from 0 to 1), the higher its rank.

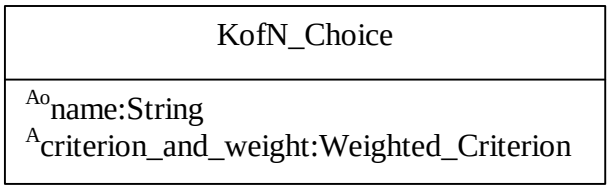


Figure 38. The KofN Choice class

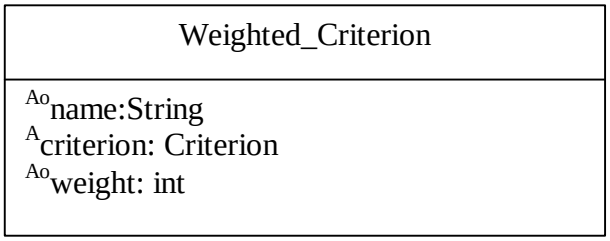


Figure 39. The Weighted_Criterion class

4.1.2.4 Utility Choice

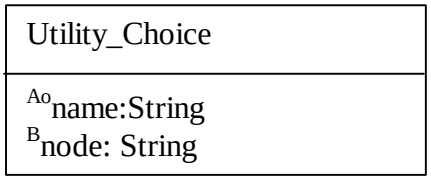


Figure 40. The Utility Choice Class

4.2 Specifying decision criteria

Criteria are expressed using three_valued_criterion_expressions that are written in a superset of Arden Syntax (see Section 3.7). The data items that are referenced by the criteria are specified in the 3-layered domain ontology of GLIF (see Section 2.3).

Suppose that we want to specify the decision criteria: (presumed PNDS in young non-smoker) or pregnant woman or patient recently started ACEI.
The criteria is specified as:

> (latest (**PNDS** where (certainty == 5)).critical_time.high >= now and (now – **Date_Of_Birth**.critical_time.high) < 40 years and **Tobacco_Smoking**.critical_time.high >= now) or (latest **Pregnancy**.critical_time.high >= now) or latest **ACEI**.critical_time.high + 4 weeks >= now

Where:

1. PNDS, Date_Of_Birth, Tobacco_Smoking and Pregnancy are terms that are defined in the domain ontology.
2. “certainty” is an attribute of the class to which PNDS belongs (Observation), as defined in the domain ontology.
3. “critical_time” is an attribute of the class to which PNDS, Date_Of_Birth, Tobacco_Smoking, and Pregnancy belong (Observation), as defined in the domain ontology.
4. “high” is an attribute of the class to which “critical_time” belongs.

5. “presumed” is a legal value for the certainty attribute.
6. “40 years” and “4 weeks” are literal data items that match the type of the attribute critical_time.high.

4.3 Defining patient data

In the above example, PNDS is a term that is defined in the domain ontology. We will show how this patient data item is defined in the 3-layered ontology.

PNDS:

Core_GLIF: PNDS is an instance of Variable_Data_Item with the following values:
name: PNDS
referring_concept: Post_Nasal_Drip_Syndrome
RIM: USAM
ontology: USAM_Third_Layer
owner: 330-60-8713
data_value: an instance of the RIM class to which the referring_concept belongs. This info (to which class the referring_concept belongs to) is specified in the ontology.
service_cd: // no need to define here. It is defined in the third layer. The third layer is searched for the referring_concept Post_Nasal_Drip_Syndrome
mood_cd: event
id: 1.2.3.4 // made-up number
status_cd:
activity_time:
critical_time: (05-01-00, infinity)
recording_time:
method_cd:
body_site:
interpretation_cd:
confidentiality_cd
certainty: 5
severity:
relationships:
value:
derivation_expression:
normal_range:

USAM-RIM: PNDS is an instance of the Observation class as defined above.

USAM_Third_Layer: Post_Nasal_Drip_Syndrome has the following attributes:
term_id: 2.3.4.5 // made-up code
vocabulary_id: UMLS
Post_Nasal_Drip_Syndrome is defined in the following is-a hierarchy: Post_Nasal_Drip_Syndrome is-a Disease_Or_Syndrome_Term is-a Clinical_Event_Term is-a Observation_Term is-a Patient_Data_Term.

5. Describing actions

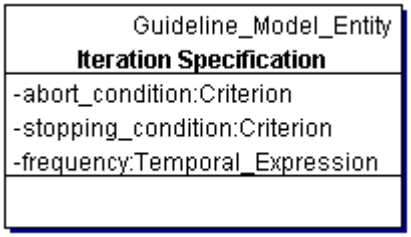
5.1 Specifying the action and parameters

See section 3.3(Action Steps).

5.2 Iterative actions (and decisions)

Package *Iteration_Package*

Class Diagram



The Iteration_Specification class specifies information regarding the loop structure of the iteration. The action- and decision steps that should be iterated, that references the Iteration_Specification, are iterated until the abort condition or stopping condition criteria hold. The iterations are carried out at a certain (fuzzy) frequency (e.g., every 8h +/- 30min or, 3 times a day). If the iteration does occur within the specified frequency, then the iteration time points remain according to schedule, and are not reset (e.g., the patient came in after 8 1/2 hours. The next iteration should take place after 7 1/2 hours). Some frequencies may have a specified allowed offset. This means that if the iteration began outside the fuzzy frequency, but within the offset, then the iteration takes place and the iteration points are reset. For example, the patient came back after 9 hours. He gets treated (since the offset is 8 +/-2 h) but the next iteration is 8 hours late.

Considerations:
1. Executing an action at irregular intervals can be either modeled as an iteration or not. For example, a visit schedule that says: "visit every 5 weeks for 5 months and then every 2 weeks" is an iteration, but an immunization schedule may be modeled as an action (Immunization) that is event triggered (and not as give shot every 1 month for month, thaen every 2 months for 2 months, etc.).
2. Sometimes one wishes to give 1-2 pills every 4-5 hours, but no more than 8 pills within 24 hours. The iteration is still every 4-5 hours, but the dose that is determined for each iteration is dependant on previous doses and may be equal to zero.

An example of an iteration specification is shown in Figure 41

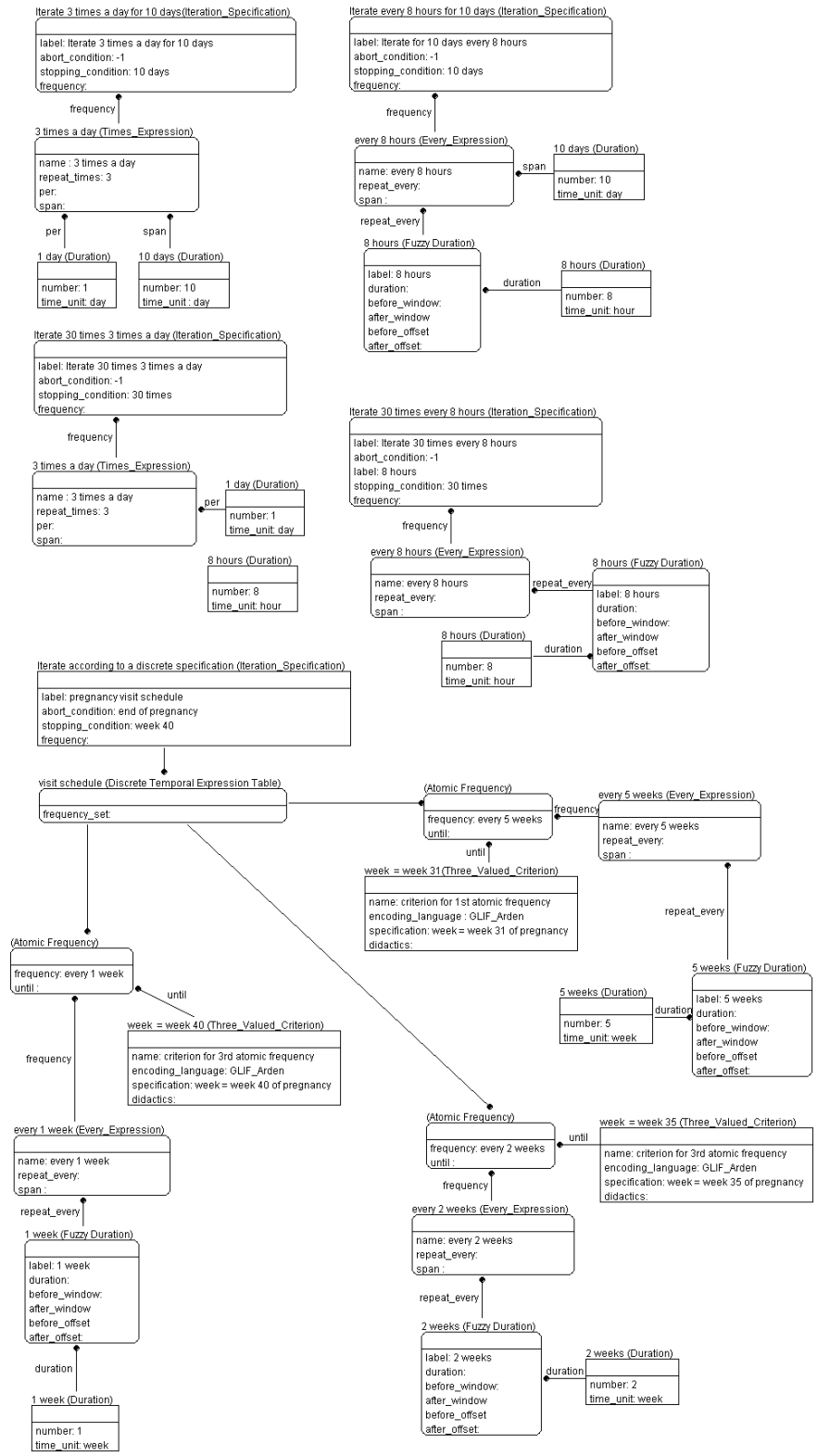


Figure 41. Examples of Iteration Specifications including loose, tight and discrete temporal criteria. (a) Iterate 10 days every 8 hours; (b) Iterate 10 days 3 times a day; (c) Iterate 30 times 3 times a day; (d) Iterate 30 times ev ery 8 hours; (e) Iterate according to a discrete specification.

5.3 Action Specifications

The action specification model includes two types of actions: (1) guideline-flow-relevant actions, such as calling of a sub-guideline, or computing values for data; and (2) clinically relevant actions, such as making recommendations. Clinically relevant actions reference the domain ontology for representations of clinical concepts such as prescriptions, laboratory test orders, or referrals.

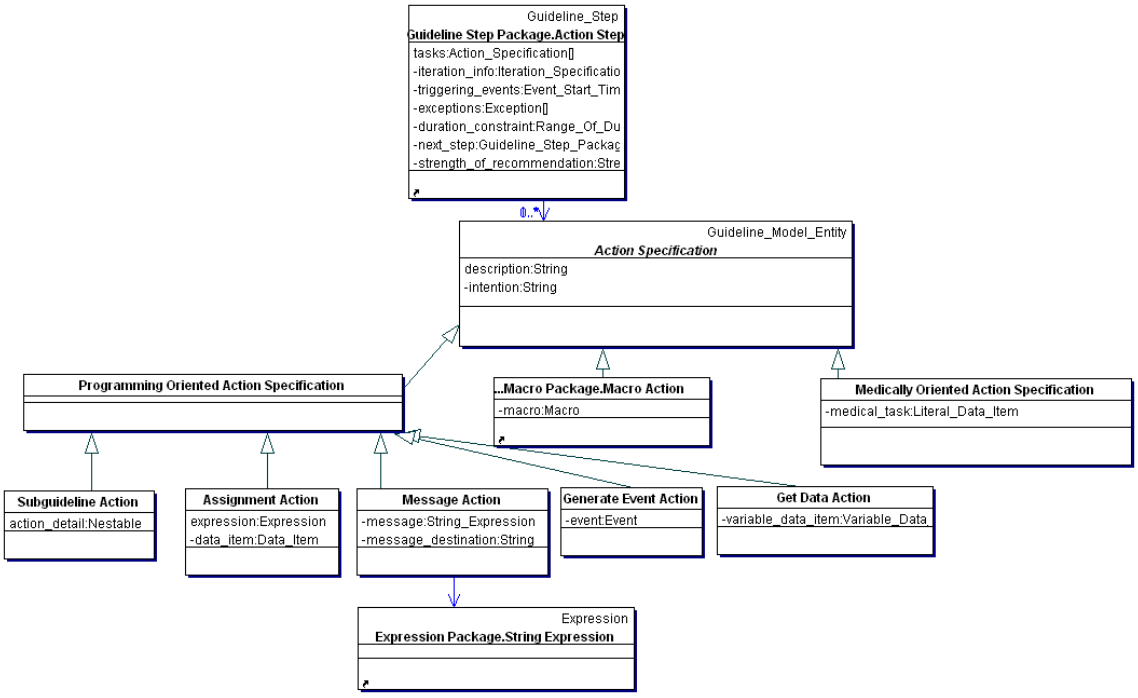


Figure 42. The action specification package

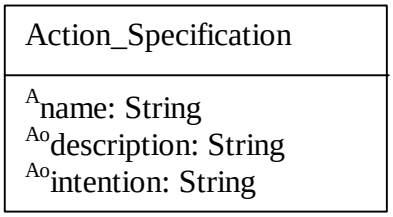


Figure 43. The Action Specification class

All action specification objects specify the details of a clinical action. The Action Specification class is abstract. Its subclasses are Programming oriented action specifications and medically oriented action specifications.

5.3.1 Subguideline Action

This is a programming oriented action specification, that shows the details of an action specification in the form of a (sub)guideline or a macro.

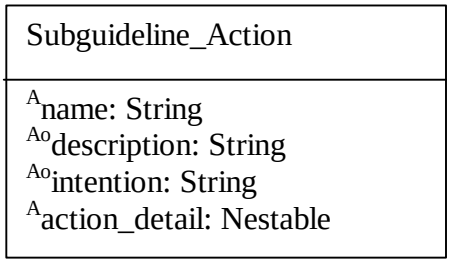


Figure 44. The Subguideline Action class

5.3.2 Assignment Action

This is a programming oriented action specification. The Assignment_Action class is used to create/instantiate a new data item. This data item is assigned the value resulting from the evaluation of the expression.

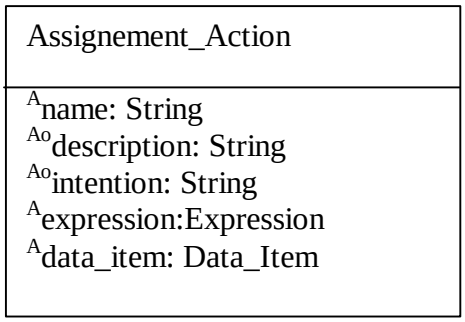


Figure 45. The Assignment Action class

5.3.3 Message Action

This is a programming oriented action specification. The Message_Action class is used to send a message to a user.

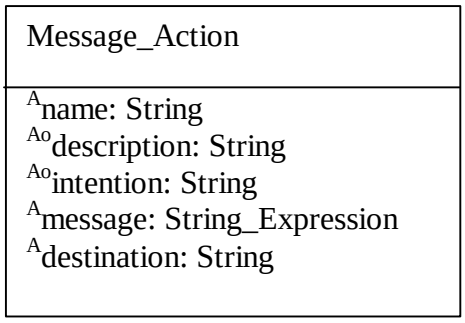


Figure 46. The Message Action class

5.3.4 Generate Event Action

This is a programming oriented action specification. The Generate_Event_Action class is used to create an event, such as a data item was written to the EPR.

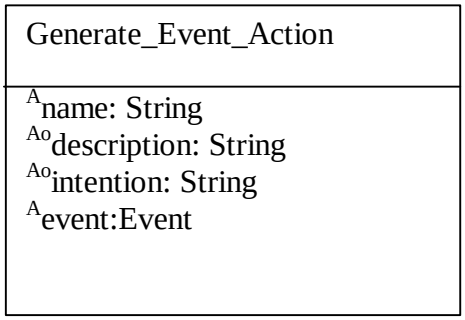


Figure 47. The Generate Event Action class

5.3.5 Get Data Action

This is a programming oriented action specification. It is used to explicitly obtain the value of a data item.

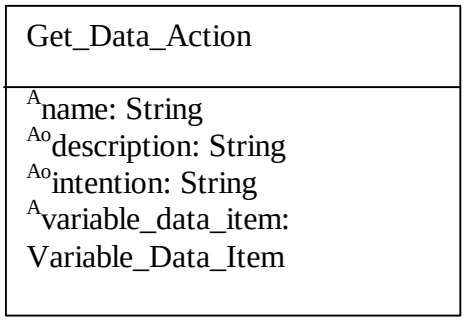


Figure 48. The Get Data Action class

5.3.6 Medically Oriented Action

This is a programming oriented action specification. The Medically_Oriented_Action class is used to define an action that refers to a medical term

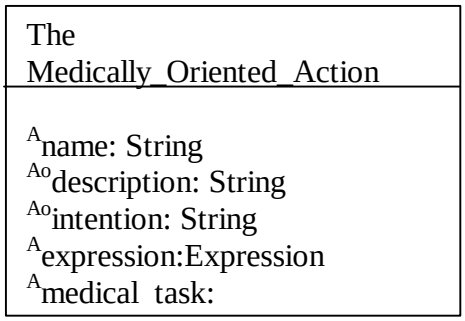
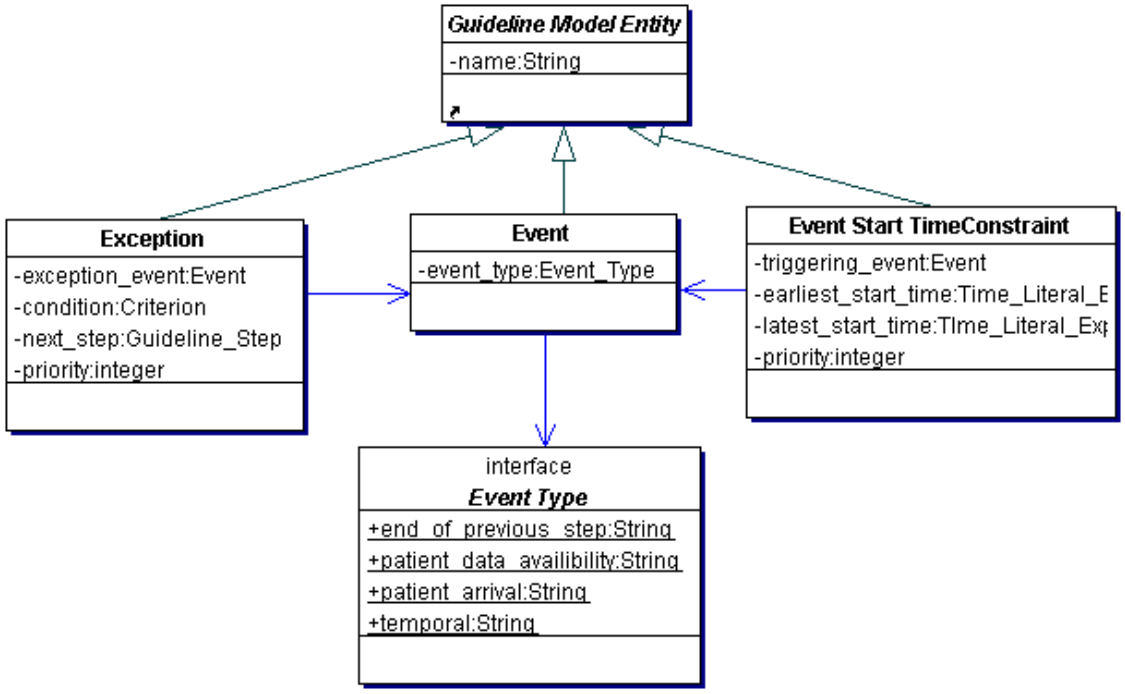


Figure 49. The Medically_Oriented_Action class

6. Events, Exceptions and states

6.1 *Specifying events and exceptions*

Class Diagram



Action- and decision steps have an attribute, called triggering_events, which specifies what events trigger the start of the step, and with what temporal constraints. The triggering events are of the class EventStartTimeConstraint. This class specifies an event and the earliest and latest times after which the step should be started following the occurrence of the triggering event.

The event class contains a triggering event and the earliest and latest times after which a system response to the event should be generated. The event triggers action or decision steps within the latest_start_time and not before the earliest_start_time. Any of the triggering events that occur can trigger the step that is about to be triggered. In cases where several triggering events are defined (each with its own start time constraints), their priorities are compared, and the highest priority event is chosen to trigger the step. Different Event types are defined: end of a previous guideline step, patient arrival, patient data availability, and temporal events, such as a certain point of time has arrived.

Considerations:
Currently, complex events are not modeled. To express an event such as "1 hour after event E1", we use the triggering event as E1 and an earliest start time of 1 hour. Complex events such as "DowJones changed by 20% in any 2 hour interval" are not possible to model right now.

Examples of events are shown in Figure 50.

Action- and decision steps have an attribute, called exceptions, which specifies what exceptions should be checked during the execution of the step. The exceptions are of the class Exception.

This class specifies the exception-event that should be checked for, a (guarding) condition and a next step. If the exception event occurs and the condition holds, then we terminate the step associated with the exception, and move on to the next step that is defined by the exception.

Any of the exceptions that occur can stop the execution of the current step and pass control to another step. In cases where several exceptions are defined (each with its own next_step), their priorities are compared, and the highest priority exception is chosen to trigger the step.

An example of an exception is shown in Figure 51.

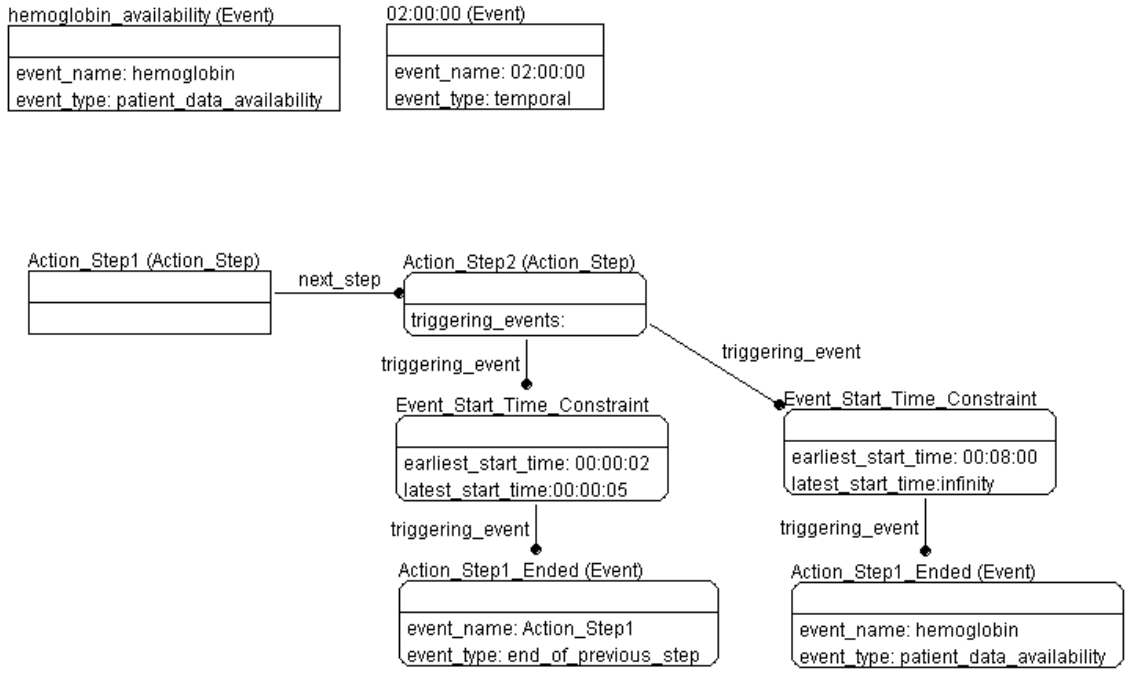


Figure 50. Examples of events. 1) hemoglobin data is available; 2) 2 am arrived; 3) Action Step2 is invoked by one of two events: (a) at least 2 seconds and not more than 5 seconds after Action Step1 ended; (b) at least 8 minutes after hemoglobin data is available.

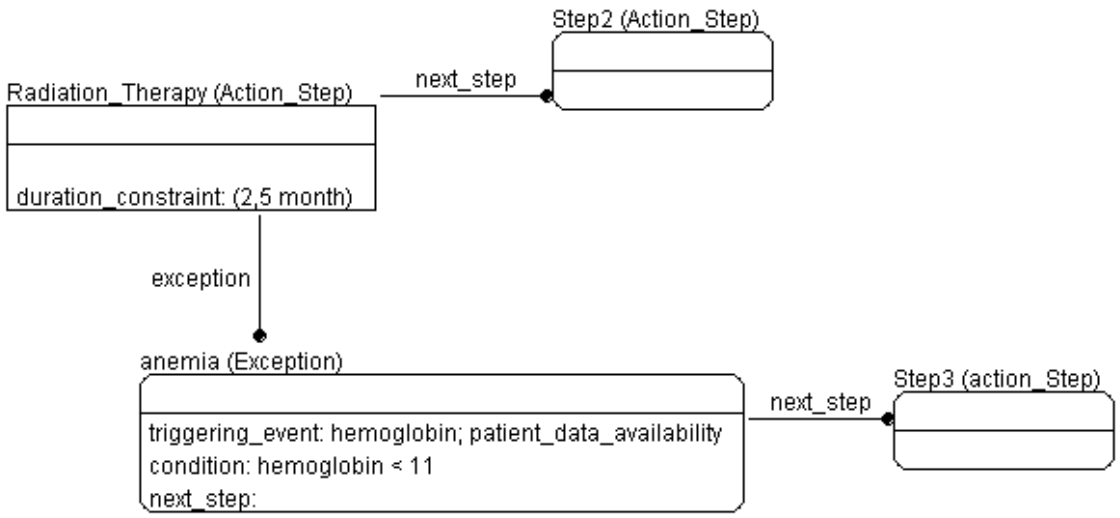


Figure 51. An example of an exception. **When radiation therapy is conducted, you check for the exception of anemia (hemoglobin result with a value of < 11). If it occurs then you go to Step3,. If it doesn't you finish radiation therapy and go to step2.**

6.2 Describing patient states

A Patient State Step is a guideline step (a node in the flowchart) that is used for two purposes. One is as a label that describes a patient state that is achieved by previous steps. This way, a guideline may be viewed as a state transition graph, where states are scenarios, or patient states, and transitions between these states are the networks of guideline steps (excluding patient state steps) that occur between two patient state steps. The other purpose of a patient state step is an entry point to the guideline (e.g., patient came back to the clinic at clinical state A).

A patient state step has a criterion that describes the state of the patient who is at that patient state. If there is a criterion that refers to a generalization (e.g., "state is not well") it also applies to specializations of that class (e.g., "state is fever"). The hierarchy of terms is defined in the domain ontology.

A patient state step is followed by a guideline step.

An example of a patient state step is given in Figure 53, Figure 54 and Figure 55.

Patient_State_Step
[A]name: String [Ao]didactics: Supplemental_Material_Package [A]label: String [A]patient_state_description: Criterion [Ao]next_step: Guideline_Step

Figure 52. The Patient State Step class

When a patient arrives at a clinic, his current state is compared to the last patient state that was recorded for him. If he is not at that state, then the patient state steps that represent new encounters are searched. These can be determined either by an implementation-level attribute

called "new_encounter" of type Boolean, which characterizes a patient state step (implementation level attribute) or by looking at patient state steps whose next-step is triggered by an event of type “new patient encounter”.

It is important to acknowledge the fact that a patient might not follow the guideline precisely, and that he/she may be treated also outside the regular clinic.

In EON’s guideline model, a patient state step has a reference advice guideline that offers advice as to what tests should be done to further evaluate the patient's state. This advice guideline is different from a regular guideline in 2 respects. (a) it is not required that this advice guideline be followed; (b) it does not include activities. This special reference advice, or consultation guideline, is used in order to avoid using the branch step. Having a consultation guideline as an attribute of a patient state step overlaps in functionality with the branch step. It allows the possibility that the physician sees the recommendations listed under the consultation guideline but can follow these recommendations (or not) up until the time that the patient enters another patient state. We would like to use the branch step, and so, we do not need to use consultation guidelines, and will stick to using just one kind of guideline for now.

We commented out the new_encounter attribute. The guideline author would probably not want to specify whether a patient state step represents a new encounter or not; this would probably be institution specific, and needs to be determined during localization/ implementations. Instead of specifying that a patient state step represents a new encounter, we can alternatively specify a triggering event of type: “patient encounter” that will trigger the next step pointed to by the patient state step.

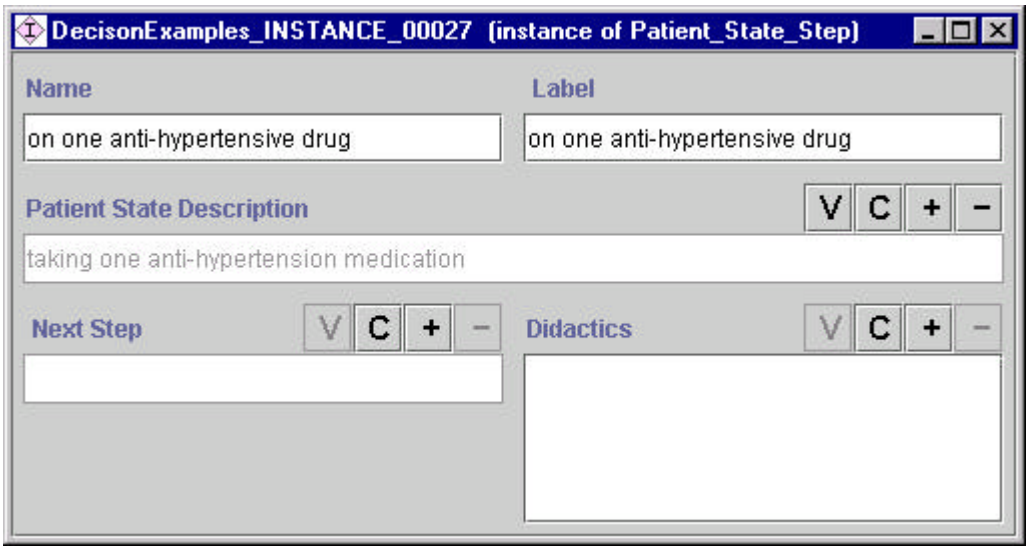


Figure 53. An example of a patient state step

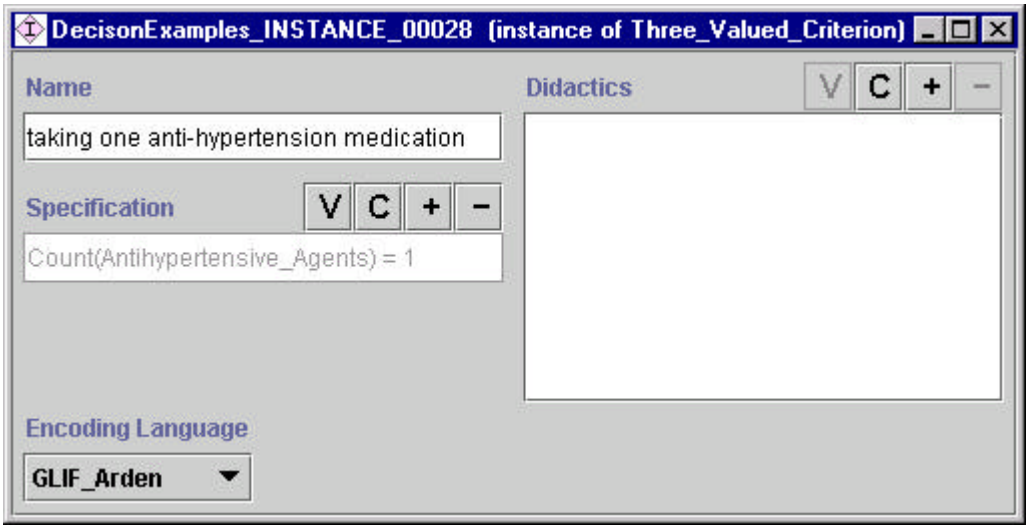


Figure 54. The criterion used to characterize the patient state step of Figure 53

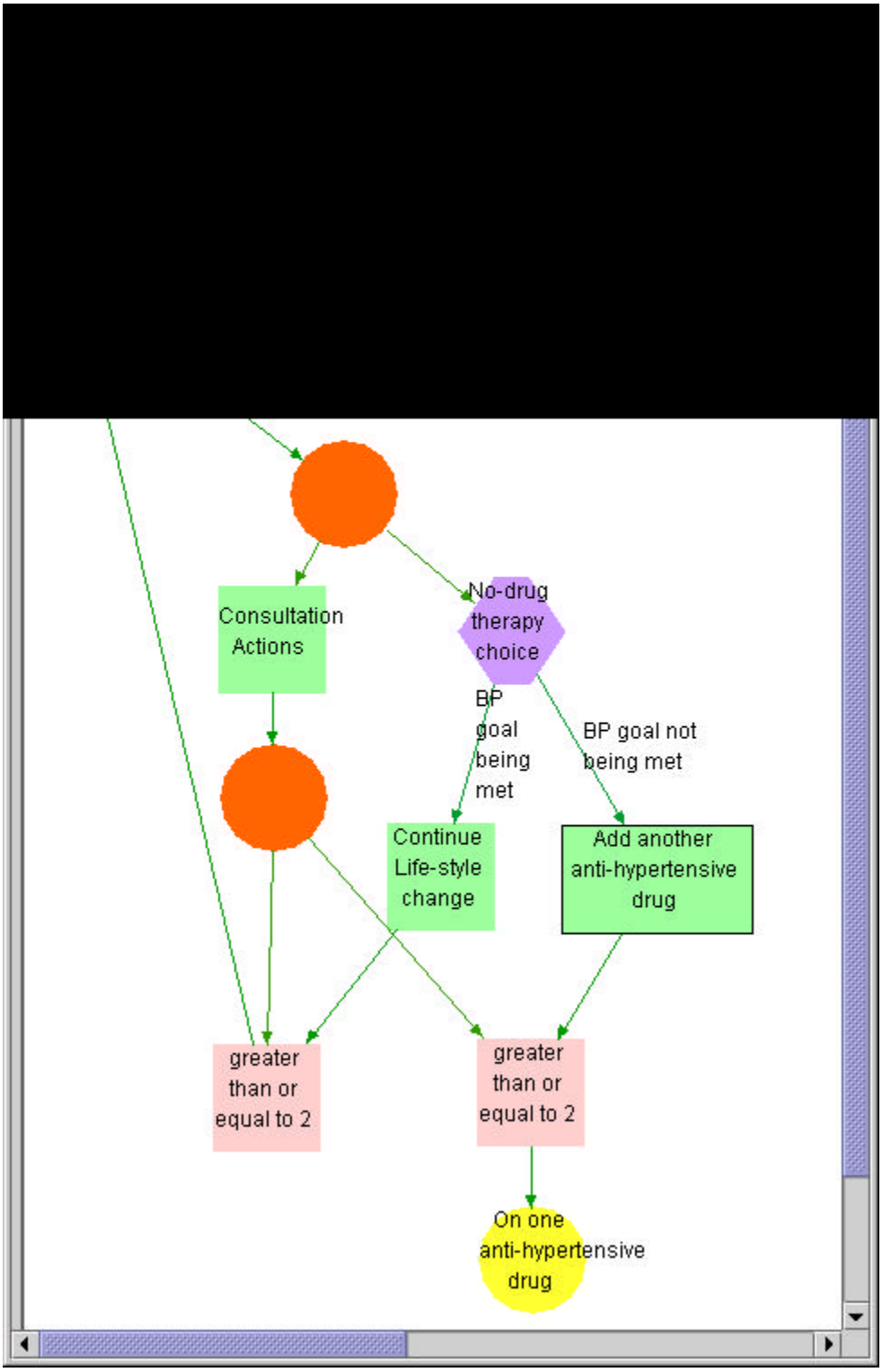


Figure 55. A hypertension guideline showing transitions between two patient state steps

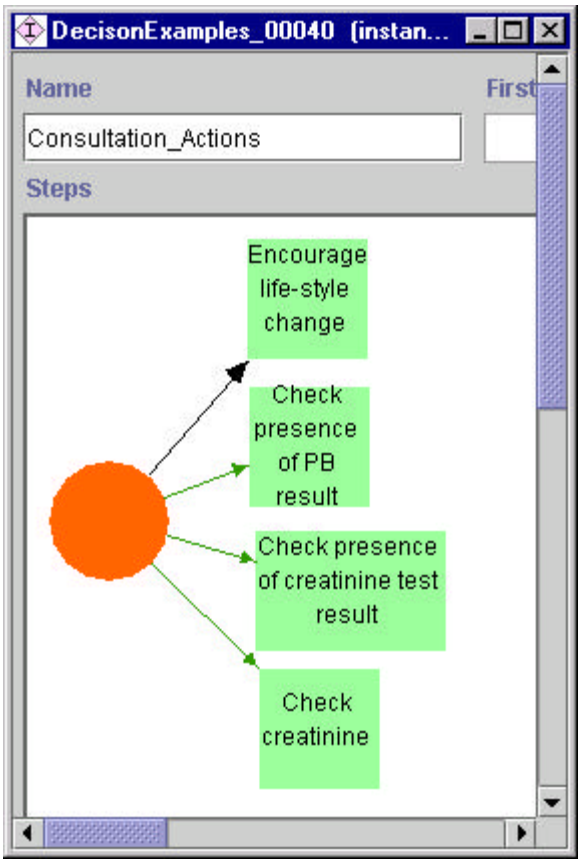


Figure 56. The consultation actions shown in this figure are executed in parallel. This is a zoom -in view of the consultation action shown in Figure 55.

7. Parallel paths in a guideline

7.1 Branching to multiple paths

See Section 3.5 (Branch Steps).

7.2 Synchronizing from multiple paths

See Section 3.6 (Synchronization Steps).

8. Dealing with complex guidelines

The Nestable class is a superclass of Guideline and Macro. It is an abstract class. Both Guideline and Macro are guideline model entities that can be nested. Nesting allows grouping of parts of a guideline into modular units (subguidelines or macros). This enables partitioning the guideline parts into manageable sized units that can be more easily comprehended. These modular units can also be reused by other guidelines.

The details of action and decision steps of a guideline can be shown in a different guideline that serves as a subguideline of the first guideline. The details of action and decision steps of the second (sub)guideline can also be given by other (sub)guidelines.

Macros enable declarative specification of a procedural pattern that is realized by a set of primitive GLIF steps. Most importantly, macros can be used to represent patterns of domain level concepts. The information contained in a single macro object can be mapped to a guideline object containing underlying GLIF steps (i.e., not containing macros).

Nesting is very useful for managing the complexity of guidelines. Nesting enables looking at a guideline from a top-level view and then zooming into/out of some of its parts. Nesting is also useful in representing a guideline in the context of other guidelines. Since nesting allows grouping of parts of a guideline into a single unit, this is a mechanism that can allow model extensibility and reuse of part of a guideline (defining macros), or adaptation of a guideline to a specific institution by replacing specifications for parts of a guideline (i.e., replacing a goal with a procedure).

8.1 Nesting decisions

Decision Steps are nested by specifying a subguideline in the decision_detail attribute of the step. This subguideline is executed before the decision criterion for that step is evaluated. The subguideline would modify or create new variables and assign them values. The use of these variables in the decision criteria makes the decision nested.

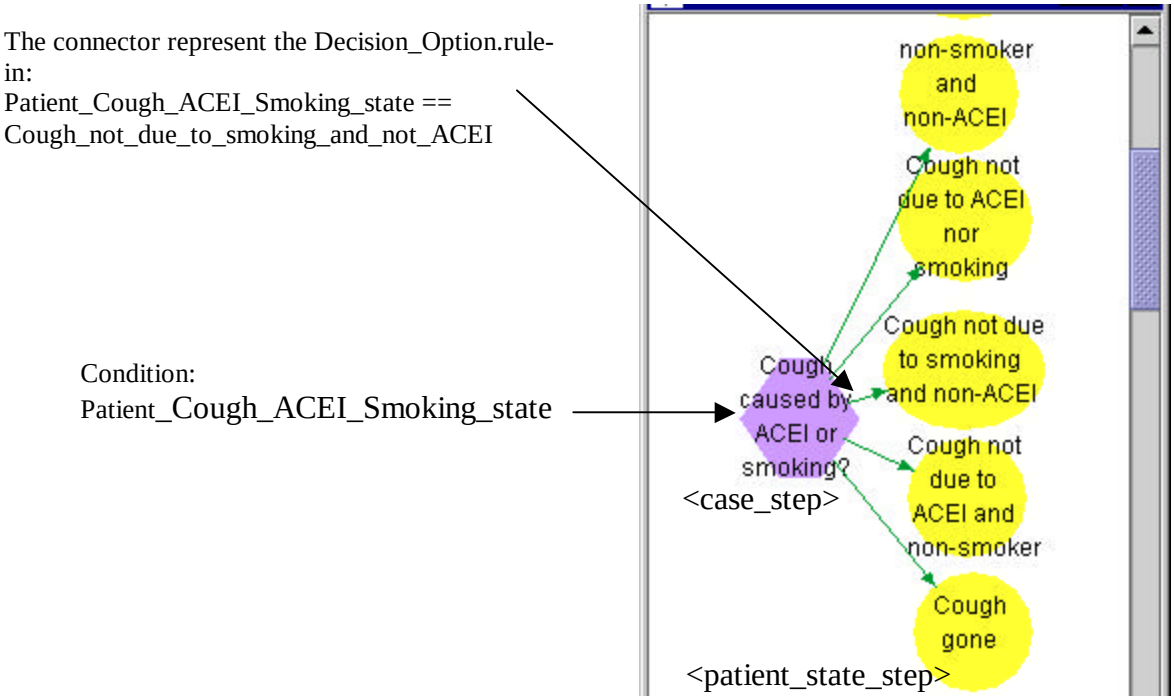
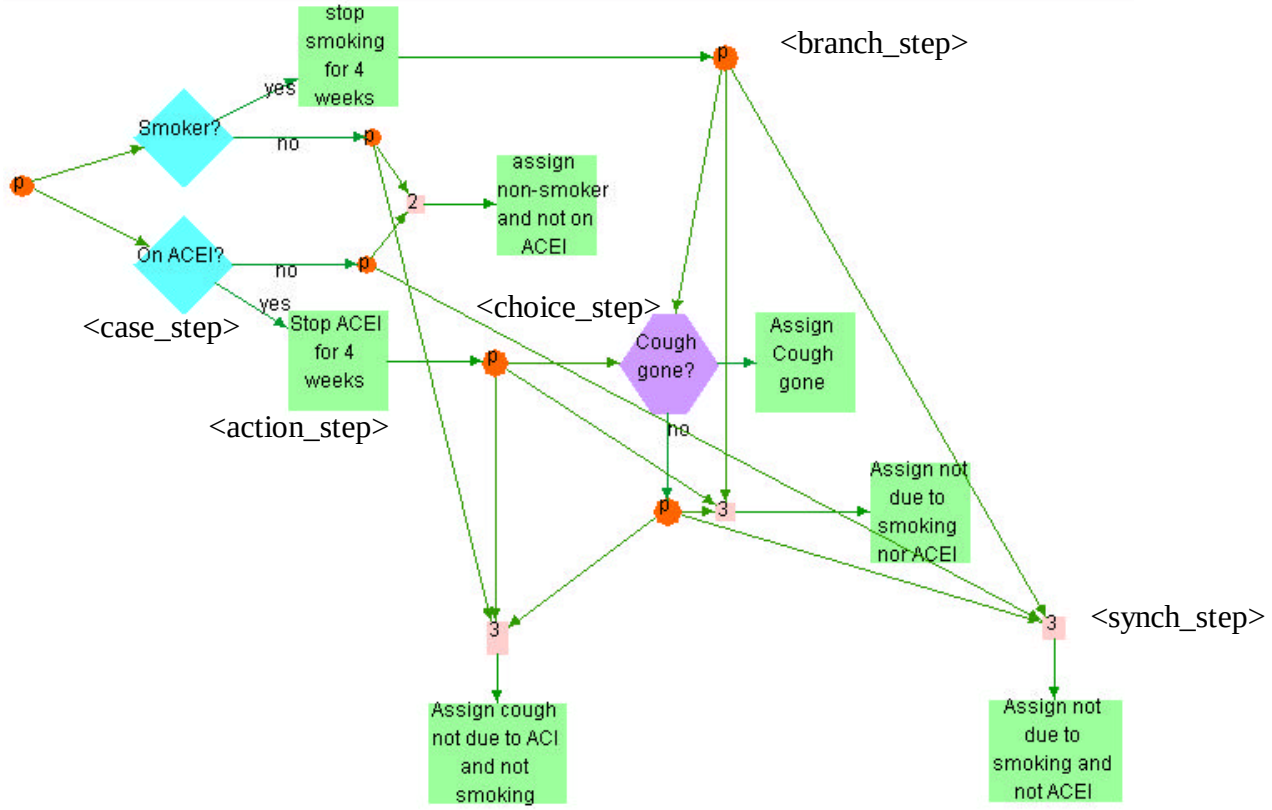


Figure 57. Atop-level view of a nested decision step.

A zoom-into view of the Case Step shows:



Data Item Name: Patient_Cough_ACEI_Smoking_State
Expression: Cough_not_due_to_smoking_and_not_ACEI

Figure 58. This is the detailed view of the decision step, shown in the previous figure. This subguideline determines state of the patient in terms of his cough, smoking, and ACEI use. The leaf steps of this subguideline assign the cough_smoking_ACEI value to a new data item named Patient_Cough_ACEI_Smoking_State using the Assignment_Action. The data item Patient_Cough_ACEI_Smoking_State that is created by the subguideline is used by the main case step in its case expression. The value is used by the case destinations to select the next step of the outer guideline.

8.2 Nesting actions

Action Steps are nested by including a Subguideline_Action type of task in the step. The Subguideline_Action task has a subguideline attribute that contains the nested subguideline.

Figure 59 shows an example of nesting an action step, for complexity management purposes, while Figure 60 shows an example of nesting which is used for adjusting a local procedure.

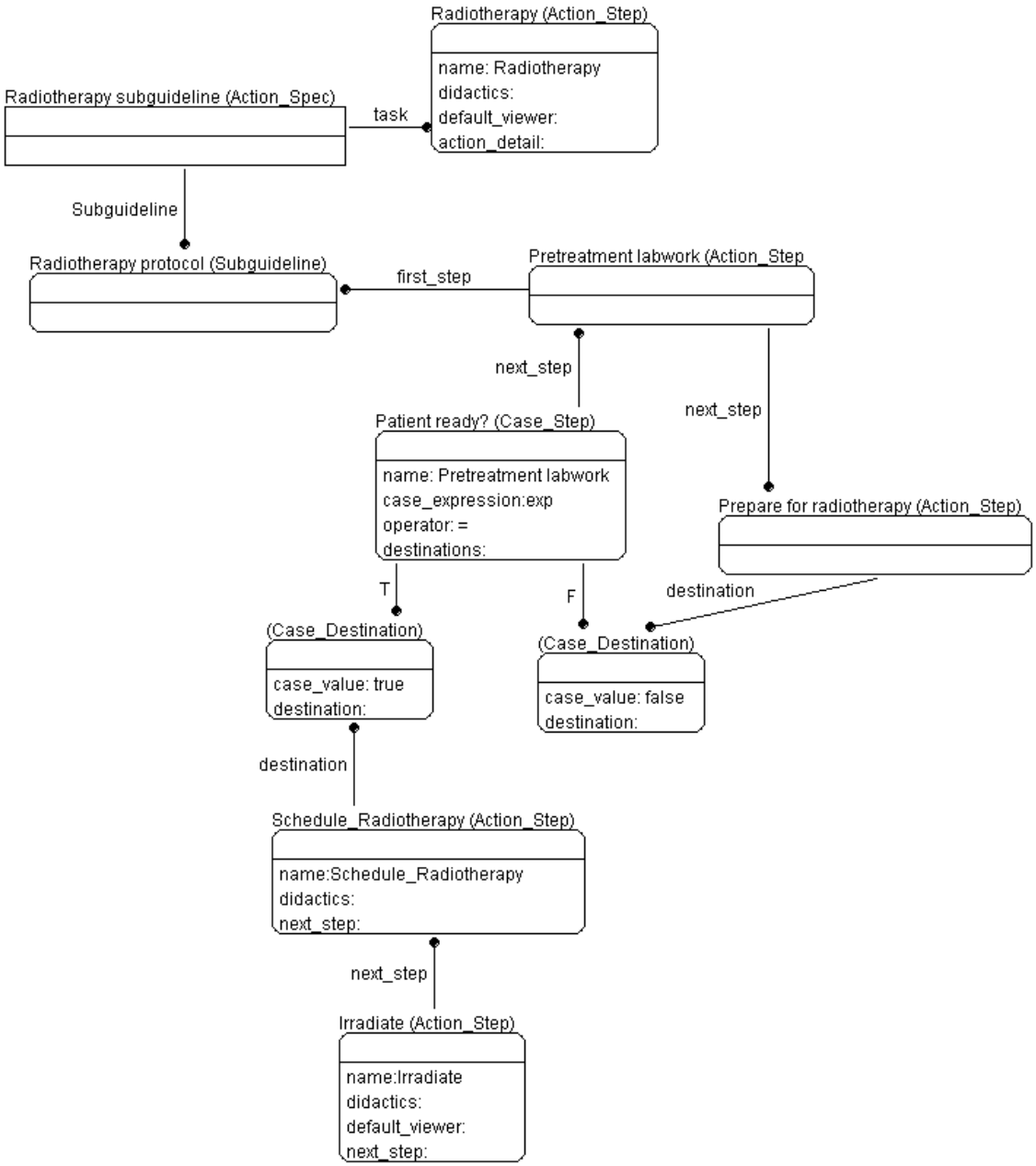


Figure 59. Nesting of an action step, for complexity management purposes.

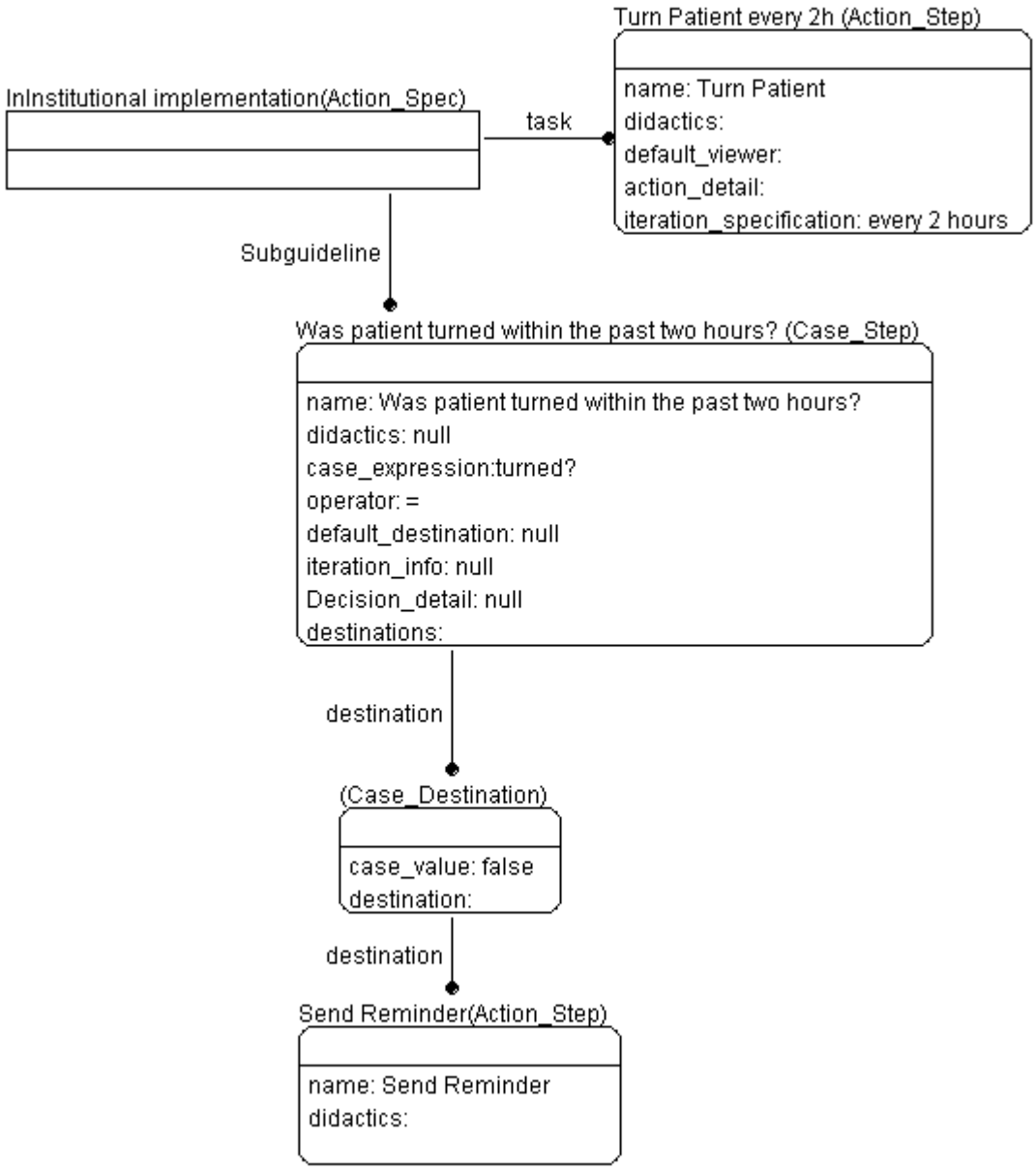


Figure 60. Nesting of an action step, for adjusting a local procedure.

8.3 Macros

Package *Guideline_Step_Package.Macro_Package*

A macro is a special class with attributes that define information needed to instantiate a set of underlying GLIF steps. Macros can be used to represent patterns of domain-level concepts. Macro steps benefit authoring, visual understanding, and execution of guidelines. They also enable declarative specification of a procedural pattern that is realized by a set of action, branch, and decision steps.

The schema defines the mapping of a macro to the underlying GLIF. The GLIF pattern is used to generate instances of GLIF objects that can be executed or viewed as such. The schema is also used for validation of macro attributes.
The schema consists of a single action step or decision step. The action step or decision step may contain a nested subguideline of any complexity.

Considerations:
A notation/language for schemas still has to be developed. This language will be used to describe schemas abstractly. The definition of this language will require significant effort, which we will defer until other aspects of GLIF are more fully developed.
We are considering a notation similar to the XML schema definition syntax for the World Wide Web Consortium (W3C).
The examples provided in this document are only an indication of our current ideas on how GLIF schemas will be described. These are not intended to be specifications.

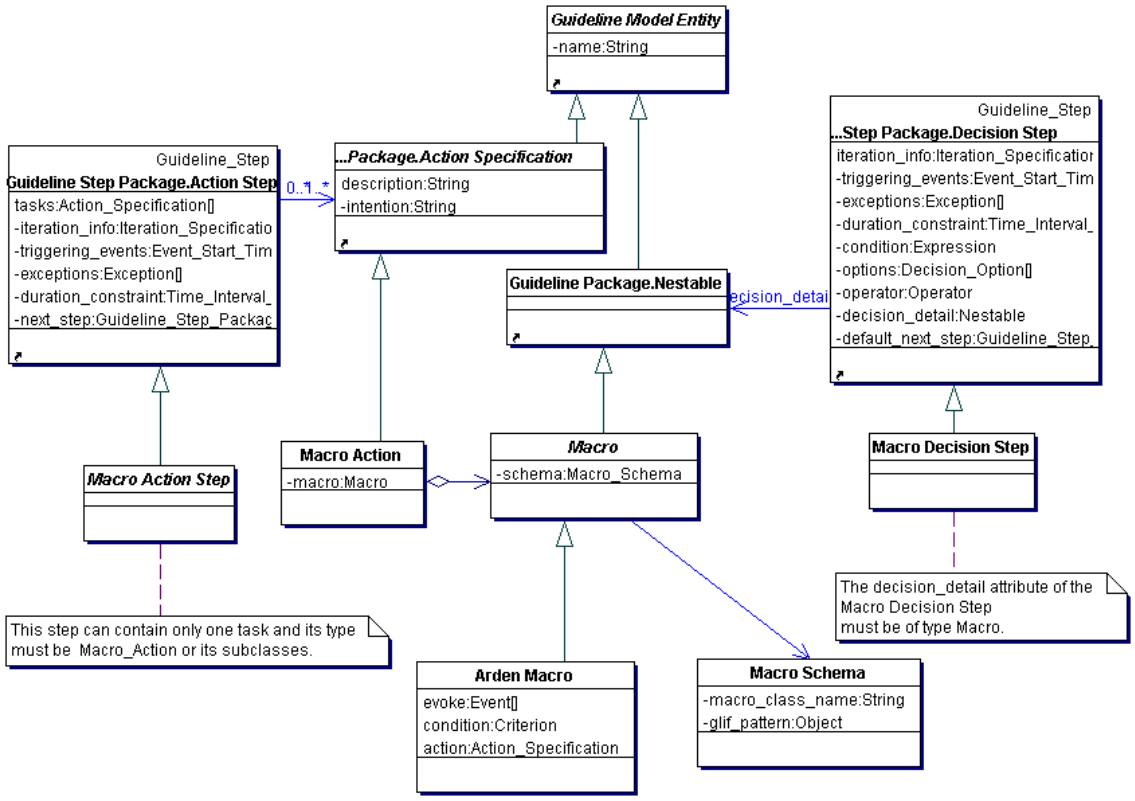


Figure 61. The Macro package

An example of an Arden macro and its expansion is shown in Figure 62 and Figure 63. Figure 64 and Figure 65 show an example of a risk assessment macro and its expansion.

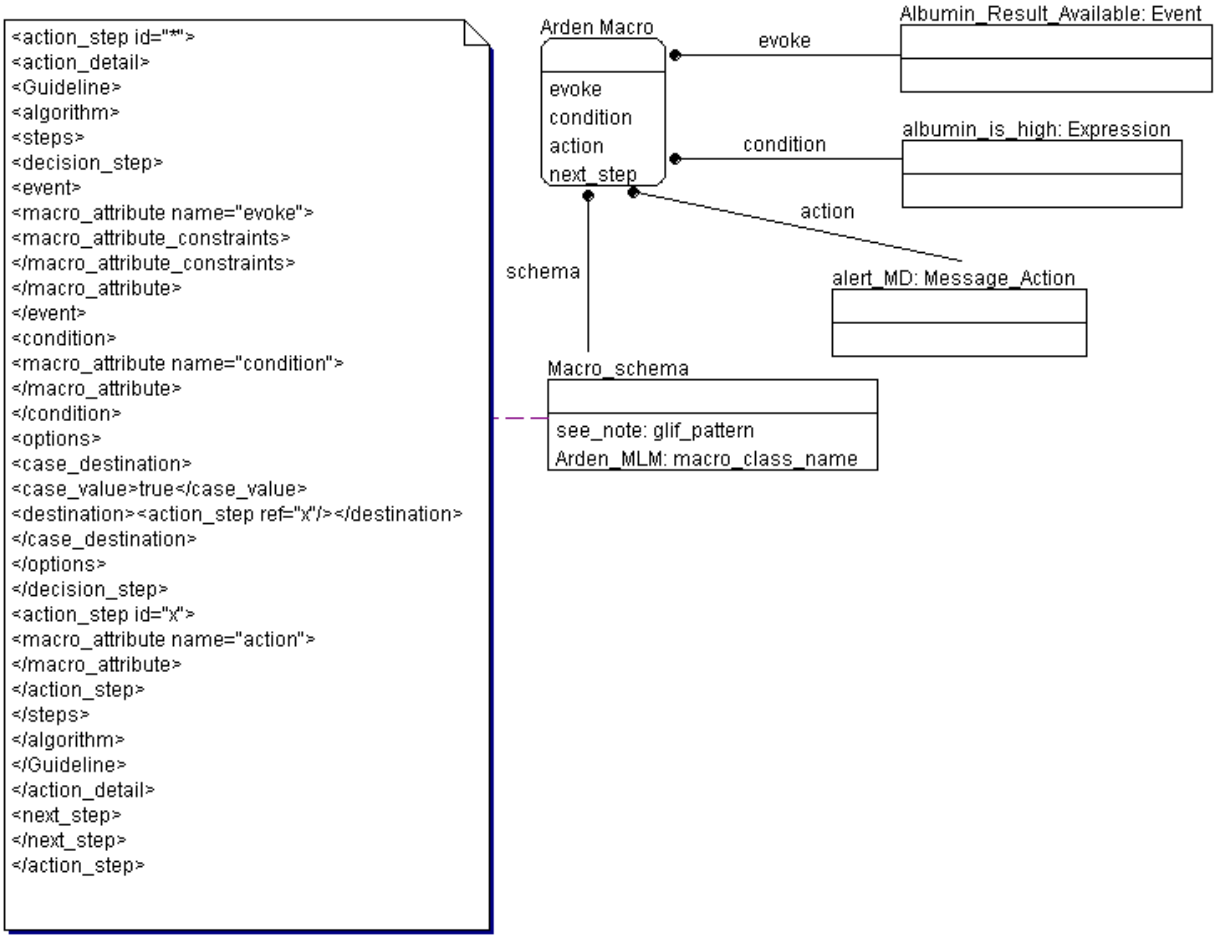


Figure 62. This is an example of a Macro ("MLM Macro") that illustrates how an Arden Syntax MLM can be described with a GLIF Macro. The Arden Macro object (a subclass of MacroStep) has 4 attributes, 3 of which map to MLM attributes, and one attribute which links in the next step. The three MLM attributes are evoke, condition, and action.

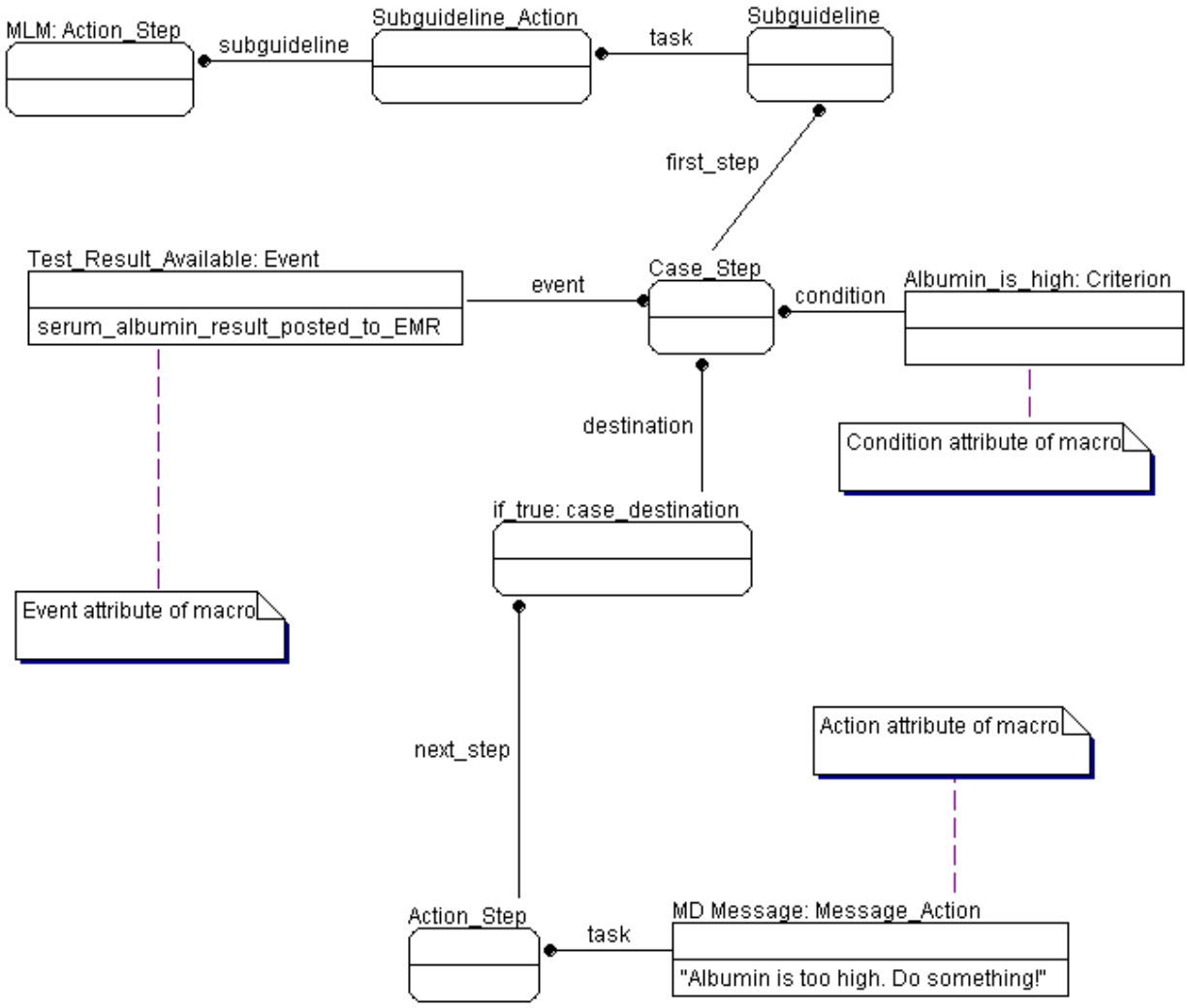


Figure 63. **This MLM macro from MLM Macro 1 is expanded into a full GLIF object.**

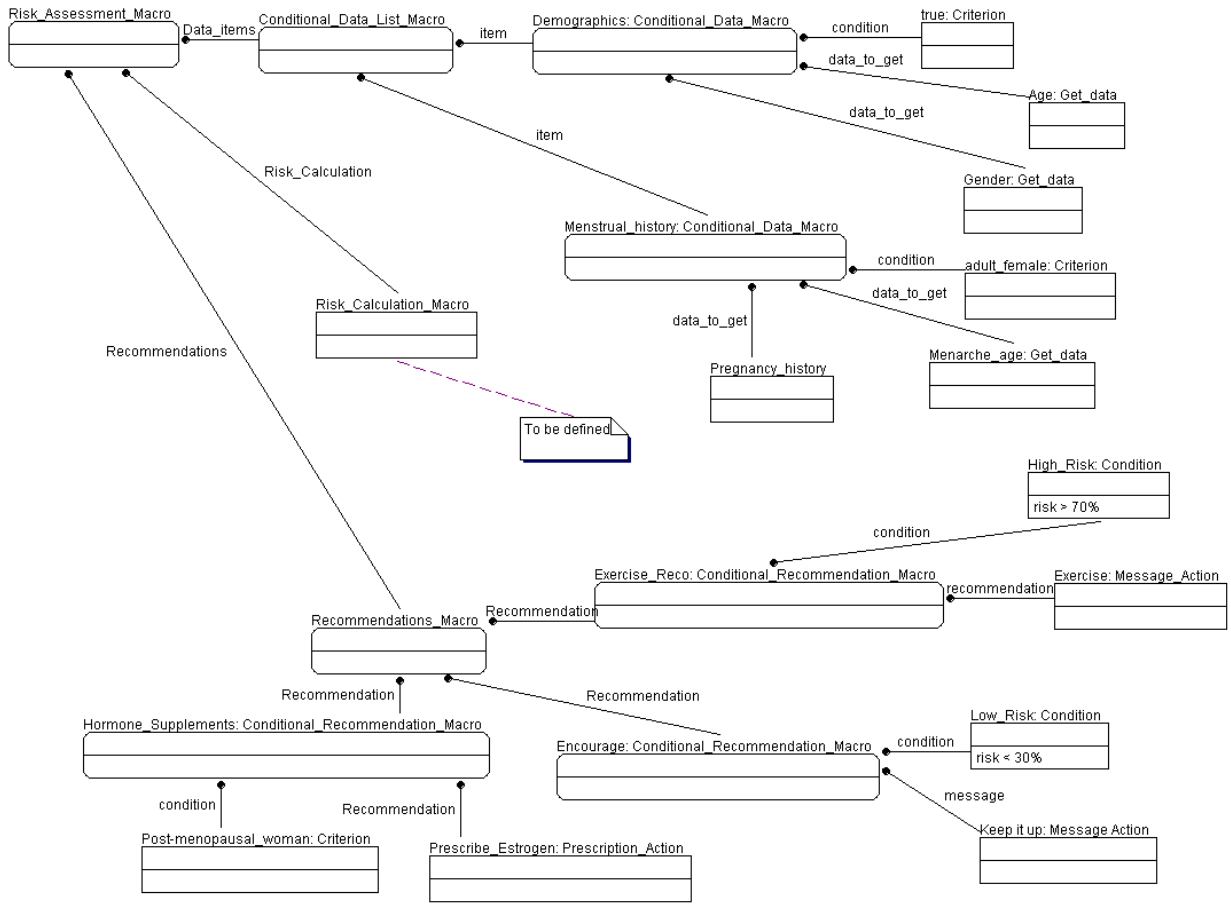


Figure 64. **A generic risk assessment guideline contains three parts or "steps". Each of these steps can be modeled as a macro.**

1. **Collecting patient data** - Data that is needed for calculating risk is collected in this step. The data may be obtained conditionally based on values of previously collected data. In this example, demographics data is obtained for all patients. Menstrual history is obtained only if the condition adult female is true.
   The data collection step is modeled using an ordered list of Get_Conditional_Data macros. This macro contains a condition and a list of patient data items that must be obtained.
2. **Computing risk** - The risk calculations are performed in this step. This macro has to be defined. It would contain definitions of variables that are to be created and the calculation of those variables in Assignment_Actions.
3. **Recommendations** - Recommendations based on computed risk and individual risk factors are provided in this step. Recommendations are provided only if an associated condition is true. The Exercise_recommendation is provided only to high risk persons. Thus, the Recommendations_Macro is structurally similiar to the Get_data_macro.

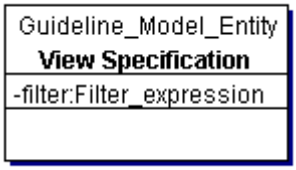
Figure 65. **This example expands the Risk Assessment Macro into its full GLIF implementation. That is, the macro has been expa nded into more atomic GLIF steps.**

8.4 Views of a guideline

For each guideline default viewers may be specified. Since different users may be interested in different parts of a large, complex guideline, differential display capability is supported. This capability is provided through the use of filters that collapse segments of the guideline into a default view of the guideline customized to a given user, situation, etc. Default viewers are specified using a View_Specification.

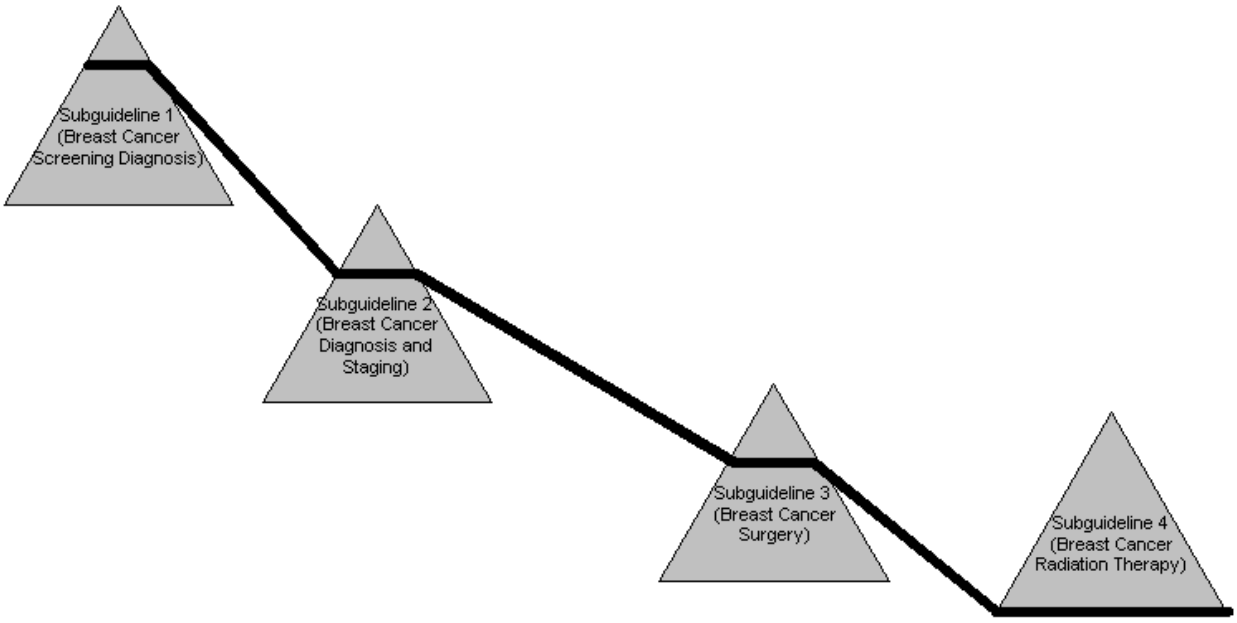
Package *Views_Package*

Class Diagram



A capability to provide multiple views of the same guideline was added in GLIF3. Since different users may be interested in different parts of a large, complex guideline, differential display capability is supported. This capability is provided through the use of filters that collapse segments of the guideline into a default view of the guideline customized to a given user, situation, etc.

Views describe the amount of detail (i.e. level of nesting) displayed by default for each of the subguidelines.



If a guideline consists of Subguidelines, each of these subguidelines may be visualized as a triangle, with one step at the highest level and multiple steps at the lowest level. That is, the width of the triangle is proportional to the number of steps at that level of nesting. Top of the triangle = less detail = small number of steps. Bottom of the triangle = more detail = large number of steps.

A given filter (e.g. MD_Radiation_Oncologist) will define the default level of nesting/zooming for each of the subguidelines. It will be up to the guideline author to define the subguidelines in an appropriate way (e.g. to avoid too many steps per screen for a given viewer) and to define the level of nesting required for each given subguideline. In the above example, suppose that a breast cancer guideline has four subguidelines as above. A Radiation Oncologist looking at the guideline may see (by default) relatively little detail about screening, diagnosis and surgery. He will see a great deal of detail regarding radiation therapy, however. A surgeon looking at the same guideline may see little detail re: screening/diagnosis, a lot on surgery and little re: radiation therapy.

The status quo of specialty bodies publishing guidelines may change as multi-specialty organizations publish multi-specialty documents. Guidelines may become quite complex.

Much of medicine is multi-disciplinary in nature. The distinction between specialties is artificial. For example, the distinction between cardiology and nursing is for the convenience of practitioners. The patient suffering a myocardial infarction (heart attack) is likely to require care from both a cardiologist and a nurse. The information needs of the cardiologist, however, are very different from those of the nurse. The purpose of default views in GLIF should be to reveal to the cardiologist only the relevant portions of the myocardial infarction guideline, which may be different from that shown to the nurse.

Consideration:

Views are default filters through which we interact with the guideline. By definition, views do not change guideline logic (e.g. if an RN should do something different from an MD, this should be represented in the guideline logic, not in the view). Although we anticipate that the most common use of views will be user and/or location, there may be other relevant filters (e.g. situation such as routine vs. disaster). The view class is a guideline entity. Alternatively, the view could have been modeled as an enumerated type attribute. The main purpose of this class is to allow differential display in the simplest possible way.

The view specification was chosen to be at the level of guideline entities and not at the attribute level. We may later choose to make attributes (and not entire guideline entities) visible or invisible to some users.

Issues yet to be resolved:

1. Semantics of no default_user specified (e.g. confidential vs. visible to developer only).

2. Issues related to nesting?

The BNF notation for filter expressions:

term: filter_type = domain_ontology_filter_instance

filter_expression: term | expression binary_operator expression | unary_operator expression | (expression)

binary_operator: OR | AND
unary_operator: NOT

filter_type: USER | LOCATION

domain_ontology_filter_instance: MD | RN | ...

An example of a view specification is shown in Figure 66 and Figure 67.

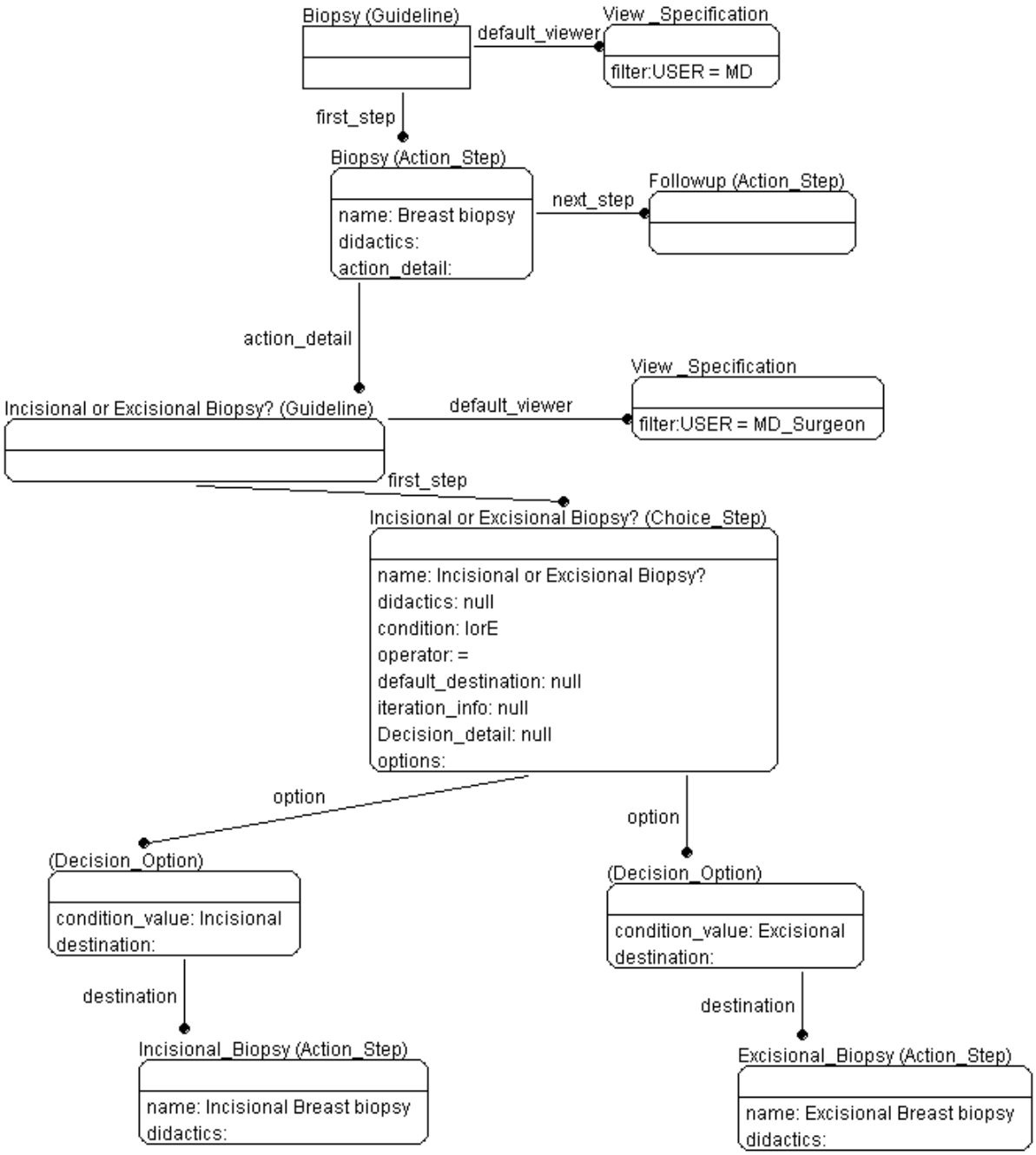
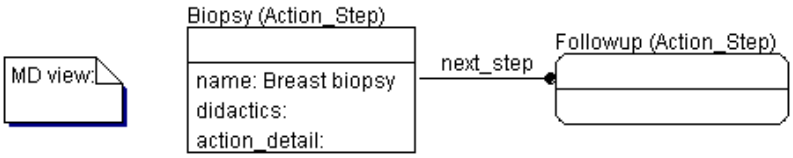


Figure 66. Specifying views: **a guideline might call for a breast biopsy. Lets say that all MDs want to see that a b reast biopsy is called for, however, surgeons want to know what kind of biopsy is needed, incisional or excisional.**


MD view:
Biopsy (Action_Step)
name: Breast biopsy
didactics:
action_detail:
next_step
Followup (Action_Step)

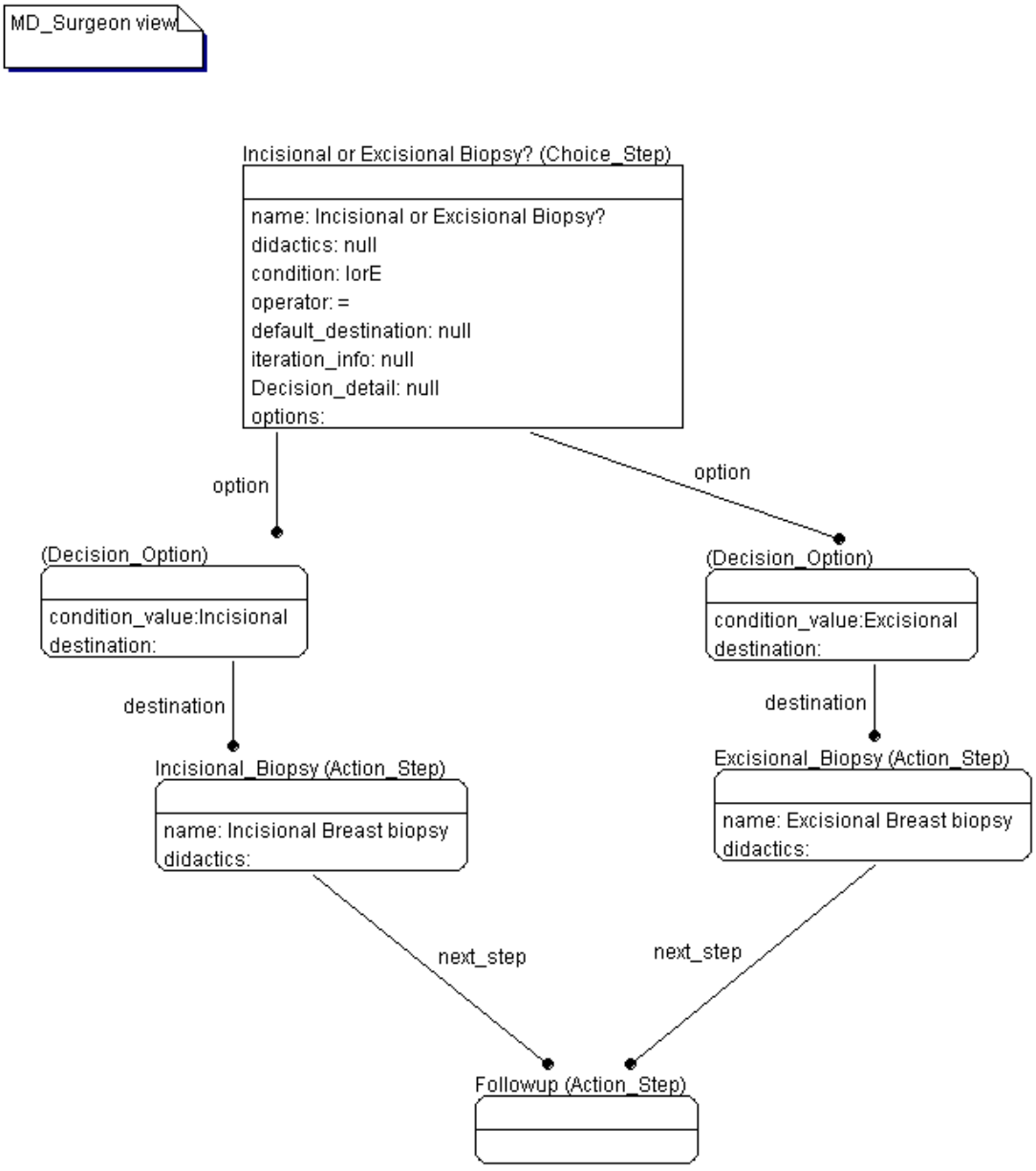

MD_Surgeon view
Incisional or Excisional Biopsy? (Choice_Step)
name: Incisional or Excisional Biopsy?
didactics: null
condition: IorE
operator: =
default_destination: null
iteration_info: null
Decision_detail: null
options:
option
option
(Decision_Option)
condition_value:Incisional
destination:
(Decision_Option)
condition_value:Excisional
destination:
destination
destination
Incisional_Biopsy (Action_Step)
name: Incisional Breast biopsy
didactics:
Excisional_Biopsy (Action_Step)
name: Excisional Breast biopsy
didactics:
next_step
next_step
Followup (Action_Step)

Figure 67. How different users view the guideline. This example shows how nesting deals with views. If the viewer is an MD he sees the top-level view of the action step Biopsy. He can zoom into the action-detail subguideline, to see that incisional or excisional biopsies can be performed. An MD_Surgeon will directly see the zoomed-in view of biopsy directly, showing the decision that is made between incisional and excisional biopsy.

9. XML-based Syntax for GLIF