SAGE Guideline Model Technical Specification

October 8, 2006

SAGE Consortium*

Document Editors:

Samson Tu, Julie Glasgow

* Consortium Partners: GE Healthcare, Apelon, Inc., Intermountain Healthcare, Stanford University School of Medicine, The Mayo Clinic, University of Nebraska Medical Center

Notational Conventions

Class names (e.g., **Guideline**) will be in **bold**.

Slot names (e.g., *label*) will be in *italic*.

Values or possible values of slots (e.g., "`JNC 7 Hypertension Guideline`") will be in `Courier New` font.

The naming conventions can be combined. Thus, a slot value that is also a class will be in **`bold Courier New`** font.

Note that not all words in bold are class names and not all italicized words are slot names.

The grayed-out text describes features of the guideline model not implemented by the SAGE Execution Engine as of November 30, 2006.

SAGE Execution Engine implementation notes are underlined.

1 Introduction

In recent years, clinical guidelines and protocols have gained support as the vehicles for promulgating best practices in clinical medicine. In the hope of disseminating best practices, reducing practice variation not supported by evidence, and minimizing inappropriate use of resources, professional organizations, government agencies, and health-care institutions have been publishing clinical guidelines at an increasing rate. This pace far exceeds our ability to disseminate that knowledge, and far outpaces the ability of individual clinicians to keep up. Similarly, the technology for bringing situation-specific decision support for guideline-based care into clinical settings has not matched the increased flow of guideline production. Organizations which have implemented guideline-based decision-support systems have either created custom software or used commercial software packages that are often little more than if-then rules. Research groups that develop sophisticated guideline modeling formalisms and execution software usually have to limit the application of their technology to their home institutions. Efforts have gone into developing shared models for representing medical decisions and clinical guidelines (Hripcsak et al., 1994l; Peleg et al., 2003) . However, as an experiment to share Medical Logic Modules (MLMs) across two institutions (Pryor et al., 1993) indicated, it takes more than a formalism for medical logic to accomplish sharing of computable medical knowledge. Lack of standards in terminologies and in data models for patient information requires re-coding of significant parts of the MLMs. Boxwala et al. elucidated a set of requirements for sharable guidelines that include the capability to represent different types of guidelines, methodology for local adaptation, integration with institutional systems, use of an explicit medical concept model, allowance for multiple modes of use, and revision management of guidelines (Boxwala et al., 2001). Work in United Kingdom to develop guideline-based decision support for primary care (Johnson et al., 2001a) suggested that reuse of a guideline knowledge base is possible as part of an infrastructure that includes medical record query interface, terminology mediation, unit-conversion mediation, and act interface. The SAGE (**S**tandards-Based Sharable **A**ctive **G**uideline **E**nvironment) project, a collaboration among research groups at IDX Systems Corporation, the University of Nebraska Medical Center, Intermountain Health Care (IHC), Apelon, Inc., Stanford Medical Informatics, and the Mayo Clinic, seeks to build on these and other prior work to create the technological infrastructure for integrating interoperable computer-based guidelines into enterprise clinical information systems. The end points of the project include end-to-end demonstrations of guideline encoding, localization, and execution at Mayo and Nebraska for several complex guidelines.

This document gives an informal description of the SAGE guideline model in the context of the technological infrastructure that the project is developing. We first motivate this work by discussing considerations that led us to develop a new model. Then we give an overview of development methodology of the SAGE project and the SAGE architecture for guideline-based decision support. We describe components of the guideline model with emphasis on how the model interacts with existing standards such as Workflow Management Coalition's process model and with other standard-based components of the infrastructure. This document is accompanied by a set of javadoc-style documentation of every class and slot in the SAGE guideline model. References to guideline model classes will be hyperlinked to the javadoc documentation.

We will use UML diagrams to show components of the SAGE guideline model, and screen dumps from the Protégé knowledge engineering environment to illustrate how the guideline model is instantiated to encode the immunization guideline.

2 Overview of SAGE Project

To accomplish the goal of creating the technological infrastructure for guideline-based clinical decision support, the SAGE project has defined an overall methodology for developing guideline knowledge bases and an architecture for using knowledge base in the context of providing decision support through a clinical information system at the point of care.

2.1 Methodology of Guideline Knowledge Base Development

The SAGE guideline modeling methodology is a seven-step deployment-driven process[1] depicted in Figure 1. The formalization of guidelines in terms of a computable guideline model must be informed by the usage scenarios of guidelines in the care process. The usage scenarios identify opportunities for providing decision support, the roles and information needs of care providers, and the relevant guideline knowledge. Thus, once the decision to implement a guideline has been made, the SAGE guideline knowledge base development methodology requires that clinicians first create clinical scenarios that are detailed enough to support integration of guideline recommendations into real clinical workflow. When the guideline-based decision-support system is to be deployed in specific institutions, the care settings of these institutions need to be studied so that the scenarios that drive the modeling process match those in the actual settings. In the second step of the methodology, clinicians analyze the information content of the desired guideline recommendations and distill, from guideline texts, medical literature, and their clinical expertise, the knowledge and logic needed to generate these recommendations. This distillation process requires clinicians to select, interpret, augment, and

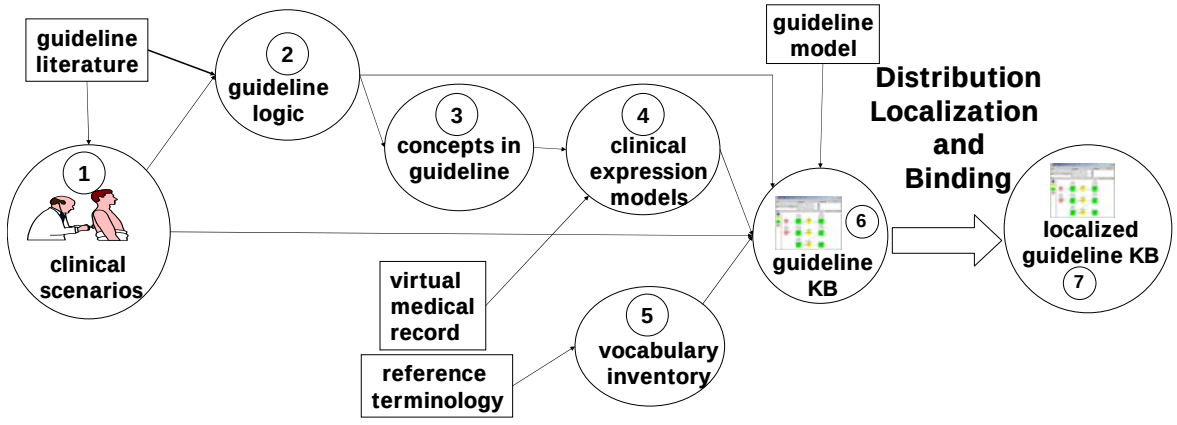


Figure 1 Methodology for formalizing a clinical practice guideline

[1] See Tu, S. W., Musen, M. A., et al. (2004). Modeling Guidelines for Integration into Clinical Workflow. Medinfo, San Francisco, CA, USA, 174-178. for a more detailed exposition of the guideline modeling methodology. In this document, we added a distribution, localization, and binding step that was out of scope for the Medinfo paper.

operationalize guideline statements in terms of unambiguous concepts and of data that may be available (third step). Thus, for example, in the development of an immunization guideline knowledge base, terms like "hepatitis B vaccine" and "anaphylactic reaction" are extracted and recorded. In the fourth step, concepts identified as part of the required guideline logic are instantiated as detailed data models (called clinical expression models) that correspond to constraints on classes of the virtual medical record" (Johnson et al., 2001b). A clinical expression model for a guideline concept spells out precisely how patient data corresponding to that concept would be represented as instances of a VMR class. For example, in the SAGE VMR, we model allergy information as instances of an 'AdverseReaction' class that has attributes such as 'code,' 'substance,' 'reaction,' and 'effective time' (time during which a patient is presumed to be allergic to the allergen). A detailed model for "Anaphylactic reaction to hepatitis B vaccine" would say that such data will be modeled as instances of AdverseReaction class where the reaction slot is constrained to be a concept subsumed by "anaphylactic reaction" and the 'substance' slot is constrained to be a kind of hepatitis B vaccine. The fifth step of the methodology calls for specifying guideline concepts in terms of standard terminologies. As we will discuss later, the use of standard terminologies requires significant extensions and must be defined in the context of the detailed data model. The sixth step in the guideline knowledge development process is the translation of the clinical scenarios and guideline logic into a computer-interpretable model of guidelines. The assumption is that clinicians will be working with analysts or knowledge engineers who have modeling expertise in formalizing the guideline knowledge through a Guideline Workbench (Shankar et al., 2002; Shankar et al., 2003). Finally, before a formalized guideline can be installed and used in a local institution, its medical content must be reviewed and revised (in what we call the *localization* process) and its data models, terminologies, organization assumptions (roles, events, and resources) must be mapped to those of the local institution (in what we cal the *binding* process).

2.2 Architecture of SAGE Guideline Decision-Support System

The SAGE Guideline Model exists in the context of the SAGE Guideline Decision-Support System (GDSS), whose center piece is the SAGE Execution Engine. The SAGE Execution Engine provides decision-support for guideline-based care by loading in the guideline knowledge

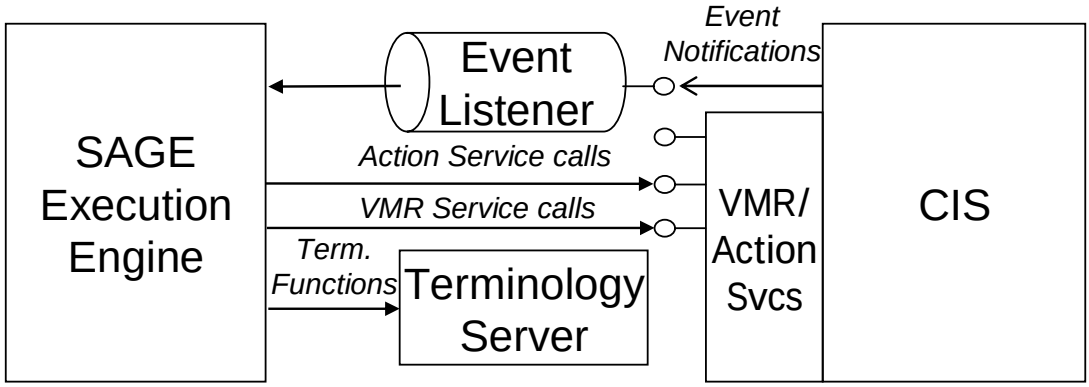


Figure 2 SAGE Guideline Decision-Support System Architecture

base and by interacting with the clinical information system (CIS) via an event listener and a set of services. Figure 2 illustrates the architecture of the system.

The event listener is the mechanism by which the SAGE Execution Engine is notified of state changes in the CIS. State changes in the CIS may involve workflow events, such as a patient checking into a particular clinic, or entry of data into the electronic health record. The listener is implemented as a web service (W3C, 2002) allowing for broad interoperability and can be used by any conforming CIS to publish events. Depending on the CIS capabilities, a terminology server may be present in a SAGE GDSS. The terminology server encapsulates standard terminologies and implements terminology subsumption and conversions that may be used by the engine. The VMR/Action services are interfaces into both patient data and application functionality provided by the CIS. The VMR services are used to get information from the CIS and the Action services are used to initiate actions within the CIS. The VMR/Action services can be viewed as wrappers around existing CIS data and functionality and support interoperability by presenting a unified view of clinical information systems to the guideline execution engine. The intention of the SAGE project is to align the VMR/Action services interfaces with standards such as HL7 messages and to propose these interfaces as standard access/action mechanisms into CIS entities external to the CIS such as the guideline execution engine. It is also the intention of the SAGE project to use the same execution engine and event listener across any CIS that conforms to the specifications of the VMR/Action services.

3 Design Goals

The literature on guideline models is full of alternative methods for formalizing clinical guidelines and protocols. What is the justification for starting yet another guideline model? We agree with the requirements articulated in Boxwala et al. (Boxwala et al., 2001) (although, instead of pursuing multiple uses of guideline knowledge bases, the project is scoped to concentrate on the use of guidelines in decision support in patient-care settings). Thus the project shares many of the goals of the projects such as the ones reviewed and compared in earlier studies (Peleg et al., 2003).

Three considerations led us to our decision to start a new model. First, with the emergence of clinical standards such as Health Level Seven's Version 3 (HL7 v3) Reference Information Model (RIM) (Health Level 7, 2003) and College of American Pathologists's SNOMED Clinical Terms (Wang et al., 2002a), we believe that we have the opportunity to build a guideline model from ground up to take advantage of these standards in a systematic way. Thus, for example, instead of defining our own data types and providing mechanism for user-defined patient data model as GLIF3 (Peleg et al., 2000) does , we incorporate the HL7 v3 data types directly in into our model and we work within HL7 to help to define the Virtual Medical Record (VMR) as the standard view of patient data assumed in guideline modeling. However, as we will discuss in the paper, make use of standards for modeling guideline is not a straight forward process. Rarely do existing standards completely satisfy the requirements of guideline modeling. For example, we need to develop mechanisms for defining new concepts based on terms in standard terminologies. Similarly, we need to extend a standard process model to account for decision-making activities in guidelines. Thus the elucidation of the complex relationship between existing standards and requirements of guideline modeling and deployment is one of the themes of the SAGE project.

The second consideration is SAGE's approach to integrating guideline-based decision support with the workflow of care process. That the success of clinical decision-support systems (DSSs) depends heavily on how the system is integrated into the workflow of the care process is widely recognized (Fieschi et al., 2003). Interpretation of the integration problem, however, varies widely. For alert-and-reminder systems, integrating into the workflow can mean the timing, modality, and format of notification (Krall et al., 2002). For implementations of chronic-disease guidelines in the primary care setting, workflow considerations may be used as factors in the design of the user interface and decision-support services that a system provides (Shiffman, 1994). In hospital environments, the protocol for managing a specific medical condition may drive the workflow that sequences care tasks and schedules resources (Quaglini et al., 2000). The SAGE project takes the approach that, as a provider of decision-support services to clinical information systems, SAGE will not be in control of host systems' workflow management. Thus, in SAGE's modeling approach, we are not required to model detailed workflow as, for example, University of Pavia's careflow methodology proposes. Instead, the SAGE system will respond to *opportunities for decision support* in the care process. We need to model enough of the workflow contexts to recognize appropriate events that should trigger decision-support services. Upon receipt of such triggering event, the SAGE decision-support system will deliver, through existing functions of the clinical information system, guideline-based recommendations appropriate for members of a care team. Thus, for example, physicians might see "inbox" notifications of guideline-based recommendations, while nurses might be presented with an active care-reminder flowsheet that is populated with a pre-authorized order set. The implication of this approach for the guideline modeling is that guideline knowledge will be embedded in an event-driven reactive system that takes into account clinical and organization contexts such as care setting and provider roles. Instead of just creating an electronic version of a clinical practice guideline, guideline modeling in SAGE formalizes guideline knowledge being used in specific scenarios and settings.

The third consideration in our decision to start a new guideline line is that, in recent years, much interchange and cross-fertilization have taken place in the guideline modeling community. Starting with workshops such as the ones sponsored by InterMed in 1999, Open Clinical in 2000, and University of Leipzig in 2001, and continuing with a number of comparison papers (Wang et al., 2002b; Peleg et al., 2003), workers in the guideline modeling community have gained much better understanding of the commonalities and differences among different guideline modeling approaches and of the design choices made in them. The SAGE project has given us the opportunity to synthesize prior work and, wherever possible, to establish mappings between the SAGE model and other models. Thus, the work on SAGE guideline model builds directly on prior models such as GLIF (Boxwala et al., 2004; Peleg et al., 2004), EON (Tu et al., 1999), PROforma (Fox et al., 2000), GUIDE (Quaglini et al., 2000), and PRODIGY (Johnson et al., 2000).

In summary, the SAGE project seeks to create a guideline model that

- is sufficient to encode guideline knowledge needed to provide situation-specific decision support
- uses standardized components that allow interoperability of guideline execution elements with the standard services provided within vendor clinical information systems.

- includes organizational knowledge to capture workflow information and resources needed to provide decision-support in enterprise setting
- synthesizes prior guideline modeling work for encoding guideline knowledge needed to provide situation-specific decision support and to maintain linked explanatory resource information for the end-user

4 Components of Guideline Model

The Institute of Medicine report on clinical practice guidelines defines guidelines as "…systematically developed statements to assist practitioner and patient decisions about appropriate health care for specific clinical circumstances" (Field et al., 1990). We conceptualize these systematically developed statements as a set of *recommendations* consisting of some usage *context* (specific clinical circumstances) where some course of *actions* are preferred over others (*decisions* about appropriate health care) in order to reach the best possible decisions. These recommendations are defined for some target population as defined by a set of *enrollment criteria*. These enrollment criteria, as well as decision criteria that we will discuss later, have to be expressed in a formal *expression language* that make use of *medical concepts* and may query *evidence statements* or *external knowledge sources*. Guideline actions are operationalized in terms of a set of CIS actions (Figure 3).

A SAGE guideline knowledge base, by design, is developed with specific usage scenarios in mind. It cannot be seen as merely a computer-interpretable form of the original guideline. The construction of the guideline knowledge requires selection from and interpretation, synthesis, and often disambiguation of the source literature. The original guideline is only one of possible many source documents used in the construction process. The guideline knowledge base should be seen as a new "guideline" that requires the same process of review, validation, and testing as the any published guideline. Thus, the guideline knowledge base is annotated with *metadata* comparable to those that should be present for paper-based guidelines.

As we described in the overview of the SAGE guideline modeling process, a key part of the process is the formalization of concepts used in a guideline in terms of standard terminologies and information models. To the extent that these terminologies and models can be standardized and mapped to those of a particular clinical information system, a guideline knowledge base and guideline execution engine have the possibility of achieving semantic interoperability. The components of the SAGE guideline model and the necessary semantic layer of standard terminologies and information models are shown in Figure 3.

In the rest of the section, we describe in detail the components of the SAGE guideline model, their design rationales and their use in clinical decision support. We will illustrate the model components using examples from SAGE knowledge bases that were developed based on an ICSI immunization guideline (Institute for Clinical Systems Improvement, 2002) and on an American Diabetic Association guideline (American Diabetes Association, 2002).

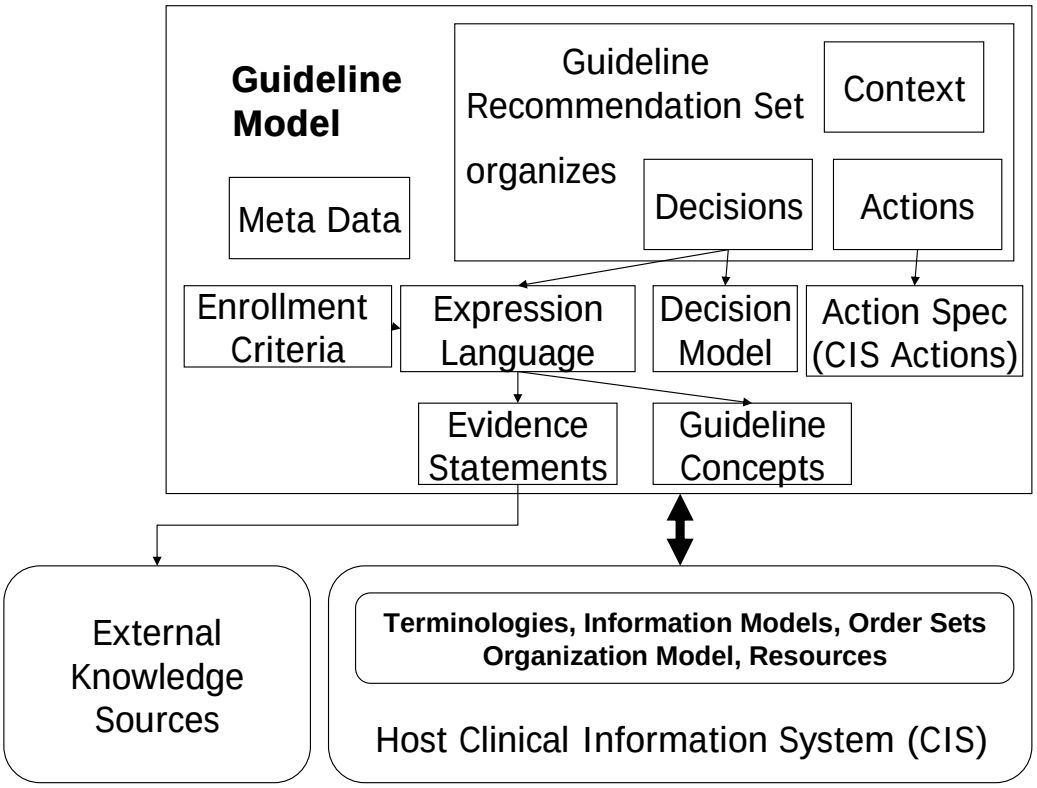


Figure 3 Top-level structure of SAGE Guideline Model and the supporting terminological and information models

4.1 Guideline Class

A guideline is an instance of **Guideline** class. In addition textual *label* and *description* slots, each guideline is linked to an instance of **Guideline_Metadata**, a set of enrollment criteria, a set of de-enrollment criteria, a set of configuration parameters, evidence statements, and a collection of recommendation sets. The evidence statements and recommendation sets of a guideline constitute the heart of a guideline are its collection of recommendations. They are described in Section 4.2 and Section 4.3 respectively.

4.1.1 Guideline_Metadata

The metadata component of the SAGE guideline model is not fully developed, as the metadata that annotate a SAGE guideline play no role in computing patient-specific recommendations. Currently the metadata component consists of selected high-level GEM (Shiffman et al., 2000) attributes such as identity, developer, method of development, and revision plan, and additional attributes, such as change log, that are useful for communication within the project (Figure 4). We consider the work to define guideline reporting requirements (Shiffman et al., 2003) and to annotate and index guidelines for search and retrieval (Shahar et al., 2003) highly complementary to the work the SAGE project. We expect to incorporate much of those results in specifying the metadata of a SAGE guideline

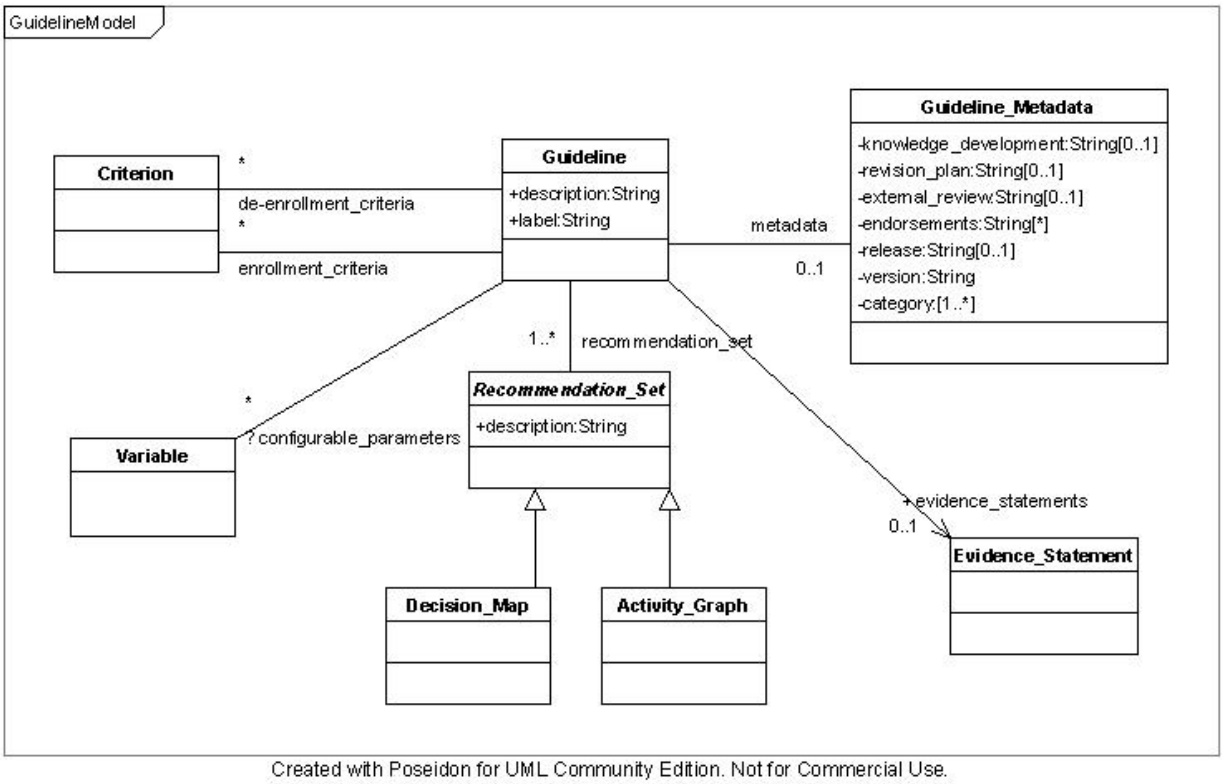


Figure 4 Top-level structure of the SAGE guideline model

4.1.2 Enrollment and De-enrollment Criteria

We expect that for a guideline to be applied in the management of a patient, the patient must be *enrolled* in that guideline. Enrollment can be automatic, as in the case of an immunization guideline, which is, by default, applied to all patients, or manual, which requires a clinician's (and possibly the patient's) explicit consent. In either case, separating out *enrollment_criteria* as a distinguished attribute allows applications to check a patient's eligibility for the guideline easily. All parts of enrollment criteria must be true before a patient is enrolled in the guideline. Like the inclusion and exclusion criteria of clinical trials, enrollment criteria are entry conditions that don't have to hold after a patient is enrolled.

Similarly, a distinguished *de-enrollment_criteria* attribute allows the SAGE execution engine to detect enrolled patients who should be taken off a guideline. Conditions that suggest that a guideline is no longer appropriate for an enrolled patient should be encoded as de-enrollment criteria. If any de-enrollment criterion is true, de-enrollment should be considered.

Implementation Note: Currently SAGE GEE enrolls patients automatically. Some events are considered *enrollable event* (e.g, new problem event sent by Carecast). The event triggers SAGE engine to evaluate enrollment criteria of guidelines for which the patient is not rolled, and enroll the patient for whom enrollment criteria evaluate to true. SAGE GEE then sends internally generated Enrollment event, which may trigger an Activity Graph recommendation set.

The types and structure of **Criterion** classes are described in Section 4.7.9.

4.1.3 Configurable parameters

A guideline may allow values like thresholds for high values of laboratory test results to be configurable. Such configurable values are modeled as instances of the **Variable** class (See Section 4.7.3). The *configurable_parameters* slot holds a collection of such variables for easy inspections and modification.

4.2 Evidence Statements

The purpose of the evidence statement construct is to provide a general method to represent guideline statements regarding the relationships between clinical conditions and interventions discussed in a guideline. The representation allows annotation of relation type (e.g., reason for), relation qualifying (e.g., compelling), links to evidence sources, strength of evidence, and subject of statement (e.g., prophylaxis of X, treatment of Y).

An *Evidence_Statement* class allows a statement like "for the treatment of hypertension, presence of diabetes is a compelling indication for the use of ACE inhibitor (Strength of evidence ICSI grade 1 conclusion, see references "great big clinical trial in the sky" and "jnc 7 guideline"). The attributes of the **Evidence_Statement** class are as follow:

- *label*: a descriptive string
- *statement_subject*: a terminology concept code representing a grouping of evidence statement used in a particular context (e.g., "treatment of hypertension")
- *relation_type*: a terminology concept code representing a relationship between a condition and an intervention (e.g., indication) Implementation Note: SAGE Execution Engine does exact match of relation_type (no subsumption)
- *relation_qualifier*: a terminology concept code representing a relationship
- *from*: a criterion that defines the patient condition that is relevant to the relation to the intervention
- *to*: a terminology concept code representing the intervention being considered.
- *strength_of_evidence*: a terminology concept code representing the strength of evidence for the relation between condition and intervention
- *references*: A collection of **Supplemental_Material**.

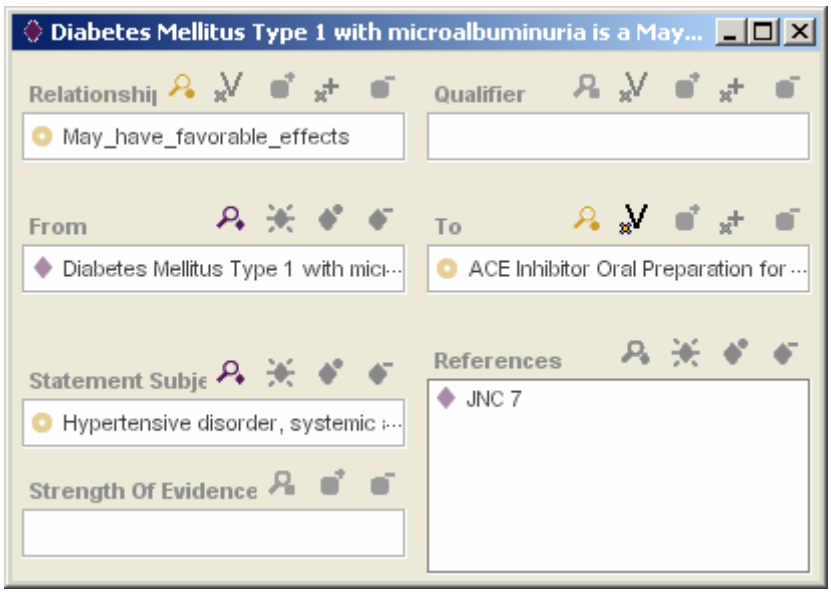


Figure 5 An instance of **Evidence_Statement**, encoding the statement "In the context of treatment of hypertension, the use of ACE inhibitor may have favorable effect on diabetes." JNC7 guideline is the reference for this statement.

4.3 Recommendation_Set

Recommendation statements represent the fundamental assertions of guidelines. In the SAGE guideline model we propose the concept of a *recommendation set* defined as a collection of related recommendations that are applicable in one or more shared contexts and that are organized according to a computable formalism. A context is defined by a combination of a clinical setting (e.g. outpatient encounter in a family medicine clinic), triggering events that make the context active (e.g. patient checking into a clinic), the care provider to whom the recommendation is directed, and the relevant patient states (e.g. a hypertensive patient who has been prescribed anti-hypertensive agents). Within each context, a recommendation may describe the preferred choice in a management decision (e.g. whether to administer the second dose of hepatitis B vaccine to a child), or it may recommend a series of actions be carried out (e.g. perform history and physical before ordering certain tests). A recommendation set specifies how decisions and actions are related to each other in a specific context. A single guideline may encompass multiple recommendation sets. For example, an immunization guideline may have one recommendation set for a new-born baby in a neonatal clinic, a recommendation set for pediatric patient in an outpatient clinic, and a third for an adult patient in an outpatient clinic.

We propose that recommendation sets consisting of either **Activity_Graphs** that represent guideline-directed processes or **Decision_Maps** that represent recommendations involving decisions at a time point.

4.3.1 Activity_Graph

We first describe the **Activity_Graph** as a type of process model for formalizing the description of inter-related activities. Then we describe the usage of **Activity_Graph** in encoding SAGE guidelines. We describe each component of an **Activity_Graph** and finish with an example.

4.3.1.1 Process Model

We propose an **Activity_Graph** formalism to describe relationships among different activities in terms of a process model. Several process models have been proposed as standards in the literature. We adopted the Workflow Management Coalition's Workflow Process Definition Language (WPDL) (Workflow Management Coalition, 1999) as the basis for the semantics of an Activity Graph. We do so because WPDL's object-oriented feature, the close correspondence between WPDL and the process specification of existing guideline models, prior experience with WPDL in the guideline-modeling community (Quaglini et al., 2000), and the available literature and software for workflow management systems. In WPDL, a process consists of a collection of activities each of which is a unit of work that is carried out by a combination of organizational resources and/or computer applications. Activities are related through precedence relations as defined by transition and transition conditions, and by hierarchical decomposition relations as defined by subflow relationships. Sets of activities may be carried out sequentially, concurrently, or in any order, as indicated by *split* and *join* constraints. Dummy activities that have no associated work (called Route activities in WPDL) can be used solely for specifying join and split constraints among other activities. Other information, such as priority and automation mode (i.e., whether the activity can be started or stopped automatically) can be attributes of an activity. The model also allows extensions through addition of attributes that are necessary for particular applications.

A generic workflow process defined in terms of activities and transitions among them, at first glance, does not make distinctions, such as contexts and decisions, which we've argued, are important aspects of a computable guideline. Thus, it is necessary to extend the standard process model. Decisions we can model as a specialized activity, where the result of decision making is commitment to selected courses of actions, and the decision-making application. A context characterized by a combination of clinical setting, care provider, patient condition and therapies is more of a description of the state of the world than an activity. Nevertheless we will incorporate the notion of context in the process model as a specialization of the WPDL Route class.

4.3.1.2 Usage of Activity_Graph

In the SAGE guideline modeling approach, we organize guideline recommendations by their scenarios of usage. A scenario is an abstraction of a care process that is implemented for specific roles, entities and actions within an enterprise. Thus, out-patient encounters, telephone triages, automated screening of patients for diagnostic tests, and in-patient administration of chemotherapies are all examples of scenarios. Each scenario involves interactions (i.e. exchange of information) among the SAGE decision-support system, components of the clinical information system, and possibly (through the CIS) human users.

A scenario is characterized by one or more **Context** that are defined by a combination of (1) clinical setting (e.g. out-patient internal medicine clinic), (2) patient state (e.g. patient having diagnosis of hypertension), (3) patient management state (e.g. patient receiving anti-hypertensive medications), (4) roles played by clinicians involved the care process, and (5) events that trigger the context. Alerts and reminders, recommendations for data collection and medical decision making are encoded as dependent on these contexts. In the SAGE guideline model, these context-dependent guideline logic are modeled as parts of "recommendation sets" (see below).

A guideline session is a run-time sequence of interactions between SAGE and a component of the clinical information system (and through it, external users) that is playing some role. Thus, for example, during an out-patient encounter, there may be simultaneous sessions between SAGE and a nurse and a physician. A session is always started by some triggering event. SAGE executes the logic as represented in an encoded guideline, interacting with the clinical information system (and, through it, possible external users) until exhausts all possible paths in the top-level activity graph. More precisely, a triggering event that activates a Context node starts a session that has an associated branch count. The branch count is incremented each time an execution thread transitions out of a node and decremented when a transition is made into a node. A session ends when there is no possible transition out of node and the branch count is zero. A transition out of a node if not possible if

1. SAGE execution of the guideline logic reaches a terminal point, with no additional recommendation to evaluate

2. SAGE execution reaches an Action node that has previously being executed.

3. SAGE execution of the guideline logic reaches a point that requires waiting for additional events. The session is considered suspended until the event being waited on triggers the re-activation of the session. (Thus, a trigger may activate a guideline action that represents the continuation of guideline execution in a particular context. Hence a context is not always the beginning of a session.)

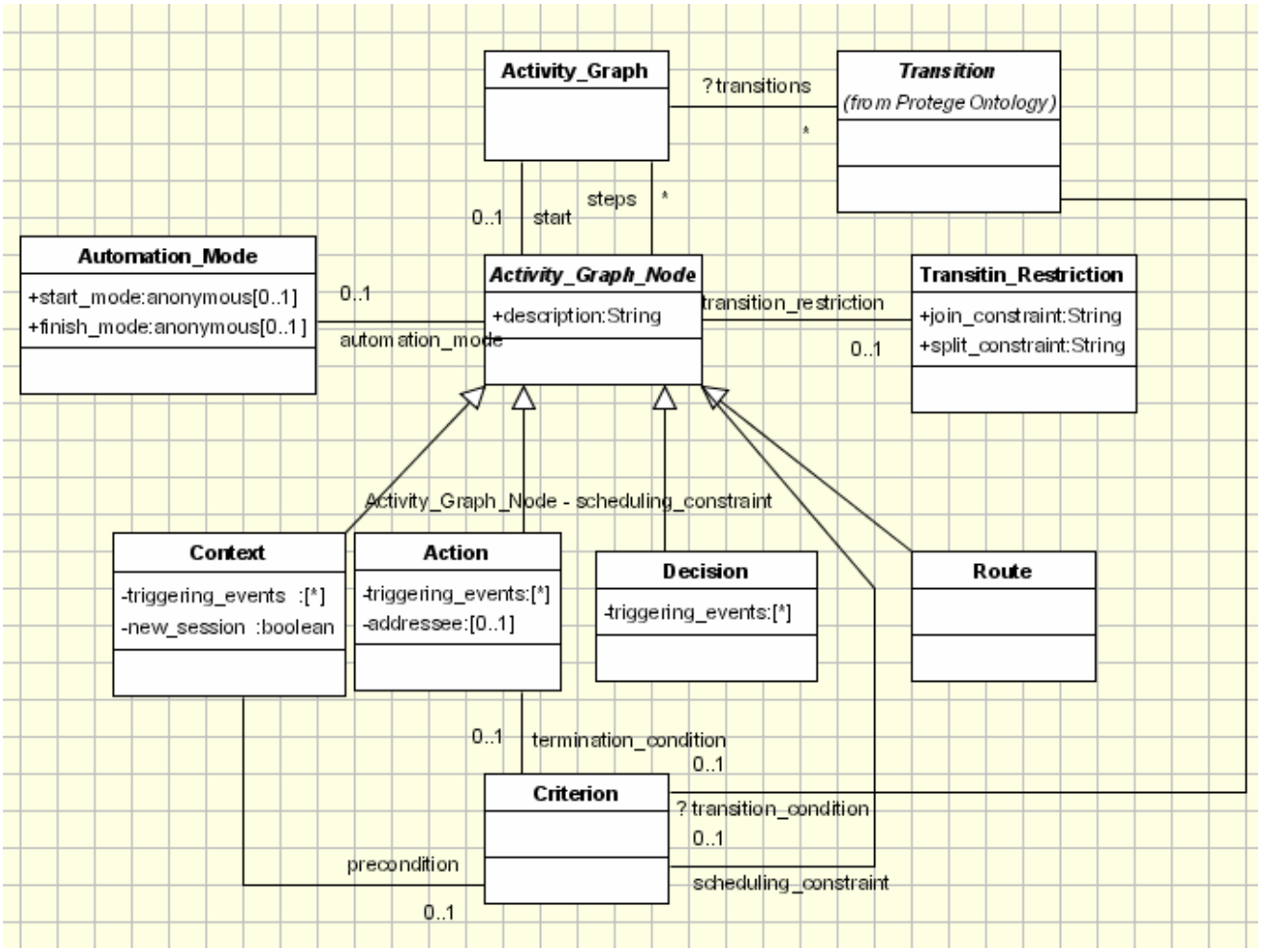


Figure 6 The structure of the Activity_Graph class. Only selected slots are displayed.

4.3.1.3 Activity_Graph_Node

An Activity Graph is a specialization of a WPDL process consisting of **Action**, **Decision**, **Context**, and **Route** nodes (Figure 6). All Activity Graph nodes have the following properties:

automation mode: specifies whether this activity can be automatically started or stopped by software. The manual alternative is for a user to interactively indicate whether this activity can be started or stopped. The value is an instance of **Automation_Mode** class that has *start_mode* and *finish_mode* slots.

transition restriction: specifies control information in relation to activities preceding and following a particular activity. It is an instance of **Transition_Restriction** class that has two slots:

> *split_constraint* (AND or XOR): if the value is AND, then all subsequent activities need to be evaluated for possible activation. If the value is XOR, then the outgoing transitions are evaluated in order, and the first transition that evaluates to true is taken.

join constraint (AND or XOR): if the value is AND, the all active preceding activities must be completed and the transition conditions evaluate to true before the current activity is enabled. If the value is XOR, then the current activity is enabled as soon as the first preceding activity completes and the associated transition condition evaluates to true. The active threads that may reach this point subsequently will not enable the node multiple times.[2]

Thus the AND split constraint allows concurrent activities to be started following the current node, and the join constraint synchronizes multiple threads of execution.

Won't execute same node again during a invocation of event (visited)

scheduling constraint: The scheduling constraint is a Boolean criterion that specifies temporal conditions that must be satisfied before an enabled activity is carried out. A node is *enabled* if it is triggered or if its join constraint is satisfied. The enabled node is *activated* either manually by a user or automatically if its automation mode is AUTOMATIC and the scheduling constraint is satisfied.

A (non-context) activated activity can be *completed* or *aborted*. If an activity is completed, then the split constraint determines which following transitions are evaluated for next steps to take.

The **Context**, **Action**, and **Decision** classes further share a set of properties:

triggering_events: a set of **Event** instances (described in Section 5.1.1) each of which can either initiates processing of a scenario as defined by a **Context** node, or resume processing of a **Decision** or **Action** node that has been suspended while waiting for the events in the *triggering_events* set.

subguideline: an instance of the **Subguideline** class that is described in Section 4.4. The intention of allowing subguidelines in each of **Context**, **Decision**, and **Action** classes is to allow hiding of details when appropriate. A subguideline of a **Context** node should recommend actions that are relevant in that context, regardless of any subsequent decisions or actions[3]. A sub-guideline in a **Decision** node should recommend actions that are helpful in making the decision (e.g. obtaining relevant information about patient state). Finally, a sub-recommendation in an Action node should refine the actions specified in the **Action** node.

Implementation Note: SAGE GEE does not make use of subguidelines in Decision or Context nodes.

---

[2] The semantics of the XOR join correspond to what has been called "Discriminator" join, where we allow the first incoming branch to trigger the activity following the discriminator, and to keep track of the other branches. Once all branches have completed, the discriminator is "reset" and the next incoming branch to finish can again trigger it. See Kiepuszewski, B., ter Hofstede, A. H. M., et al. (2003). Fundamentals of Control Flow in Workflows. *Acta Informatica* **39**(3): 143-209. for a detailed discussion of the semantics of control flow in workflows.

[3] The use of subguideline in a Context node corresponds to PRODIGY and EON models' **Consultation_Guideline** construct. Its primary use was to hide process-oriented actions in a Decision Map. In the SAGE Activity Graph, the same semantics can be obtained by having concurrent branching (AND split) of **Actions**.

references: a set of **Supplemental_Material** instances. **Supplemental_Material** are materials that support the recommendations made in a guideline. In addition to a textual *label*, they have the following properties:

role: The role of this supplemental documentation in the recommendations of this guideline. It can be one of `evidence`, `comment`, `patient_education_material`, `illustration`, `source`, or `consent_form`.

keywords: textual keywords related to this **Supplemental_Material** instance.

URLs: The universal resource locations for this **Supplemental_Material** instance.

metadata: The bibliographic information about this **Supplemental_Material** instance.

abstract: A textual summary of this **Supplemental_Material** instance

4.3.1.3.1 Context

Context is a basic element of the SAGE guideline model. **Context** nodes specify and declare the assumptions made about the health care enterprise work model that are otherwise implicit in every instance of a guideline implementation. Using this approach, SAGE does not control the enterprise work model, but does expect to enumerate the workflow parameters and function within it. In the example of Figure 9, the first **Context** specifies the context of a patient checking into an outpatient primary-care clinic, while the second **Context** is triggered by a primary care nurse accessing the patient record.

In terms of the workflow process model, a **Context** node is a specialization of a WPDL Route activity that has been extended to represent the clinical and organizational situation in which one or more guideline recommendations are to be applied. It is defined by a combination of (1) clinical setting (e.g. out-patient internal medicine clinic), (2) patient state (e.g. patient having diagnosis of hypertension), (3) patient management state (e.g. patient receiving anti-hypertensive medications), (4) roles played by clinicians involved the care process, and (5) events that trigger the context.

Context nodes are complex objects. Their defining attributes specify their trigger events, clinical setting and patient state. Additional attributes specify the enterprise information-processing needs and resources employed during the session. These enterprise resources are modeled within a health-care organizational ontology, which is meant to define all roles, events, resources and settings that are material participants in guideline recommendation sets (see Section 5.1).

In addition to properties inherited from **Activity_Graph_Node**, properties of a **Context** node include:

triggering_events: the set of events that can trigger this context for evaluation

precondition: the condition that should be true for this context to become active. It can be overridden by a clinician.

clinical_context: an instance of **Clinical_Context** that specifies the nature of the clinical interaction:

Clinical setting for the scenario

Clinical roles involved in the care process being modeled

Clinical resources (material, equipment, machines)

informatics_context: an instance of **Informatics_Context** that specifies the nature of the man-machine interactions within the scenario

Communication utilities employed in the scenario process

Knowledge-based utilities as required by the scenario

Implementation Note: Knowledge-based utility of *Population model alerting engine*: Get all patients who are enrolled in the guideilne and execute the context for each patient. Triggering event that initiate the context can be for any patient. (*Population model alerting engine*: is treated as another triggering event.)

The execution semantics of **Context** nodes is roughly as follow:

1. A CIS event matches the triggering event for one or more **Context** nodes in any SAGE activity graph in any SAGE guideline
2. If event carries a patient id, verify that the patient is enrolled in the guideline, then evaluate the precondition of the **Context** node (possibly separate parallel sessions for separate triggering event. (However, Inquiries won't be asked more than once, i.e., Pending states won't be executed more than once). The state of patient on a recommendation set is global regardless how many clinicians are accessing the guideline.

Implementation Note: SAGE Execution Engine assumes all events have patient id

3. If the event does not carry a patient id, then, for each enrolled patient, evaluate the precondition of the **Context** node
4. If the precondition of the **Context** node evaluates to true, then **Context** node is *enabled*

Implementation Note: The SAGE Execution Engine does not look at the scheduling_constraint property, so there is no *enabled* state in the current implementation.

5. The node is *activated* subject to the constraints of the *scheduling_constraint* property
6. Once activated, if the **Context** node has a subguideline, then subguideline is called. If the invocation of the subguideline is synchronous, then completion of the **Context** node waits for the completion of the subguideline; otherwise execution of the Context node is complete, and the flow of control follows that of the *transition_restriction* property.

4.3.1.3.2 Decision

Decision nodes support representation of decision knowledge required to recommend a choice among alternatives. The contract between a recommendation set and a Decision node is that, at the end of the decision-making process, commitment to one or more of the alternatives are made.

In terms of the workflow process, a Decision node is specialization of a generic WPDL activity. A **Decision** node is an activity that has been extended with a *decision model* attribute, through which decision knowledge for determining preferences among alternative courses of actions can be represented.

In addition to properties inherited from **Activity_Graph_Node**, **Decision** node has:

automation mode: specifies whether this activity can be automatically started or stopped by software. The manual alternative is for a user to interactively indicate whether this activity can be started or stopped. The value is an instance of **Automation_Mode** class that has *start_mode* and *finish_mode* slots.

addressee: the agent with whom the guideline is interacting in this activity.

decision_model: an instance of **Decision_Model** that encapsulates the decision-making knowledge and methodology to generate preferences among the alternatives

triggering_events: a set of events that may trigger the decision node, if it is suspended.

subguideline: an instance of **Subguideline** that allows guideline encoder to describe what need to be done to facilitate making a decision at this point.

The execution semantics of a Decision node is roughly as follows:

1. Once the **Decision** node is enabled and activated (see Section 4.3.1.3), if it has a subguideline, call that subguideline.
2. If the invocation of the subguideline is synchronous, then completion of the **Decision** node waits for the completion of the subguideline before evaluating the *decision_model*; otherwise it evaluates the *decision_model* immediately.
3. The result of evaluating the *decision_model* is a preference ordering of the *alternatives* following the Decision node. (Implementation Note: SAGE Execution Engine requires yes/no decision for each alternative)
4. If the *finish_mode* of *automation_mode* is "automatic," and if the *split_constraint* of *transition_restriction* is "`XOR`," then select the alternative with highest preference {that is at or above the recommendation_threshhold} and pass execution to that alternative. If there are more than one alternative with such preference, select one randomly. If there is no alternative that satisfies the recommendation_threshhold, then stop execution of this thread.
5. If the *finish_mode* of *automation_mode* is "`automatic`," and if the *split_constraint* of *transition_restriction* is "`AND`," then pass the execution control to all that satisfy the recommendation_threshhold.
6. If the *finish_mode* of *automation_mode* is "`manual`," then present the preference orderings of the alternatives to the *addressee* of the Decision node. (SAGE EE assumes automatic)

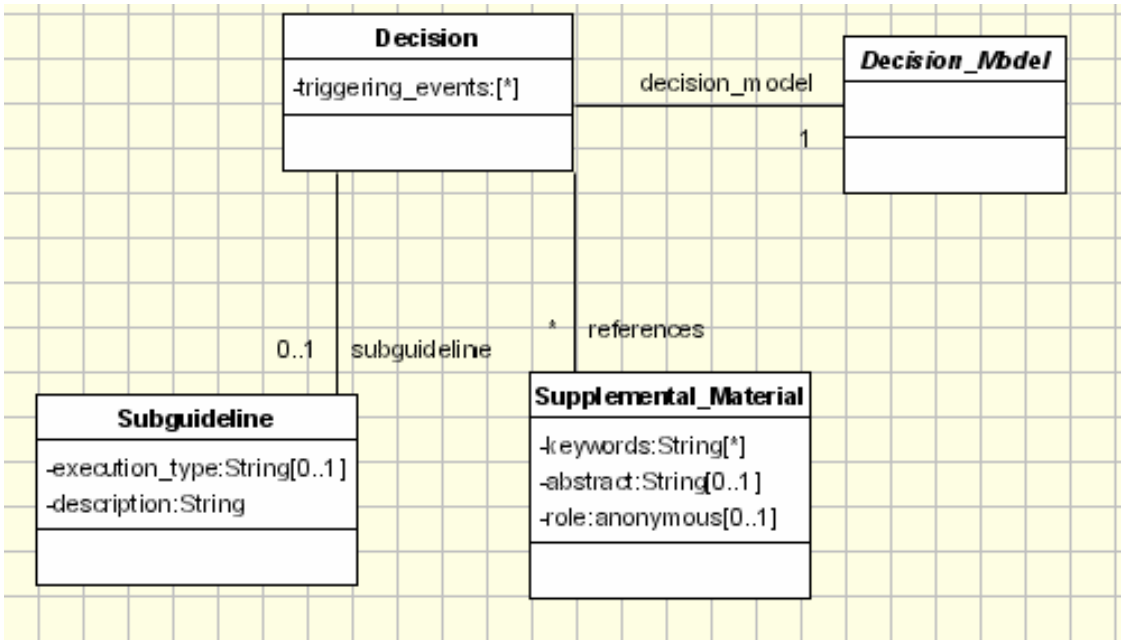


Figure 7 Structure of a Decision Node

4.3.1.3.3 Action

Action nodes model one or more information system activities employed in support of a recommendation set. In terms of the workflow process model, action nodes are specializations of generic WPDL activities. An Action node encapsulates a set of work items that should be performed either by a computer system or by a healthcare provider. The central part of an *Action* node is the set of **Action_Specification** specified in its *action_spec* property, or, in the case it has a subguideline, the *Subguideline* instance in the *subguideline* property. In order to support a variety of implementation options, action specifications may include support for messaging to system devices, including Inbox reminders and workstation interaction, goal specification, database retrieval and storage, and scheduling events. Complex actions may also be constructed as templates of retrieval/storage activities to support action features such as interactive questionnaires (see Section 4.6).

In addition to the properties inherited from the **Activity_Graph_Node**, an **Action** node has the following properties:

automation mode: specifies whether this activity can be automatically started or stopped by software. The manual alternative is for a user to interactively indicate whether this activity can be started or stopped. The value is an instance of **Automation_Mode** class that has *start_mode* and *finish_mode* slots.

triggering_events: a set of events that may trigger the **Action** node, if it is suspended.

addressee: the agent with whom the guideline is interacting in this **Action**.

termination_condition: for actions that may have duration during its execution (e.g. having subguidelines that are stateful), this action of this **Action** node is deemed completed if this criterion evaluates to true. The frequency of evaluating the termination_condition is left to implementation.

action_spec: specification a set of tasks that should be performed in the **Action** step. See Section 4.6 for descriptions of the **Action_Specification** classes.

repeat_expression: an instance of **Repeat_Specification** that describes the temporal, logical, and/or iterative conditions under which this action should be repeated. An instance of **Repeat_Specification** has the following properties.

number_of_cycles: an integer

repeat_every: an instance of temporal duration (e.g. 2 months)

for_how_long: an instance of temporal duration (e.g. a week)

The semantics of the repeat expression is that the action should be repeated at interval of *repeat_every* for *number_of_cycles* or for the duration of *for_how_long*.

- *subguideline* An instance of **Subguideline** that is executed as a part of this step.

The execution semantics of the **Action** node is as follows:

1. If the *repeat_expression* is null, then, once the node is activated (see Section 4.3.1.3), call the *subguideline,* if there is one; otherwise perform actions specified in the *action_spec* property. If there is both a subguideline and action_spec actions, the subguideline is executed first. If there is a subguideline, the execution mode of the subguideline (`synchronous` or `asynchronous`) determines whether we wait for the completion of the subguideline before performing the tasks specified in the *action_spec* property. (Implementation note: SAGE Execution Engine only executes subguidelines synchronously.)
2. If the *repeat_expression* is not null, then once the node is activated, perform the actions specified in *subguideline/action_spec* iteratively according to the **Repeat_Specification**.
3. If the *termination_condition* property is not empty, then, after performing the actions in *subguideline/action_spec*, checks whether the *termination_condition* is true before completing this activity and make transition to the subsequent activities. In case where this activity is being repeated, check the *termination_condition* after each repetition.

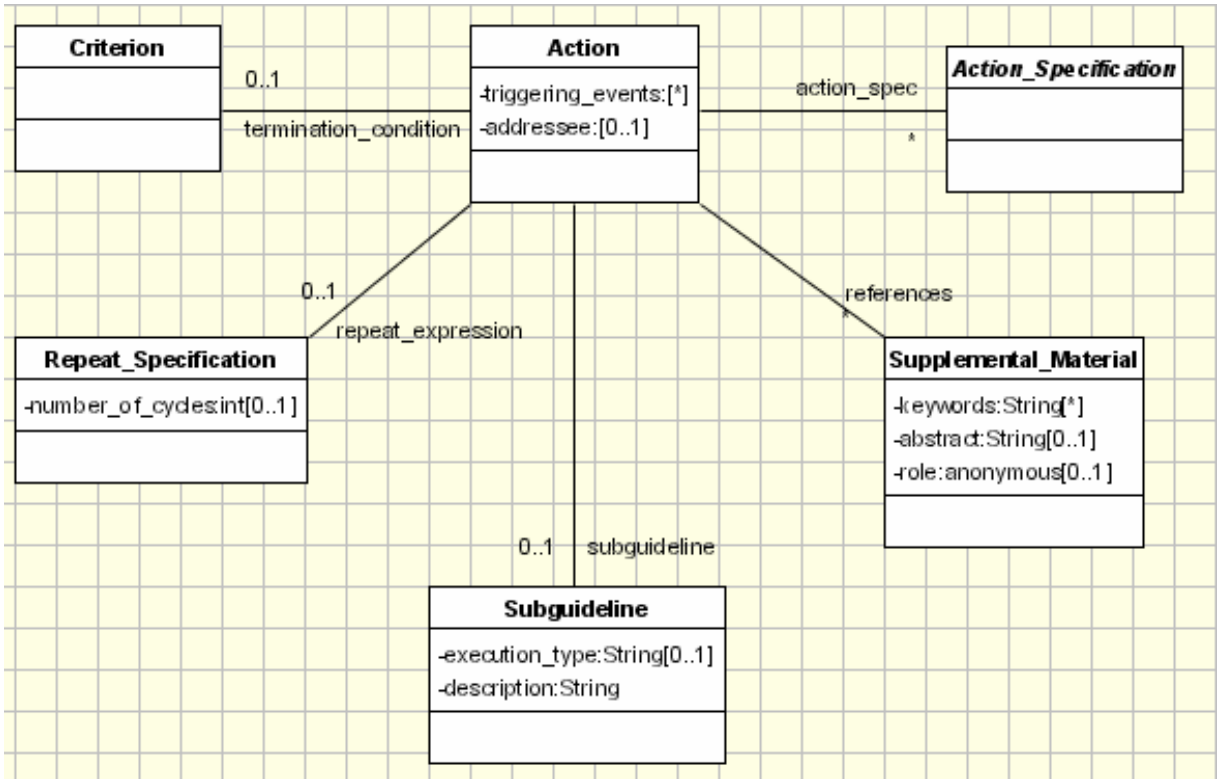


Figure 8 Structure of an Action Node

4.3.1.3.4 Route

A **Route** node is a "dummy" activity that can be used as a purely branching or synchronization node. The execution semantics of a **Route** node follows that described in Section 4.3.1.3.

4.3.1.4 Activity Graph Example

We use the Activity Graph formalism to model how SAGE DSS should respond to opportunities in the care process for providing appropriate decision support. Figure 9 shows a Protégé view of a top-level process specification in a SAGE immunization guideline. In Figure 9, the ovals represent instances of the **Context** class, rectangles instances of the **Action** class, and hexagons instances of **Decision** class. The context is triggered upon admission of a newborn to a neonatal clinic. The process specified in the **Activity_Graph** is not the workflow process in the clinic, but the computational process of the SAGE GDSS and how it coordinated with events in the clinical workflow. Thus, after the initial admission event activated this context, the SAGE GDSS checks whether hepatitis B vaccination is due and whether there are tests that need to be ordered both for the baby and for the mother.

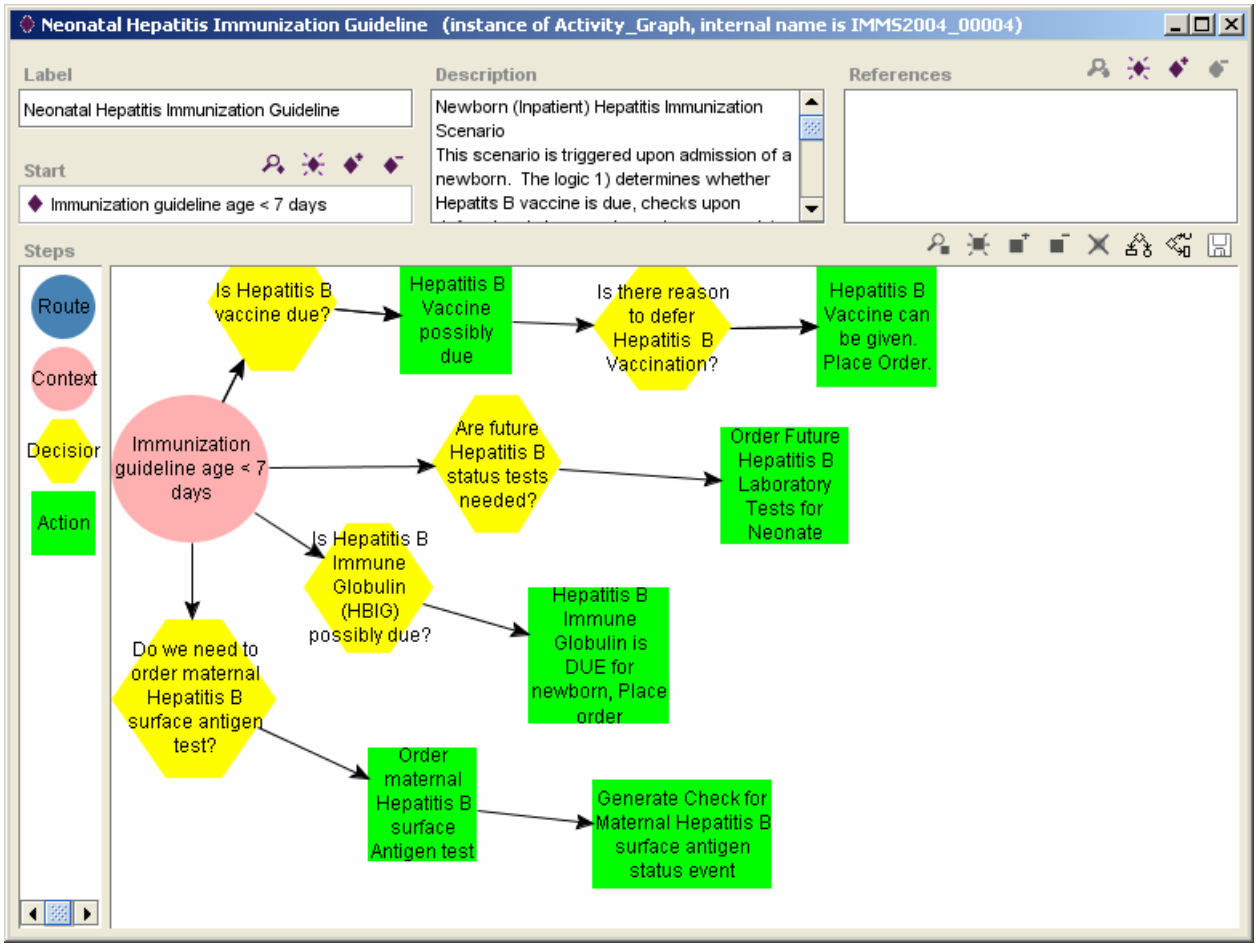


Figure 9 The Protégé view of a top-level process specification in the SAGE immunization guideline. It defines how a guideline DSS should react to the events in the care process. The ovals represent contexts, boxes actions, and hexagons decisions.

4.3.2 Decision Map

We argue that not all guideline recommendations are best represented as part of a process specification. The information for determining when a particular immunization is due, for example, involves declarative knowledge that is evaluated at a single point in time. The SAGE guideline model organizes such atemporal decision-making knowledge as a *Decision Map* consisting of a collection of decision points, each of which is defined by a context and a decision involving multiple alternatives (which may be additional decisions or actions). The context, decision, and action primitives in a Decision Map are similar to those in an Activity Graph, but, not being workflow activities, they have different execution semantics. Each decision point is separately triggered, if it is in a top-level recommendation set of a guideline, or requires separate evaluation to ascertain its applicability. Thus, one use of the Decision Map is the encoding of a collection of asynchronous alerts and reminders that are not organized as a connected process of activities. Alternatively, a Decision Map can be used as the decomposition of a high-level action that involves decisions made by a single provider in a specific clinical context at a single time.

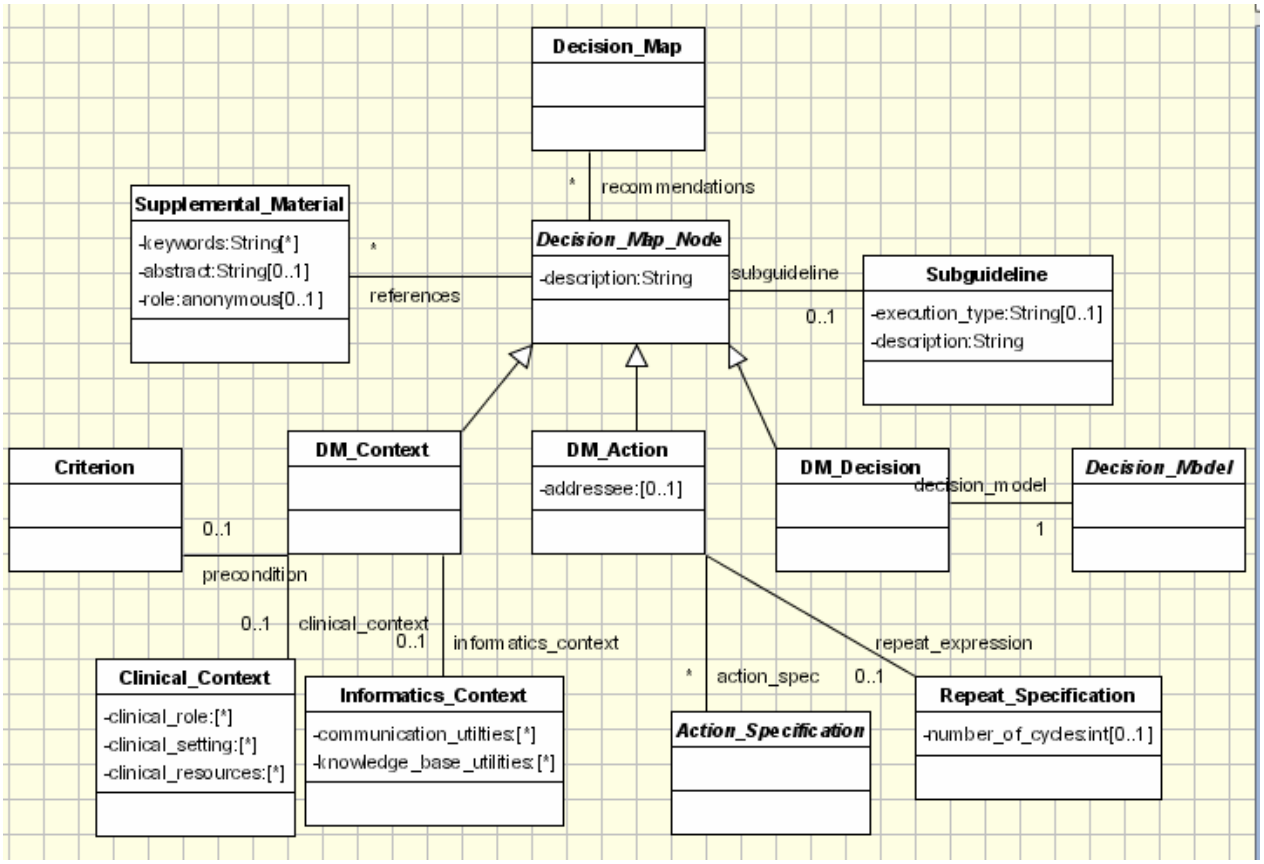


Figure 10 The Structure of a Decision Map

Implementation Note: In current SAGE Execution Engine, decision maps can only be subguidelines.

Computationally a Decision Map allows several possible specializations. A Decision Map can be a collection of event–condition–action rules where the Context nodes define triggering events, the Decision node the condition, and the Action node the actions. If triggering is defined externally, and if a Decision Map constraints that only one choice can be made in a decision, and an action always results in a new context, then computationally the Decision Map becomes an augmented transition net. Within this paradigm, each time the Decision Map is invoked, the guideline execution system determines the current context, applies the knowledge in the decision model, and commits to one alternative among the choices.

Implementation Note: In current SAGE GEE, decision maps cannot have triggers

4.3.2.1 Usage of Decision Map

When is a Decision Map the best choice for representing guideline recommendations?

1. When your guideline consists of a collection of decisions, each of which can be evaluated independently of each other at one point in time. This is the case when your recommendation set consists of decisions that are analogous to rules that have no

dependency on each other (although, in the case of the SAGE guideline model, the decision model can be more sophisticated than deterministic antecedents of a rule). Alternatively, decision maps are appropriate when recommendations depend on each other, but with any invocation of the recommendation set, only unrelated decisions are activated.

2. When the same decision knowledge in your guideline can be used in multiple contexts (e.g. mass screening versus primary care encounter). You can encapsulate the re-usable decision knowledge in a decision map, and model the workflow-dependent interactions as separate activity graphs that call the decision map.

4.3.2.2 Decision Map Node

The **Decision_Map_Node** classes that make up a **Decision_Map** are functionally similar to their **Activity_Graph** counterparts, except that there are no flow-of-control properties, such transition restrictions and schedule constraints, which govern the activation of the activities in a process[4]. The *automation_mode* constraint determines whether a **Decision_Map** node should wait for manual intervention before starting or finishing. In addition, a *split* constraint in a **DM_Decision** node specifies whether the choices at that decision point is XOR or AND.

Implementation Note: Since SAGE Execution Engine does not implement any of the workflow flow-of-control mechanisms, the only differences between Decision Map and Activity Graph are that, Decision Map may have multiple entry points (DM_Context instances) and that Decision Maps, as subguideliens, cannot have triggering events.

4.3.2.3 Decision Map Example

Figure 11 shows part of a decision map for determining whether certain immunizations are due for a patient. The pre-condition of each **DM_Contex** is evaluated to see whether the context applies. If it is, then the alternatives associated each **DM_Decision** node are evaluated according to the decision model that is specified in the **DM_Decision** node.

[4] Implementation Note: Currently, the **Decision_Map_Node** instances don't have triggering events. Consequently they cannot be used to model event–condition–action rules, as asserted in the description of the Decision Map formalism. Instead, in the current SAGE usage decision maps are always subguidelines that are invoked by other recommendation sets.

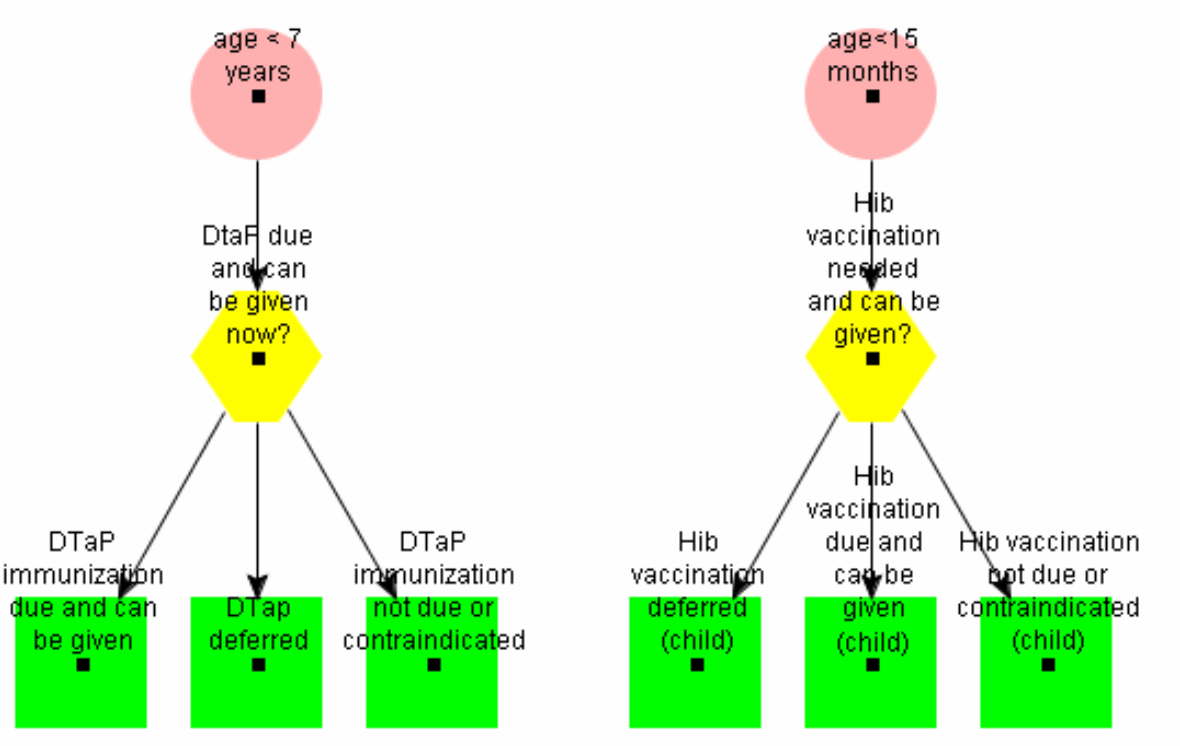


Figure 11 An Example of Decision Map

4.4 Subguideline

In order to manage complexity in guideline recommendations, we need to support hierarchical nesting of recommendations. Each **Context**, **Decision**, and **Action** node in an Activity Graph or a Decision Map may be associated, through an instance of **Subguideline**, with another recommendation set that we call sub-recommendation. The **Subguideline** instance specifies whether the sub-recommendation should be executed synchronously or asynchronously. A synchronous sub-recommendation means that the execution of the parent node is not completed until execution of the sub-recommendation is completed. Mandatory subtasks, for example, can be encoded as a synchronous sub-recommendation. Asynchronous sub-recommendation means that the parent node does not wait for the sub-recommendation to complete its execution. Instead, management of is forked off as an independent process.

A **Subguideline** instance also has associated references (i.e. instances of **Supplemental_Material**).

As a matter of guideline encoding convention, a sub-recommendation in a **Context** node should recommend actions that are relevant in that context, regardless of any subsequent decisions or actions. A sub-recommendation in a **Decision** node should recommend actions that are helpful in making the decision (e.g. obtaining relevant information about patient state). Finally, a sub-recommendation in an **Action** node should refine the actions specified in the Action node.

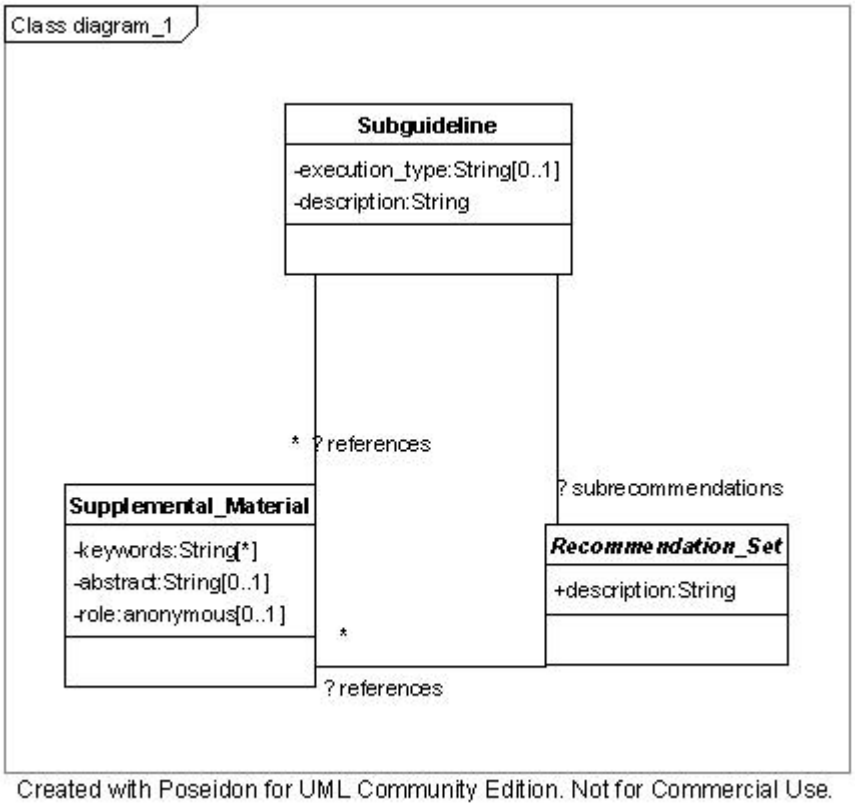


Figure 12 The Structure of a Subguideline

4.5 Decision Model

A decision model is a model for formulating preferences among a set of alternatives. In a Recommendation Set, a **Decision** node (in the case of **Activity_Graph**) or a **DM_Decision** node (in the case of **Decision_Map**), a reference is made to an instance of **Decision_Model** class. Alternative decision models can be substituted into a recommendation without changing the structure of the recommendation.

Current SAGE decision model is one that derives from PROforma, GLIF3, EON, and PRODIGY. It expresses preferences for an alternative in terms of a for-against argumentation model (Figure 13).

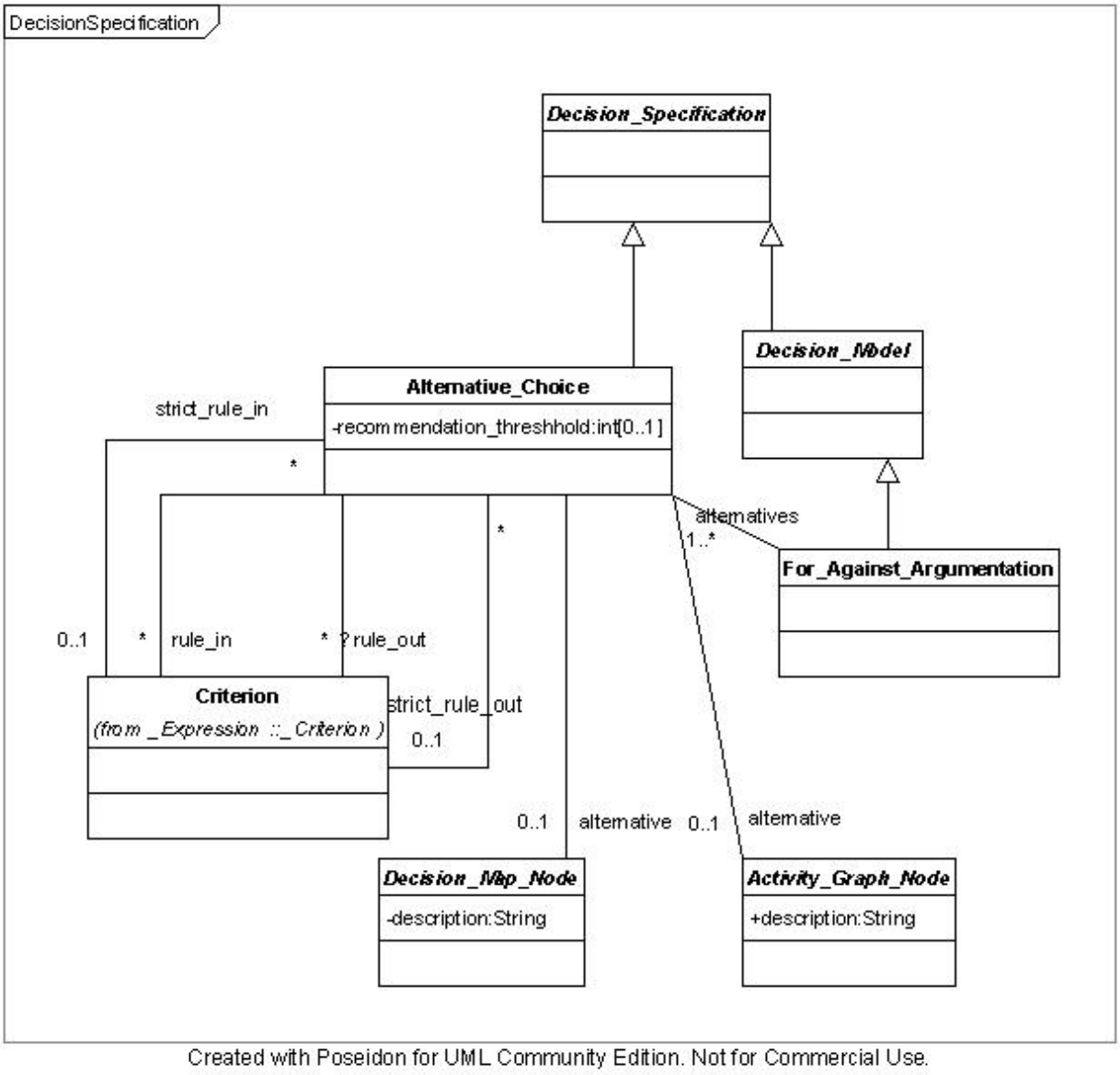


Figure 13 The Structure of a Decision Model

The **For_Against_Argumentation** decision model consists of a collection of **Alternative_Choices** instances, where each **Alternative_Choice** instance has the following slots:

alternative: an alternative at this particular decision point. The alternative should be an instance of the recommendation structure (**Activity_Graph** node or **Decision_Map** node).

strict_rule_in: A collection of criteria that express strong indications for the alternative.

strict_rule_out: A collection of criteria that expresses strong contraindications for the alternative. If any of the *strict_rule_out* criteria is true, then this alternative should not be considered.

rule_in: A collection of criteria that expresses relative indications for the alternative

rule_out: A collection of criteria that expresses relative contraindications for the alternative

recommendation_threshhold: The number of *strict_rule_in* criteria that, if true, makes this alternative a recommended choice

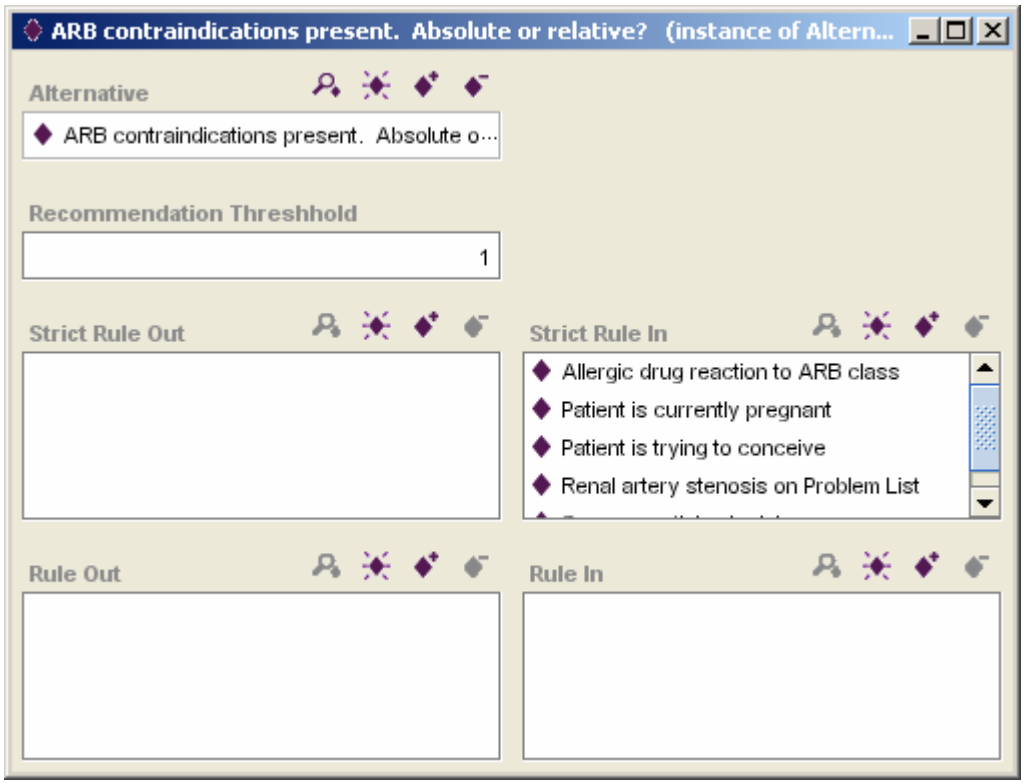


Figure 14 An Example of Alternative_Choice specification

4.6 Action Specification

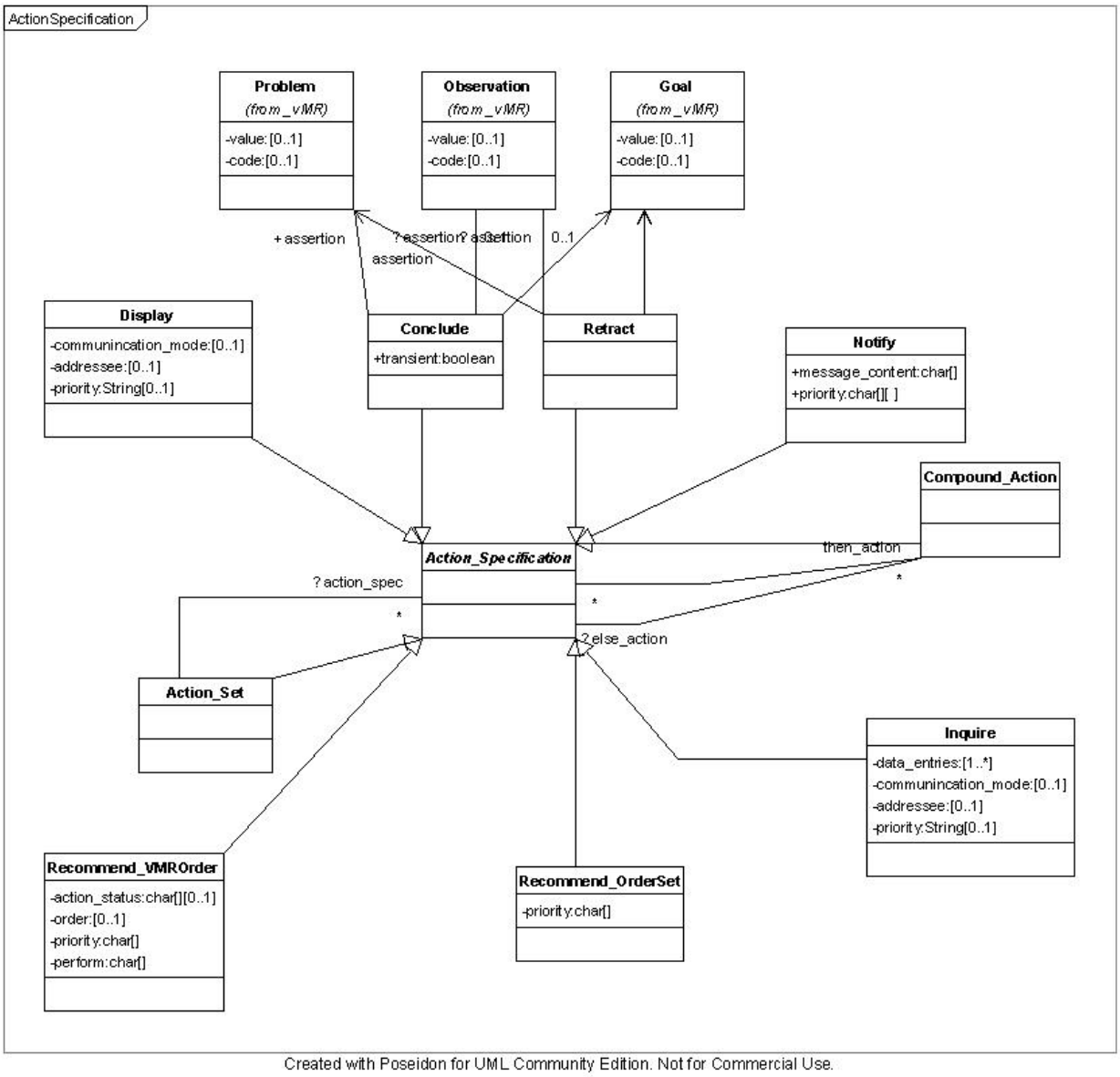


Figure 15 Action Specification Classes

Instances of **Action_Specifications** are abstraction of CIS actions that operationalize the guideline actions. They include:

1. A set of actions that assert to and retract from conclusions to the VMR
2. A set of actions that has effect on the CIS. This set includes communicate with some CIS agent (e.g. send message, display information, inquire about data) and acts such as placing an order, making a referral, setting goal, providing education, making an appointment.
3. An act to generate events in the future

4. Aggregations of actions

4.6.1 Actions that operate on VMR classes

The first type of operations involves making or retracting conclusions.

> **Conclude** writes inferred patient state and decision-historic data to the **Observation**, **Problem**, or **Goal** VMR classes. If transient flag is true, then data are not written to permanent record, but disappear after session ends.
>
> *Retract* retracts system-asserted **Observation**, **Goal**, or **Problem** instances.

4.6.2 External_Action

There are five types of external actions: **Notify**, **Inquire**, **Display, Recommend_VMROrder, and Recommend_OrderSet.** All external actions share three attributes: *label*, *condition*, and *priority*. The *condition* should evaluate to TRUE for the action to be executed. Currently, the *priority* slot is a string.

4.6.2.1 Notify

An instance of **Notify** passes a textual *message_content* on a *subject* with certain *priority* to an *addressee* using a *communication_mode*.

Implementation note: SAGE Execution Engine only send messages to 3 addressee (session owner, primary care provider, attending).

4.6.2.2 Inquire

An instance of **Inquire** requests a set of data elements from electronic medical record or from another agent. Like a **Notify** instance, an inquiry has *communication mode*, ***addressee***, and *priority* attributes. Instead of message content, an **Inquire** action has a set of *data_entries* that have the form of clinical expression models (Section 5.2.2). The *optional* attribute specifies whether the SAGE Execution Engine should wait for the results of the inquiry before resuming processing.

4.6.2.3 Display

The **Display** action has a *display_data* that allows the specification of display data in the forms of **Expressions** (e.g. **Variables**, **Functions**, **and VMR_Queries**) or **Clinical_DataSet_Entities** (e.g. flowsheet data elements) and **Supplemental_Material** (e.g., patient information leaflet).

Implementation Notes: Only Expressions are evaluated. If display items are instances of data model or Supplemental_Material, then only label field is displayed

4.6.2.4 Recommend_VMROrder

Different types of interventions allow the clinician user to structure, tailor and plan complex events following guideline recommendations. **Recommend_VMROrder** requires the specification of the following slots:

order: an instance of **VMROrder**, **Referral**, or **Appointment** VMR classes

perform: either `add`, or `delete`. The `add` slot value indicates that the specified VMROrder should be started, and the `delete` slot value indicates that the specified VMROrder should be discontinued.

action_status: `submit` or `authenticate`. `Submit` value indicates that the order can be submitted to the order entry system. `Authenticate` value indicates that the order requires further approval. (Implementation Note: If action_status is not specified, submit is the default)

4.6.2.5 Recommend_OrderSet

Sometimes the implementer of a decision-support system may want to deliver guideline recommendations in the form of order sets. The **Recommend_OrderSet** action specification has an *order_set* slot that references the appropriate order set. See Section 4.8 for detailed description of SAGE's support for order sets.

Implementation Note: As of Oct 4, 2006, the condition associated with **Recommend_OrderSet** is not evaluated.

4.6.3 Aggregations of Actions

We define two types of aggregation constructs.

4.6.3.1 Action_Set

An instance of **Action_Set** is collection of **Action_Specification** instances.

4.6.3.2 Compound_Action

Compound_Action is an if-then-else construct that allows selection of sets of **Action_Specifications** based on the result of evaluating a Boolean *condition*. If the condition evaluates to `unknown`, then neither the *then* nor the *else* clauses are executed. (Implementation Note: SAGE Execution Engine treats `unknown` as `false`).

4.6.4 Deprecated Action Specifications

Prior to version 1.40, the SAGE guideline model uses a set of **Action_Specifications** that had evolved in an ad hoc way. Many of the action specifications have attributes that parallel those of VMR classes. They are listed below, but their usage is discouraged.

Inquiry

Conclusion

Schedule_Appointment

Make_Referral

Provide_Education

Order

Prescribable_Item

Message_Action

Set_Goal

Retract_Conclusion

4.7 Expression Language

One characteristic of a computable guideline model that set it apart from an unstructured or semi-structured guideline document is that it has a formal representation of eligibility criteria and decision criteria that, given a set of patient information, can be evaluated to generate patient-specific conclusions. The computable representation has three components: (1) a model of patient information that defines the structure of data, (2) codified terminologies that allow matching of concepts between an encoded guideline and patient data, and (3) an expression language that defines the syntax and semantics of the decision criteria. For the SAGE guideline model, we adopt the GELLO expression language being developed by the Clinical Decision Support Technical Committee (CDSTC) of Health Level 7 as the foundation of SAGE's expression models.

GELLO is a generic expression language that can be used with any object-oriented data model. However, GELLO is a complex string-based language that is not easy to write for someone who is not trained technically. To make it possible for guideline encoders to author computable expressions, we introduce a number of classes that organize expressions into typed data values, variables, queries, functions, and criteria (as shown in Figure 16). Except for data types, each expression class corresponds to a template of stereotypical GELLO expressions. These classes can be used to generate fill-in-the-blanks forms that are much easier to instantiate. These forms also allow us to develop specialized tools to facilitate the use of standard terminologies while writing expressions. In this section, we first give a brief introduction to GELLO and show how it can be used to write a complex expression. Then we discuss the expression classes that we introduce to facilitate the use of GELLO.

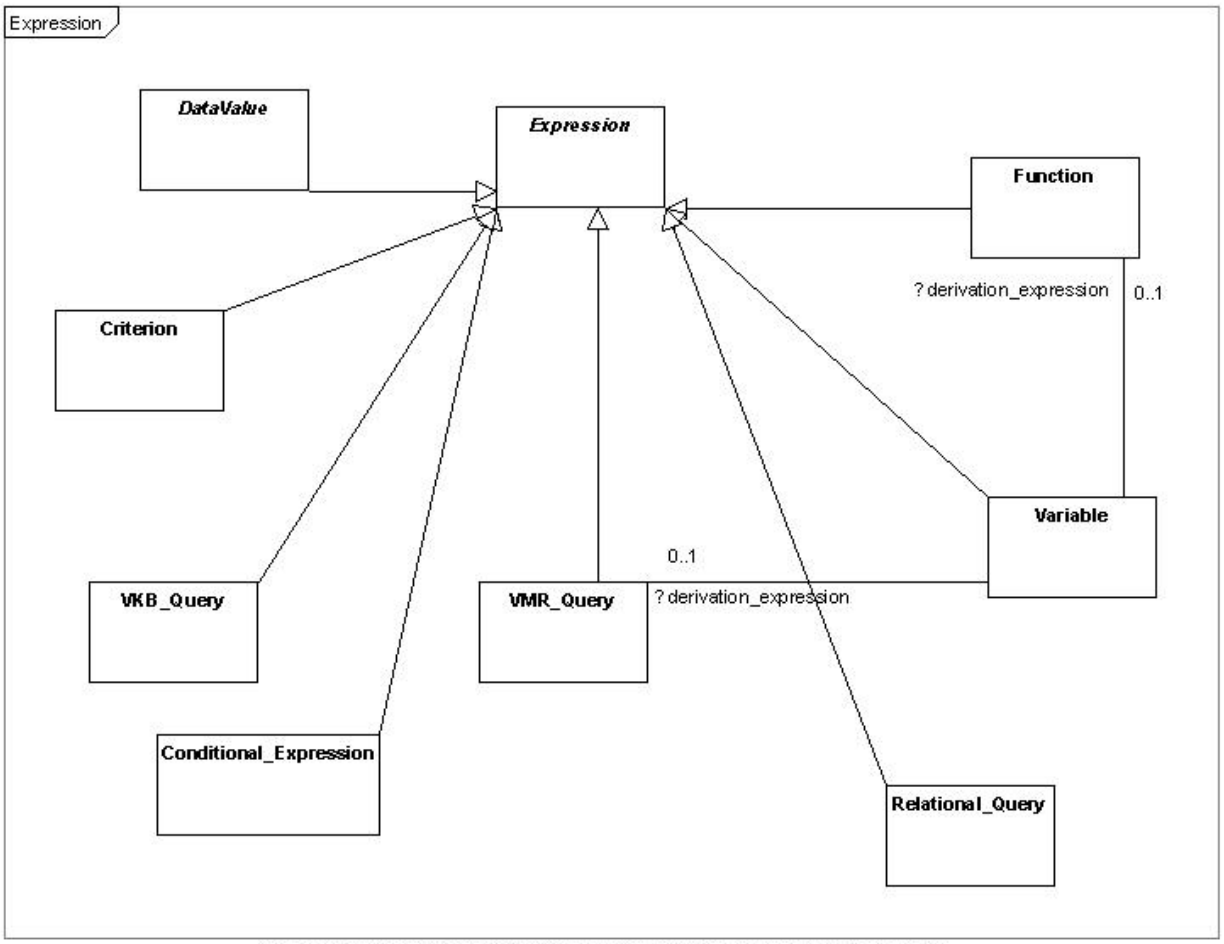


Figure 16 Classes of SAGE Expressions

Note that NULL is a possible value of an expression, including Boolean expressions. The semantics of Boolean operators are defined by the following truth tables (where other means null or a value of non-Boolean data type (e.g. 3.5, “aString”) :

AND operator	true	false	other
true	true	false	null
false	false	false	false
other	null	false	null

OR operator	true	false	other
true	true	true	true
false	true	false	null
other	true	null	null

NOT operator	true	false	other
	false	true	null

Implementation note: SAGE Execution Engine implemented binary Boolean value. When anything evaluates to unknown, that defaults to FALSE

4.7.1 GELLO

The syntax of the GELLO language depends on the use of an object-oriented data model. We refer to this as a "virtual medical record" (or vMR) (Johnson et al., 2001b). GELLO is based on the Object Constraint Language (OCL)(Object Management Group, 2003). OCL is well-developed as a constraint language and has a number of features that make it desirable for use as an expression language. However, because OCL is designed for writing constraints in an UML model, a number of adaptations have to be made for it to be suitable as a language for writing clinical decision criteria. First, a number of OCL features, such as pre- and postconditions, invariants, self and implicit references to objects, are not applicable in the guideline modeling setting. The context declaration mechanism has to be modified so that, for guideline designed to assist in the management of individual patients, the context of the expressions have to be defined to be the medical record of a single patient[5]. Similarly, we have to introduce ways of referencing instances of model classes that are not necessary in the UML modeling context. The full specification of GELLO is maintained by HL7.

In the following, we give some examples to show how GELLO expressions can be written to specify complex relationships.

The patient has a prescription for a drug such that the drug is compellingly indicated (i.e., there exists a patient problem such that the problem is a compelling indication for the drug)

We assume that we have a declarative representation of drug information such that properties of drug usage (e.g. compellingIndication) are attributes of the representation of drugs.

Three places where GELLO-lile expression occur: Function and Conditional_Expression

MedicationOrder->exists (Problem ->exist (problem: drug.AllInstances () -> exist (not (drug.compellingIndication -> intersection (problem) -> isEmpty))

Average systolic blood pressure from today

let "now" (a PointInTime) be the context-dependent name for the current time, and timeOfDay() be a method that returns the hour:minute:second portion of a PointInTime.

```
let SystolicBPCode CodedValue = CodedValue.valueOf("LOINC", "xxx")
let mmHGCode CodedValue = CodedValue.valueOf("LOINC", "yyy")
let midNightToday PointInTime = now.minus(PointInTime.timeOfDayNow()
let todaySystolicBP Set = Observation->select(code.implies(SystolicBPCode) and
effectiveTime.within(IVL<TS>.valueOf(midNightToday, now)
```

[5] As of 2004/02/11, specification of context has not been established in GELLO.

```
let SysBPSum PhysicalQuantity =
  if todaySystolicBP->notEmpty() then
    (todaySystolicBP->iterate
     sysbp : Observation
     sysbpsum : PhysticalQuantity = PhysicalQuantity.new(0, mmHGCode) |
     sysbpsum.plus(sysbp.value)
    )
let SysBPAverage PhysicalQuantity =
   if todaySystolicBP.notEmpty() then
      SysBPSum.divide(todaySystolicBP.size())
   else null
   Endif
```

4.7.2 Basic Data types

GELLO's native data types consist of string, Boolean, integer, and float. For modeling guidelines, we adopted the data types defined in HL7's abstract data type specification. The HL7 defined the UML Implementation Technology Specification for the data types (Grieve et al., 2003) defines the canonical representation of data types used in the SAGE guideline model.

We extend the HL7 version 3 data types with additional types as follows:

4.7.2.1 ParametrizedString

A **ParametrizedString** derives from the **String** data type class. It is designed to provide a template for dynamically constructed string. In addition to the *label* slot, it has a *variables* and a *value* slot. The slot value for the *variables* slot is a list of *Variable* instances. The *value* slot holds a string that may have embedded references to variables. At run time, the variables are evaluated and the string representation of the variable values is substituted into variables referenced in the *value* string.

The syntax of referencing variables in a string is "*'|(?variable)|'*, where '*' is the wild-card character and ?variable is the name of a variable.

4.7.2.2 ConditionalString

A **ConditionalString** derives from the **String** data type class. It is designed to provide a way to specify alternative text strings based on a patient-specific decision criterion. It has three additional slots: *condition* (instance of **Criterion**), *condition_true_strintg* (instance of **CharacterString** data type class), and *condition_false_string* (instance of **CharacterString** data type class). The slots *condition* and *condition_true_string* are required to have values.

ConditionalString is equivalent to a GELLO if-expression

if *condition* then
condition_true_strintg else
condition_false_strintg endif

4.7.3 Variable

An instance of the Variable class with

datatype: Type

referenced_as: variableName

derivation_expression: function, query, or data value

is equivalent to a GELLO expression of the form:

let variableName Type = derivation_expression

The Variable class has additionally optional slots “label” to give a user-friendly label to the variable name.

Figure 17 shows an example of the date of birth being specified as a variable. The derivation expression is an instance of VMR query.

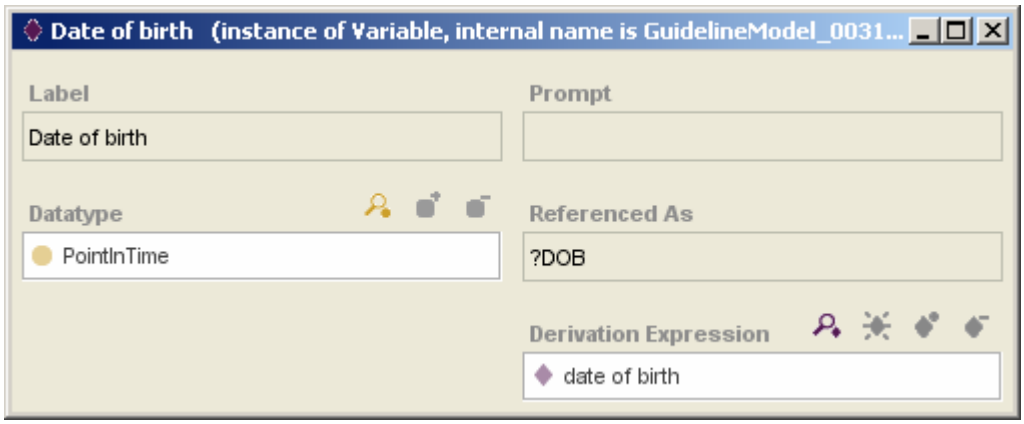


Figure 17 Example of a SAGE Variable

4.7.4 Function

An instance of the Function class is defined by a string-valued expression and a reference to the expression language used. An optional “parameters” field enumerates, for explanation purpose, the global variables used in the expression.

In terms of GELLO, a SAGE function is equivalent to the functional expression itself.

Implementation Note: The SAGE Execution Engine supports the following operators: "-", "minus", "+", "plus", "(", ")", "*", "multiply", "/", "divide", "maximum". Functions support modifiers: "new", "average", "high", "low". Types are: "Quantity", "PhysicalQuantity", "CodedValue", "SNOMED CT", "PointInTime", "month", "today", "now".

An optional *exception_alternative* field allows the user to specify a default value to use if the functional expression cannot be evaluated because of data problem.

Figure 18 shows the function that computes takes the maximum of NOW and 3 months from the time of the last HDL test. If the expression cannot be evaluated (e.g. there is no time for the last HDL test), then the value of *exception_alternative* is returned instead. Encoder should ensure that the *exception_alternative* can be evaluated.

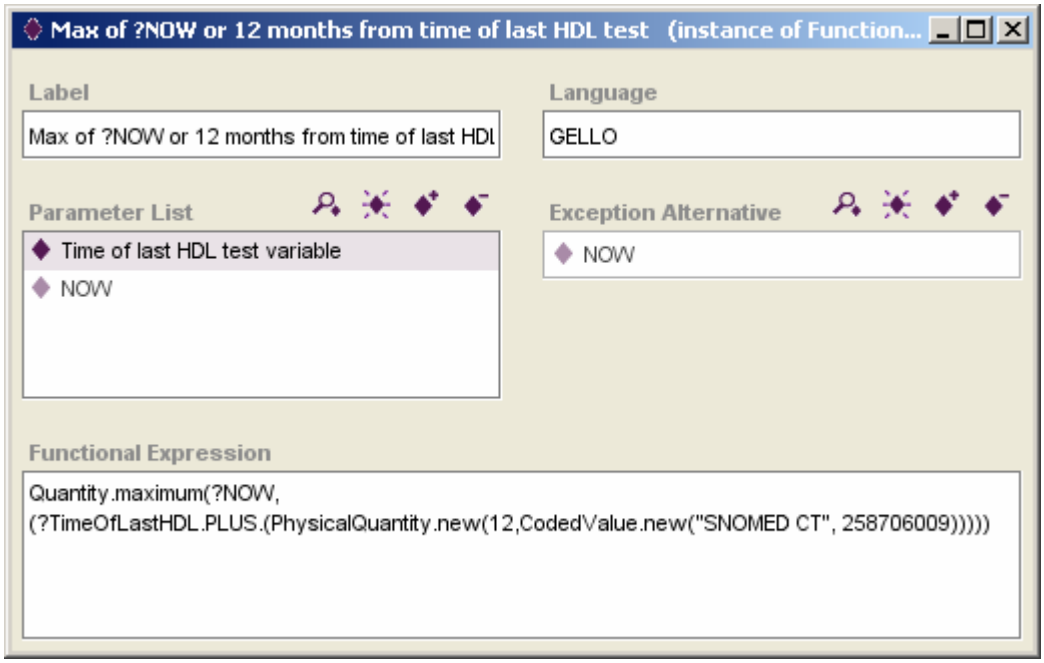


Figure 18 Example of a SAGE function (maximum of now or 12 months after last HDL lab)

4.7.5 VMR_Query

The **VMR_Query** class allows the specification of a subset of GELLO expressions that can be formulated as a constraint on the values of a single **VMR** class. The constraint (*VMR_specification* property of **VMR_Query**) is expressed as an instance of a **VMR** class that has selected attributes specified. **Figure 19**, for example, is equivalent to a GELLO expression of the form:

Observation.select (code.implies (SNOMEDCodeForDateOfBirth))

The result of evaluating the selection is a set of **VMR** instances. The set is the query result unless the *aggregation_operator* and/or *selection_attribute* properties of **VMR_Query** are used to refine the query. The *selection_attribute* specifies the property of the **VMR** class whose value should be returned. In **Figure 19**, for example, evaluating the *vmr_specification* query constraint results in a set of Observation instances whose code is the SNOMED code for date of birth. The *selection_attribute* indicates that the final query result should be the value of the *value* property of the instance whose effectiveTime is most recent.

Implementation Note: Oct 2, 2006 VMR queries default to *most recent* when no *selection_attribute* is selected (i.e., VMR_Query never returns a set; An aggregation operator is always applied to result of vmv specification query)

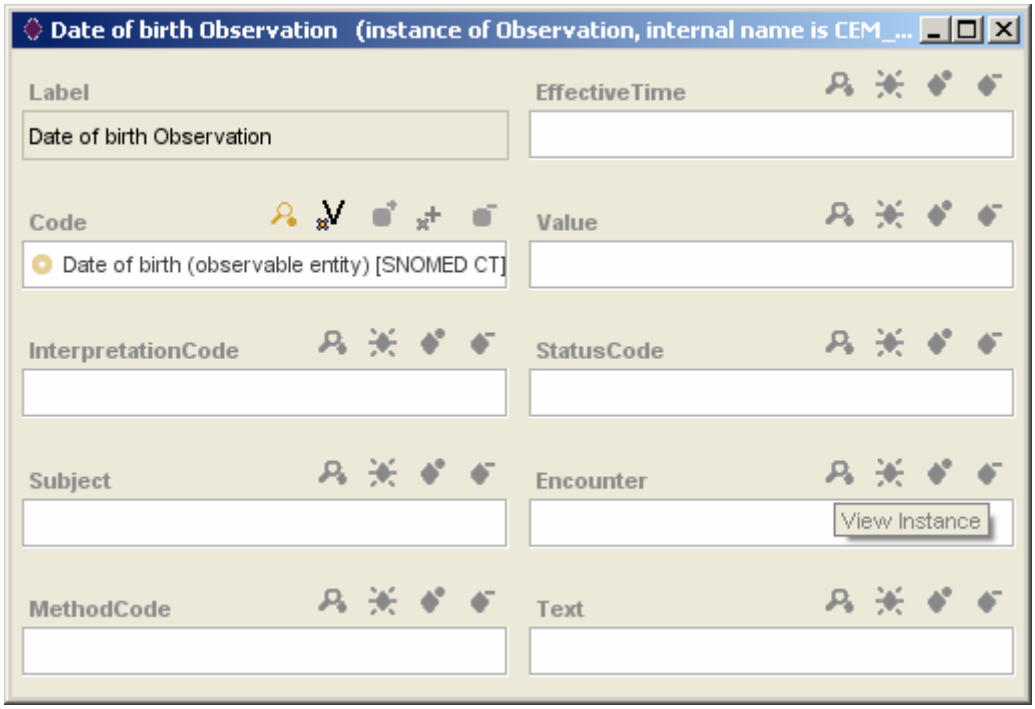


Figure 19 Example of a VMR query for the Date of Birth

A set of convenience aggregation operators have been defined. Their definitions in terms of GELLO are as follows (VMRInstanceCollection is the collection of VMR instances returned by evaluating the *vmr_specification* constraint):

most_recent

- When *selection_attribute* property is empty

  VMRInstanceCollection.sortBy(effectiveTime).last()

- When *selection_attribute* property has the value of *attribute*

  VMRInstanceCollection.sortBy(effectiveTime).last().attribute

number_of

VMRInstanceCollection.size()

first

- When *selection_attribute* property is empty

  VMRInstanceCollection.sortBy(effectiveTime).first()

- When *selection_attribute* property has the value of *attribute*

  VMRInstanceCollection.sortBy(effectiveTime).first().attribute

any

- When *selection_attribute* property is empty

  Random selection of an instance of VMRInstanceCollection

- When *selection_attribute* property has the value of *attribute*

  The value of the *attribute* property of a randomly selected instance of VMRInstanceColllection

average

See Section 4.7.1 for a description of how an average may be computed using GELLO expressions. To compute averages, the *selection_attribute* property must have values for which addition and division by an integer are defined.

Implementation Note: VMR instances as query specification is documented in SAGE Execution Engine Specification.

4.7.6 VKB_Query

In the SAGE system, external knowledge sources are modeled as virtual knowledge bases (see Section 4.9). Just like the virtual medical record (VMR), a virtual knowledge base (VKB) defines the information structure of the knowledge source so that queries can be written to retrieve information from external knowledge sources. A virtual knowledge-base query (**VKB_Query**) has a very simple structure. It has three slots:

label: a human-readable string describing the query

vkb_specification: an instance of a VKB class or **Evidence_Statement_Query_Template**

selection_attribute: selection_attribute slot specifies the property of the VKB class whose value should be returned.

Just as in a VMR query, the *vkb_specification* specifies that query evaluation should first return all instances of the VKB class whose attribute values match those in the *vkb_specification* instance. For example, the **Evidence_Statement_Query_Template** can be seen as a template for query of the form:

```
Evidence_StatementCollection->select(
{conditions}.includes(from) AND
relationship_qualifier.implies(
                    {relationship_qualifier}) AND
relationship_type.implies({relationship_type}) AND
to.implies({ti})AND
statement_subject.implies({statement_subject}) AND
strength_of_evidence.implies({strength_of_evidence})
```

The *selection_attribute* determines whether the query should return instances of the VKB class or (when *selection_attribute* is specified) the attribute values of the VKB instance.

In Figure 34, for example, evaluating the *vkb_specification* query constraint results in a set of **CodedValue** instances that represent the ACE inhibitor drugs that a patient is taking. The *selection_attribute* indicates that the final query result should be the value of the *NDF_Maxiumum_Daily_Dose_Strength* property of the instances. (We assume that a set of one element is casted into the element itself.)

4.7.7 Relation_Query

An instance of Relation_Query is designed to query for instances of relations (e.g., **Evidence_Statements** that match patient conditions. Figure 20 shows an example of a Relation_Query that searches for all evidence statements on absolute contraindications of ACE Inhibitor such that the *condition* evaluates to true and then returns the *from* slot values (i.e., the medical conditions that constitute absolute contraindications). The semantics of a **Relation_Query** cannot be expressed in GELLO because GELLO does not have an *evaluation* operator that evaluates a Boolean expression as part of the language. The semantics of relation queries can be expressed informally as follows:

```
SELECT valueof selection_attribute FROM instances of
[relationship_class] where
     relationship_type subsumes[ relationship_type] AND
     relationship_qualifier subsumes [relationship_qualifier]
     AND
     statement_subject subsumes [statement_subject] AND
     to subsumes [to] AND
     [from] evaluates to TRUE, FALSE, or UNKNOWN
```

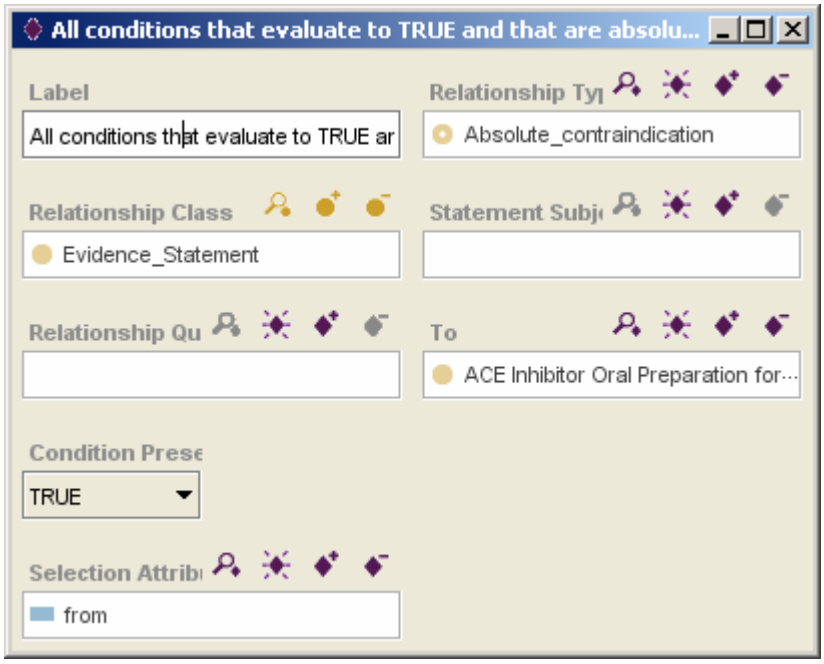


Figure 20 An example of Relation_Query that searches for all evidence statements on absolute contraindications of ACE Inhibitor such that the *condition* evaluates to true.

4.7.8 Conditional_Expression

A **Conditional Expression** an expression designed to provide a way to specify alternative data values based on a patient-specific decision criterion. It has three additional slots: *condition* (instance of **Criterion**), *if_true_condition* (instance of **Expression**), and *if_false_condition* (instance of **Expression** class). If the *condition* slot evaluates to true, then the evaluated

if_true_expression is returned. If the *condition* evaluates to false, then the *if_false_expression* is returned. The slots *condition* and *if_true_string* are required to have values.

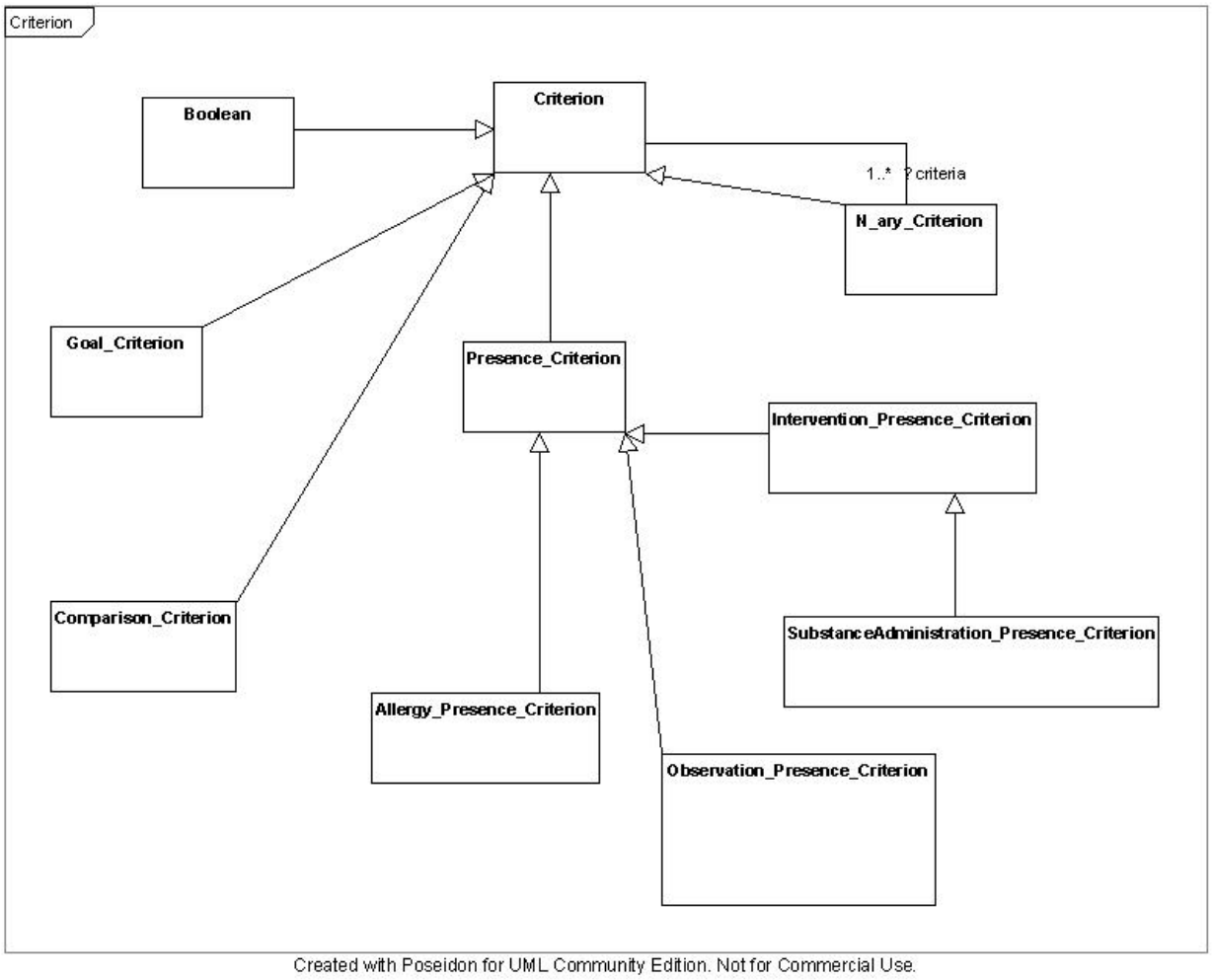


Figure 21 Subclasses of Criterion

4.7.9 Criteria Templates

Criteria templates are structured templates that allow a user of the SAGE guideline model to encode decision criteria in a syntax-independent form-based method. It also allows workbenches for encoding guidelines to develop specialized graphical user-interface and tools to facilitate the guideline authoring process.

There are five major types of criteria templates: Boolean combination, comparison, existence (presence or absence), goal-satisfaction, and GELLO criteria that can be used when other types are not sufficient (Figure 21). To determine which of the numerous criterion classes to use, follow the following algorithm:

If the expression you want to write evaluates to true, false, or unknown, then

1 If the expression is a BOOLEAN combination (AND, OR, or NOT) of simpler criteria, then use **N_ary_Criterion**

2 If you want to check to see whether the goal (i.e. target range) of a measurable clinical data is satisfied, use **Goal_Criterion**

3 If you want to make some kind of comparison (e.g. comparing values or time, or whether value is within a range), then you want to use one of the comparison criteria:

3.1 If you query for one of more instances of a VMR class according to the code and valid time, aggregate the result somehow (e.g. take average, count number of instances, or select the first or last instance) and compares the result (or the value of the "value" slot in the case of Observation VMR class) to some value, then use **Comparison_Criterion**.

3.2 If the expression is not comparing the value of an instance (or the average value), but comparing the timing of a VMR instance that is selected using code and valid time as constraint, then use the **Temporal_Comparison_Criterion**.

3.3 If the comparison you want to make cannot be formulated comparing the value or time of a VMR instance to some value or time, then you should consider the **Variable_Comparison_Criterion**, which simply compares the value of a variable to some other value.

4 If the expression checks the presence or absence of some VMR instance, then you should one of the "presence" criteria.

4.1 If the instances whose presence|absence you are checking are instances of the Adverse_Reaction class, then use **Adverse_Reaction_Presence_Criterion**

4.2 If you only want to check the presence|absence of a VMR instance with a specific code and whose valid time (*effectiveTime* slot) intersect time window (from some past time point to now), then use **Presence_Criterion**

4.3 If, additionally, the instance whose presence|absence you are checking have certain *statusCode* or *priorityCode* then use **Intervention_Presence_Criterion**

4.4 If the instances whose presence|absence you are checking depend the properties *rateQuantity*, *doseQuantity*, *routeCode*, and *duration* (all properties of SubstanceAdministration or MedicationOrder) then use **Medication_Presence_Criterion** [Implementation note: SAGE GEE implements Medication_Presence_Criterion as Presence_Criterion.)

4.5 If the instances whose presence|absence you are checking are instances of Observation and the criterion depends on the properties *value* and *methodCode* then use **Observation_Presence_Criterion** [Implementation note: SAGE GEE does not implement matching of *methodCode* in Observation_Presence_Criterion.)

5 If you want to check to see if any of the evidence statements applies for the case, then use the **Relation_Presence_Criterion.** This criterion type allows you to specify the subset of evidence statements you want to check (e.g., all evidence statements regarding *compelling indications* (relation_type) for use of *ACE inhibitors* (to) in the *treatment of hypertension* (subject)).

6 If the criterion you want to write cannot be written using any of the above criterion template, then you need to use GELLO criterion.

4.7.9.1 N_ary_Criterion

An **N_ary_Criterion** is a Boolean combination (`AND, OR, NOT`) of other criteria. It is equivalent to the use of “and”, “or”, or “not” operators in GELLO.

Example:

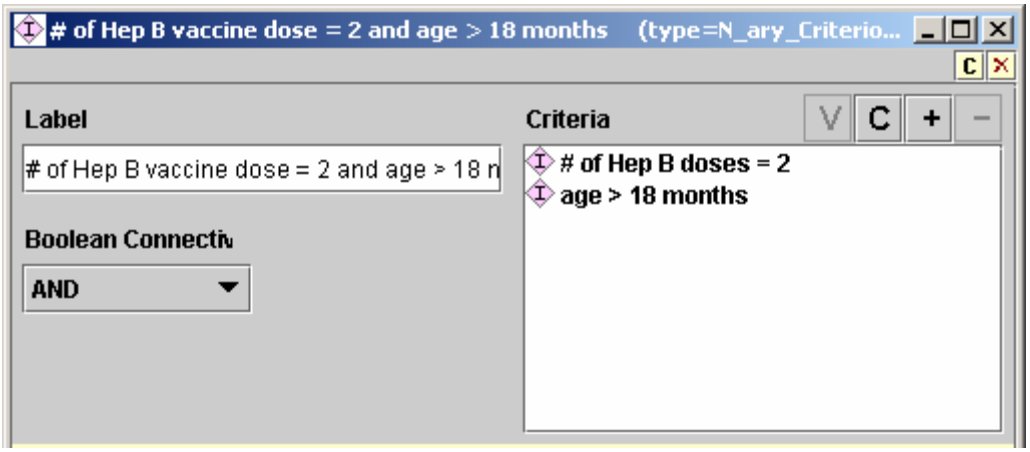


Figure 22 An example of N_ary_Criterion

4.7.9.2 Comparison_Criterion

The comparison criterion is used to express criteria of the forms:

1. When *VMR_class* is Observation: Is the (most_recent|average_of|first|second|third|any|all|number_of) Observation value(s) within *valid_window* (i.e. *effectiveTime* intersects with *valid_window*), whose coded-concept is specified in the *code_concept* slot *equals, >, <, >=, <=, not equals, in, not in, or implies* the value of *value* slot. The *assume* property indicates what assumptions should be made when there is no Observation available to make the comparison. The possible assumptions are *no_assumption, assume_true,* and *assume_false*.

   Null valid window defaults to 'ever in the past.'

   Example: “Most recent creatinine clearance < 70 ml/min”

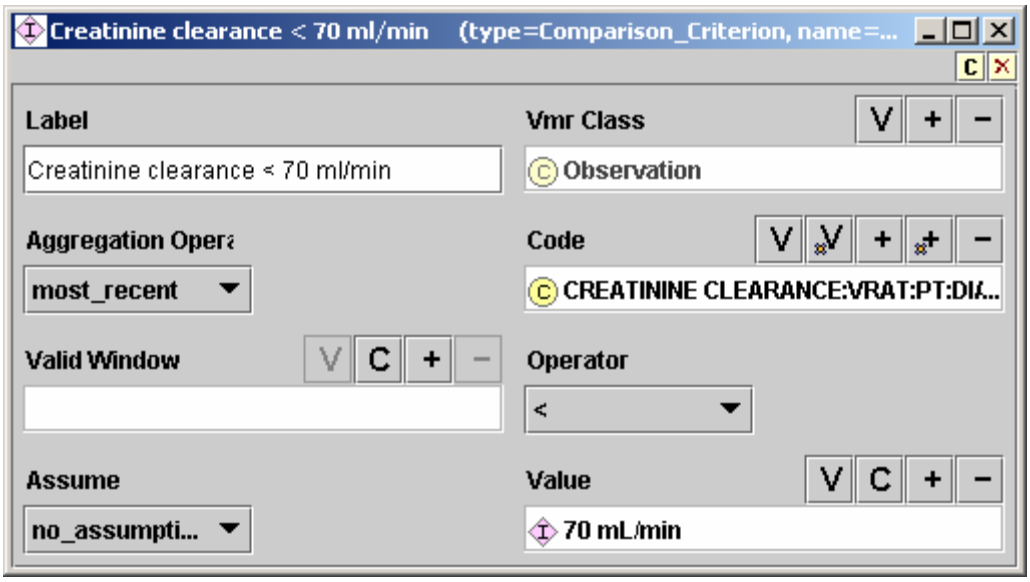


Figure 23 Comparison Criterion "value of most recent Serum Creatinine > 2 mg/dL

2. If vmr_class is not Observation, then the aggregation operator should be `number_of`. We use this to check if the *number of* instances of such vmr class that satisfy the specified constraints is equal to, greater than, less than, or not equal to the value of *value* slot.

   Example: The number of DTP vaccinations in the past = 3 (Figure 24).

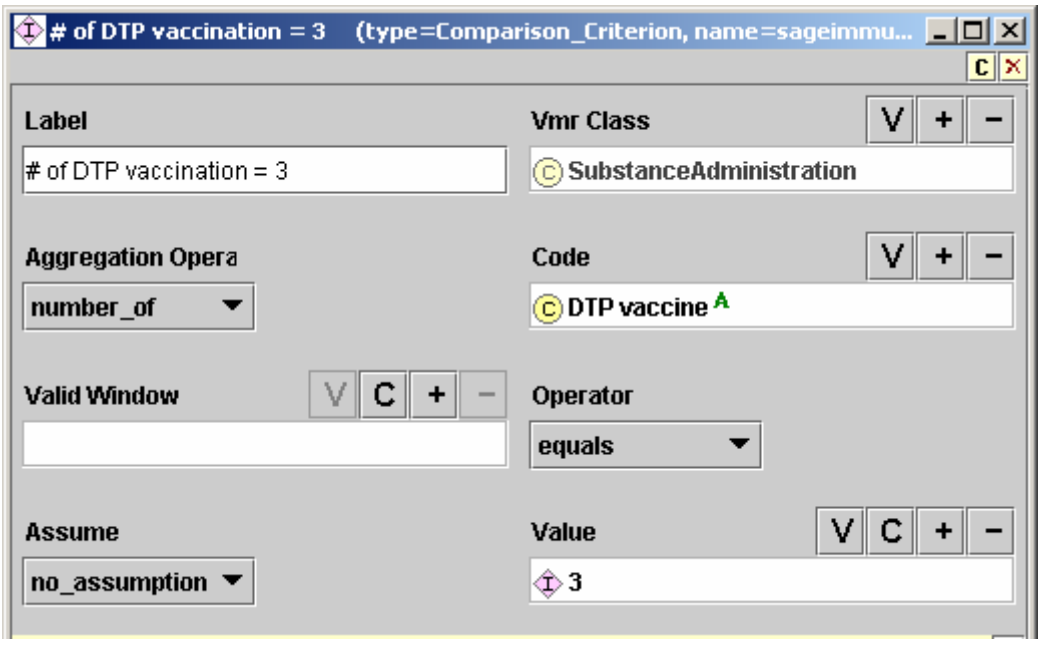


Figure 24 Comparison Criterion "Number of DTP vaccination substance administration in the past = 3"

4.7.9.3 Presence_Criterion

A presence criterion checks for presence or absence of coded concept in instances of a VMR class within the valid window. There are five flavors of **Presence_Criterion**, as described in Section 4.7.9. Their usages are fairly similar, we will describe **Adverse_Reaction_Presence_Criterion**. Details of the other types Presence Criterion can be found in the javadoc-style specification documents.

4.7.9.3.1 Adverse_Reaction_Presence_Criterion

An **Adverse_Reaction_Presence_Criterion** checks for the presence or absence of an **AdverseReaction** instance that satisfies the *code*, *substance*, *reaction*, *severity*, and *effectiveTime* (*valid_window*) constraints specified in the criterion. In terms of GELLO, it can be formulated as

> VMR.AdverseReaction->exists(code.implies({*code*} and effectiveTime.intersect({*valid_window*})) and substance.implies({*substance*}) and reaction.implies({*reaction*}) and severity.implies({*severity*}))

Figure 25 shows an instance of *Adverse_Reaction_Presence_Criterion* class that expresses “No allergic reaction to ACE Inhibitor drug now or in the past”

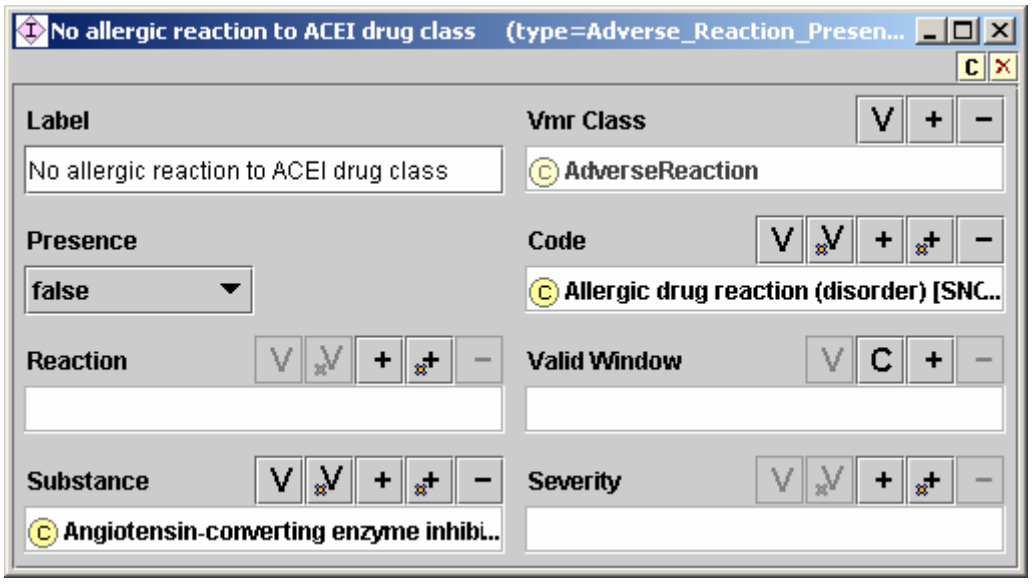


Figure 25 An example of Adverse_Reaction_Presence_Criterion

4.7.9.4 Temporal_Comparison_Criterion:

A temporal comparison criterion compares the temporal relationship (before, after, concurrent, ...) between the time of an VMR instance (with specific coded_concept and aggregationoperator) and some time_for_comparison. In terms of GELLO, if we use {*slotname*} to denote the value of the slot, the criterion can be written as

> {*VMR_Class*}.exists(code.implies({*code*} and effectiveTime.intersect({*valid_window*})).{*aggregation_operator*}().{*time_attribute*}.

{*operator*}
({*time_for_comparison*})

Thus, the example shown in Figure 26 can be written as

```
SubstanceAdministration.select(code.implies(SNOMED code for
Hib vaccination)).first().effectiveTime.before(15 months of
age)
```

The possible values of *aggregation_operator* are

```
most_recent, first, second, third, fourth
```

The possible values of operator are

`before, after, concurrent, before_or_on,` and `after_or_on, intersect`

Since the *time_attribute* and *time_for_comparison* can be either time interval or time point, a time point is made into an interval with the same high and low points. The concurrent operator should be interpreted as equality of time intervals.

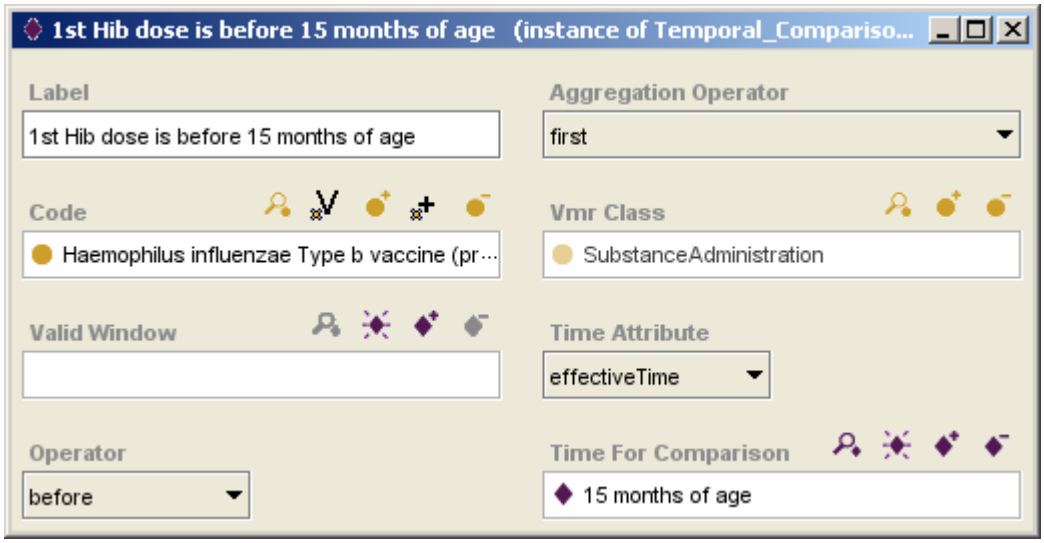


Figure 26 Example of Temporal_Comparison_Criterion: First Hib dose was given before 15 months of age

4.7.9.5 Variable_Comparison_Criterion

A **Variable_Comparison_Criterion** compares the value of a variable to some other value. You would define the variable first as a query or a function (See section 4.7.3 to 4.7.6), then compare the value of the variable to some other value. The possible comparison operators are `>, >=, <, <=, in, not_in, equals, and not_equals`. The semantics of the operators depend on the data types of the variable and expression being compared. The `in` and `not_in` operators, for example, assumes that the *value* slot is a multi-valued data type such. An

implementation of this type of criterion assumes that there is a mapping from the comparison operators to the underlying methods of the data types.

Implementation Note: `in` and `not_in` operators are not implemented in SAGE Execution Engine.

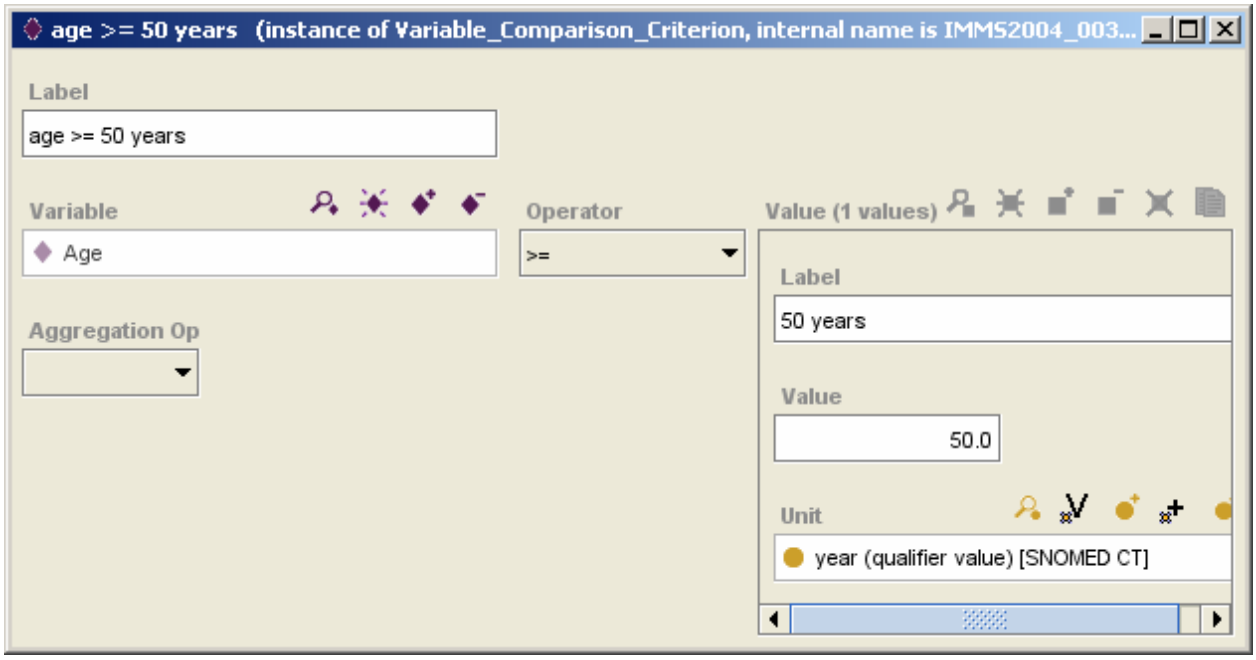


Figure 27 An example of Variable_Comparison_Criterion

4.7.9.6 Goal_Criterion:

The **Goal_Criterion** evaluates the status of a goal. The current SAGE **Goal_Criterion** template only allows a user to specify a simple criterion that checks that the most recent value of the **Observation** with the matching code is within the target range of the goal that has the same code. Thus, for example, the **Goal_Criterion** in Figure 28 should return true only if the most recent value of HDL observation is within the target range of the most recent HDL goal.

More complex goal criteria (for example, that the average of the last two observations should be, say, within 10% of the target range) can be encoded using explicit comparison criteria. For example, “average of the last two observations” and “10% of the target” range can be constructed using VMR queries and functions written in GELLO.

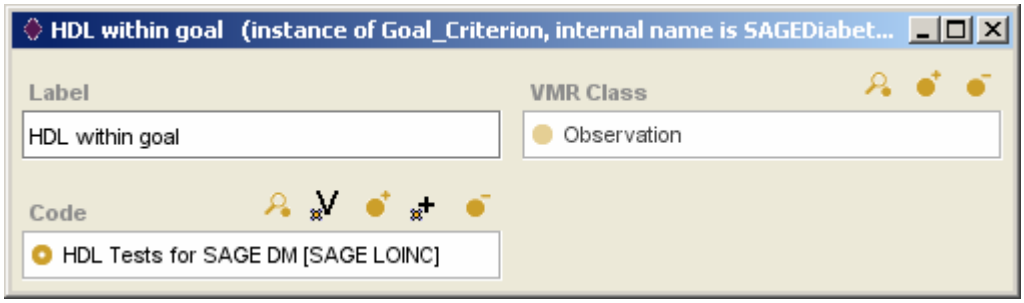


Figure 28 Example of a Goal_Criterion

4.7.9.7 Relation_Presence_Criterion

We define **Relation_Presence_Criterion** to specify whether there exist instances of evidence statements that match specified attributes and where the condition is present. For example, Figure 29 shows a query: is any instance of **Evidence_Statement** for which the condition that is a compelling indication of ACE Inhibitor present?

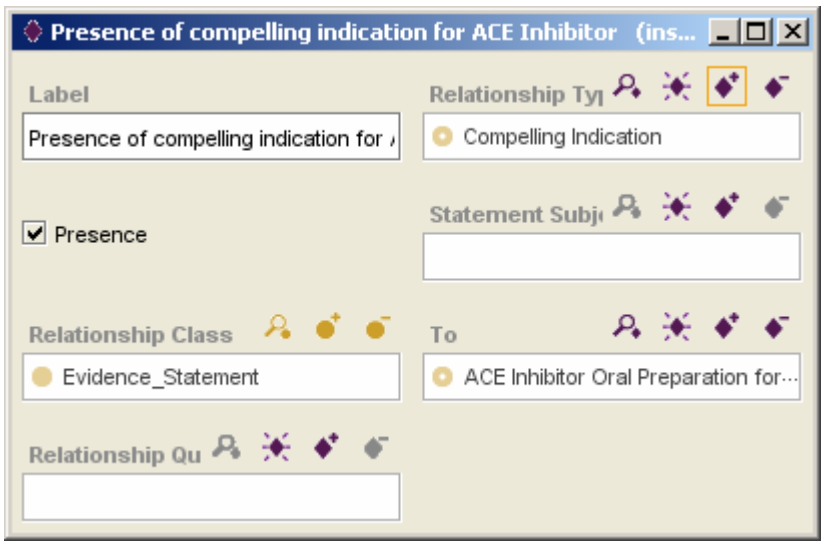


Figure 29 Example of a Relation_Presence_Criterion

The criterion can be used as a strict rule-in criterion for adding ACE inhibitor to a patient's medication list. Thus, the *strict rule-in* slot of a decision alternative does not have to enumerate specific conditions for adding ACE (Figure 30).

Implementation Note: The SAGE Execution Engine does not use *relationship_qualifier*, *statement_subject* in matching for relation instances.

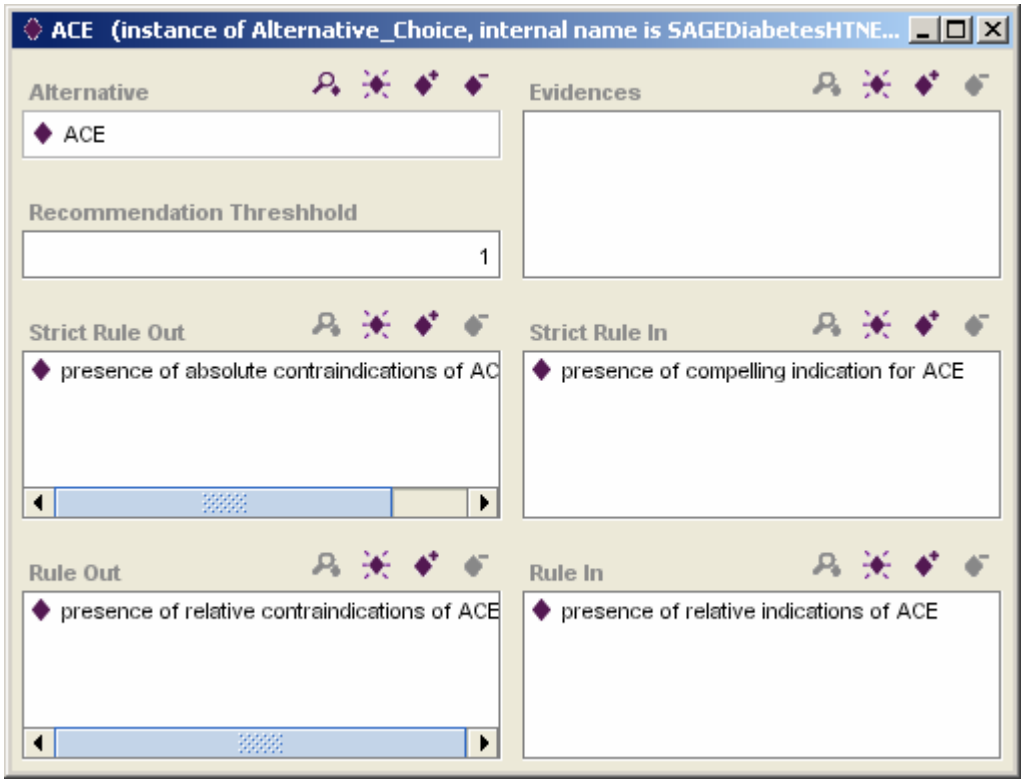


Figure 30 Example of the usage of relational query in a decision model

4.7.9.8 GELLO Criterion:

SAGECriterion is a "fall-back" criterion type for writing criteria that cannot be expressed using other criterion template.

The **GELLOCriterion** class allows the specification of a textual *expression*. The *let* property specifies a set of **Variable** instances that are available for explanation purpose.

A decision criterion used in a guideline line may be arbitrarily complex. It may mix medical knowledge with patient data. For example, a statement like “There exists (for a patient) an anti-hypertensive prescription (*?prescripton*) such that there exists (for the patient) a problem (*?problem*) such that *?problem* is a compelling indication for *?drug*” uses a relationship (compelling indication) between a patient problem and a class of drugs. If such relationship is defined through instances of a ConceptRelation class that defines *relation_type*, *from*, and *to* as properties (where *from* and *to* have as their values coded concepts), then the statement can be written as follows:

MedicationOrder->exists(?prescription: Problem->exists(?problem: ConceptRelation->exists(?relation: ?problem.code.implies(?relation.from) and ?prescription.code.implies(?relation.to) and codeForCompellingIndication.implies(?relation.relation_type)

4.8 Order Set

An executable SAGE guideline interacts with order sets based on the following assumptions:

- There is an external reference encoding of order sets
- SAGE can reference specific items in the order set by their unique ids and offer decision-support services by modifying the content of the order set
- SAGE may process logic based on patient data and context to:
    - Select among pre-loaded order sets.
    - Determine if an item within an order set is to be **pre-selected.**
    - Compute specific **annotations** for those orders.

Because of the need to compute annotations and to pre-select appropriately, the representation of order sets in the SAGE guideline model correspond to, but is not identical to the representation of a standard order set. Specifically, a SAGE assumes that order sets will have the following structures:

- An **Order_Item** consisting of the following attributes:
    - *identifier* that uniquely identifies the corresponding order item in the reference encoding of the order set
    - *orderitem* that can be an instance of a **VMROrder**, **Observation**, or **Goal**
    - *condition* a Boolean criterion that, if evaluate to true, pre-select this order item
    - *fullTextOrder* that is a textual statement of the order item
    - *orderAlertingOrExplanationText* that hold zero or more strings that may be instances of **ParametrizedString** (see Section 4.7.2.1) or **ConditionalString** (see Section 4.7.2.2)
- A **Boolean_Order_Collection** which has the attributes *condition*, *identifier*, and *orderAlertingOrExplanationText* (similar to those of **Order_Item)**. In addition, it has
    - *group_body* that is a required collection of **Order_Item** or **Boolean_Order_Collection** instances.
    - *boolean_connective* that may have one of `XOR`, `OR`, and `AND`. `XOR` means only one of the items in group_body should be performed; `AND` means all of the items in *group_body* should be performed, and `OR` means at least one of the items in *group_body* should be performed.
    - *title* a textual string describing the collection of orders
- An **Order_Set** that is just a **Boolean_Order_Collection** with an *additional metadata* slot**.**
- An **OrderSet_Metadatat** class that inherits the attributes *date*, *developer*, *identifier*, *publisher*, and *title* from the **Resource_Metadata** class. The attributes specific to **OrderSet_Metadata** are
    - *ageRestriction*: an instance of a **Criterion**.

- *genderRestriction*: an instance of a **Criterion**.
- *problemCode*: zero or more instances of **CodedValue**
- *sessionCode*: a class with allowed parent **Clinical_Setting**
- *references*: zero or more instances of **Supplemental_Material**

The class hierarchy for the order set constructs is depicted in Figure 31.

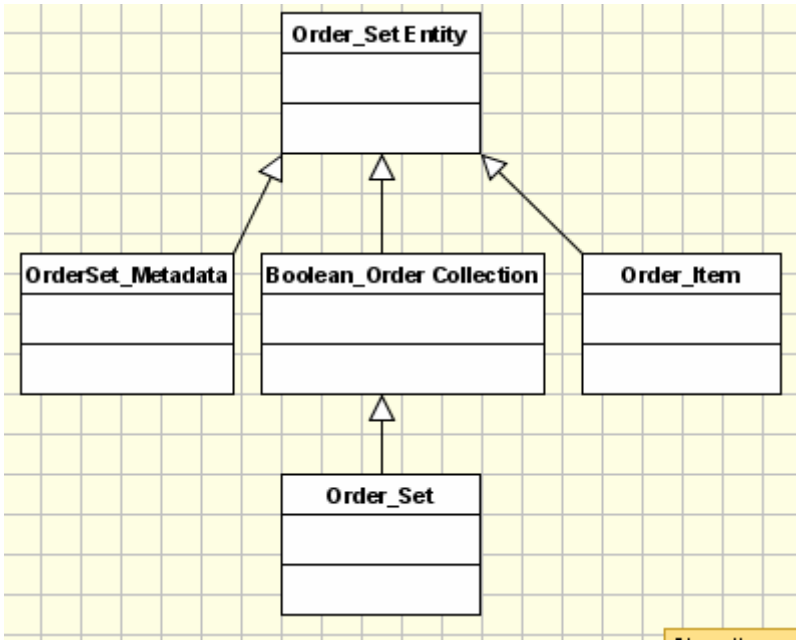


Figure 31 The class hierarchy for order set entities

Figure 32 shows an example of a SAGE order set.

SAGE implements external order sets as XML documents that conform to an order set schema. When SAGE execution engine recommends an order set, it uses patient information to compute the pre-selection flag and annotations that are specified in the guideline order set model. The output is a patient-specific XML file that is merged, through XSLT, with the external order set to produce an annotated order set for a patient (Figure 33).

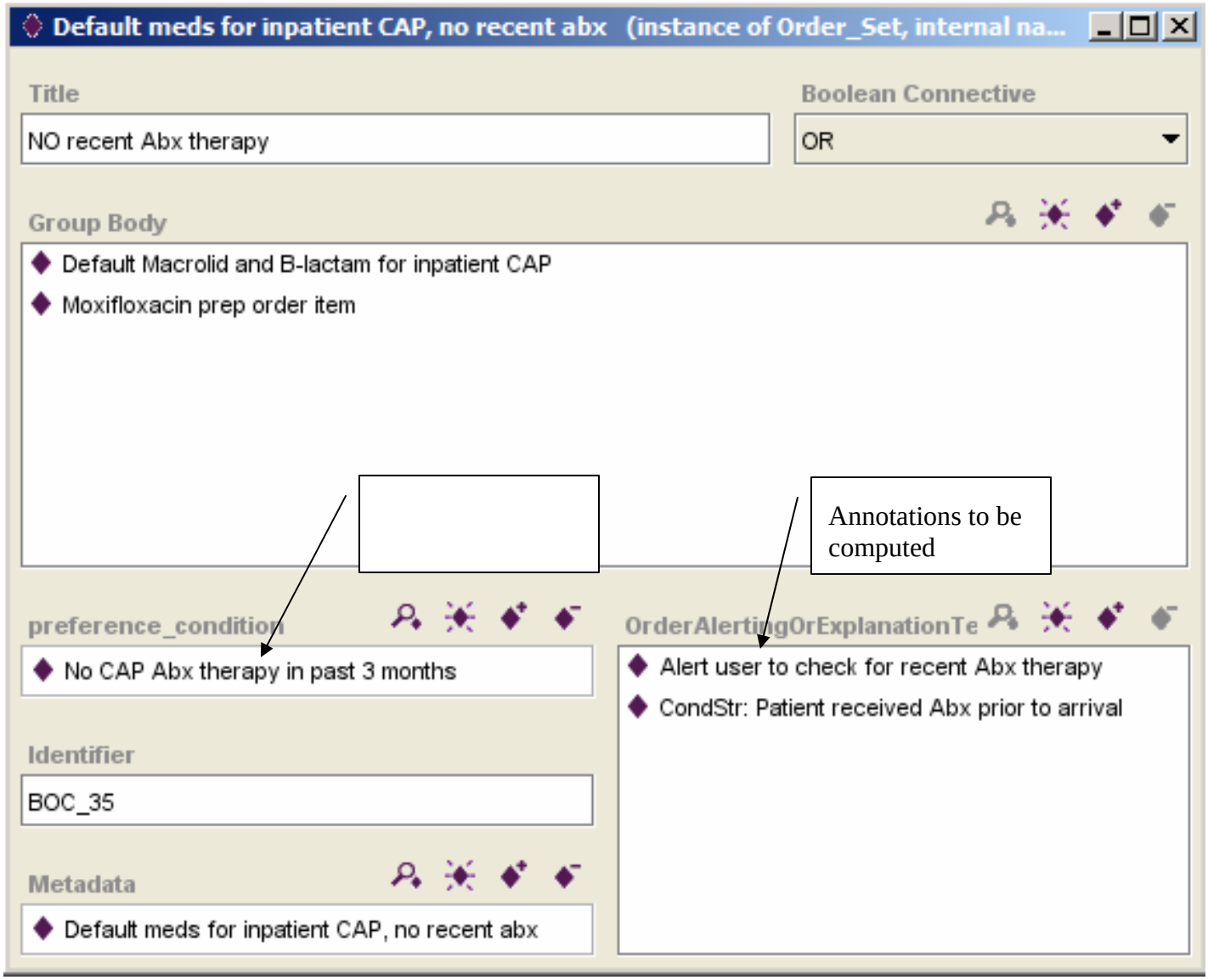


Figure 32 An example of SAGE order set (*preference_condition* is the display label for the *condition* slot).

Error! Objects cannot be created from editing field codes.

Figure 33 The Medication section of the CAP order set that shows checked pre-selection flags aand patient-specific annotations.

4.9 Use of External Knowledge Sources

The application of a guideline to the care of a patient requires that we augment the guidance specified in the guideline with general medical knowledge. It is neither practical nor desirable to assume that all applicable medical knowledge is part of a monolithic guideline knowledge base.

Thus, effective guideline encoding requires that we have well-defined method for accessing medical knowledge outside the guideline model.

The SAGE Guideline Model does not mandate a particular division between medical knowledge that should be encoded as part of a guideline knowledge base and that should be externally supplied. The division is likely to depend on the guideline and on the application environment. Thus, for example, the Seventh Report of the Joint National Committee on Prevention, Detection, Evaluation, and Treatment of High Blood Pressure discusses indications and contraindications of various anti-hypertensive agents.[6] The information contained in a generic drug knowledge base may overlap with those described in the guideline.

For referencing external knowledge sources, we take the same approach as what we do with patient data. That is, we define a virtual interface that can be mapped to the real knowledge source. The virtual interface consists of a collection of entities and relationships based on which decision criteria can be written.

A drug knowledge base is the most important external knowledge source that a SAGE guideline can access. For the SAGE project, we adopt the drug model defined in the National Drug File - Reference Terminology (NDF-RT)(Carter et al., 2002). In particular, we use the specifications of **ActiveIngredientPreparation**, **DrugComponent**, and **ClinicalDrug**, where **ActiveIngredientPreparation** (e.g. LISINOPRIOL PREPARATION) has roles such as "may_treat," and "CI_with" (contra-indication with), where **DrugComponent** (e.g. LISINOPRIL 40 MG) are defined by properties such as *NDF_Strength* (40) and *NDF_Units* (MG), and where **ClinicalDrug** (e.g. LISINOPRIL 40 MG TAB) adds the *DoseForm* (e.g. TAB) property. Furthermore, **ClinicalDrug** (e.g. HYDROCHLOROTHIAZIDE 12.5MG/LISINOPRIL 10MG TAB") may be made up multiple **DrugComponents**.

We assume that entries in the external knowledge source are indexed by a terminology code. Thus, **ActiveIngredientPreparation**, **DrugComponent**, and **ClinicalDrug** are modeled as subclasses of **Coded_Value**.

To illustrate how we use this drug model to access the NDF-RT as an external knowledge source in guideline encoding, we show how the drug model and the SAGE the virtual medical record (VMR) are combined to find the *NDF_Maxiumum_Daily_Dose_Strength* of the ACE inhibitor that is in a patient's medication list. We will work backward. First, in Figure Figure 34, we show the derivation of the *NDF_Maxiumum_Daily_Dose_Strength* of the ACE inhibitor as a **VKB_Query** (see Section 4.7.6 for a description of **VKB_Query)** :

[6] For example, "ACEIs and ARBs are contraindicated for women who are or intend to become pregnant because of the risk of fetal developmental abnormalities"

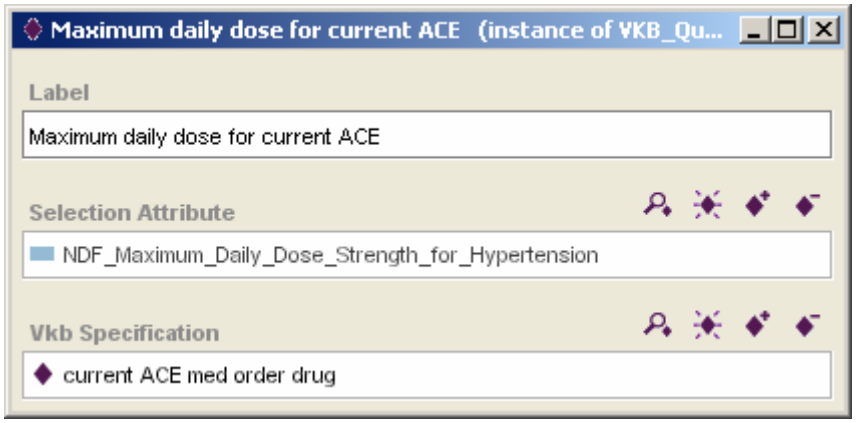


Figure 34 A VKB query to find the maximum daily dose for the current ACE inhibitor that a patient is taking

"current ACE med order drug" (Figure 35), is a variable whose value is derived from a VMR query:

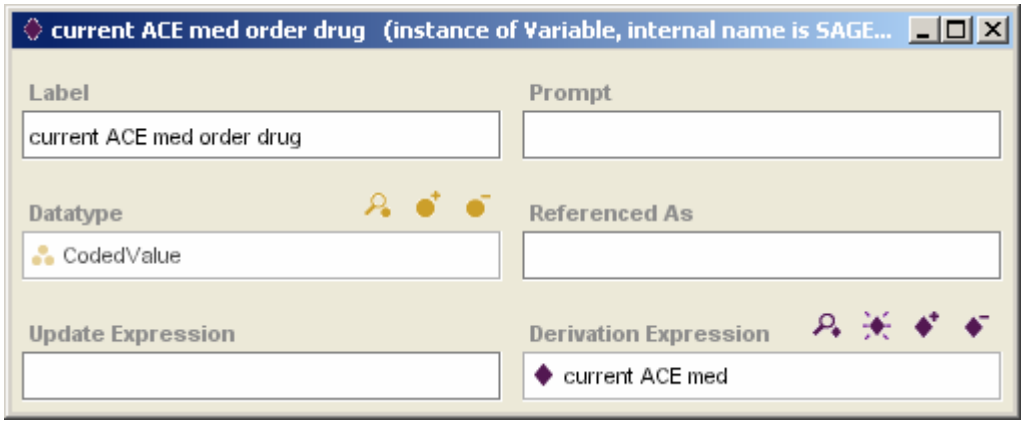


Figure 35 The "current ACE inhibitor medication variable

The "current ACE med" derivation expression is a VMR query that uses attribute selection (Figure 36:

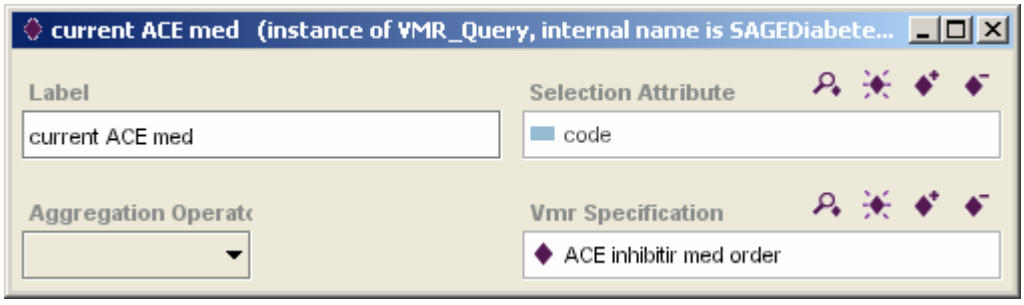


Figure 36 A VMR query to get the code of the current ACE inhibitor medication order

Here we make the assumption that the code of an medication order is a **ClinicalDrug** that has the *NDF_Maxiumum_Daily_Dose_Strength* attribute that we are querying in Figure 34.

As a second example, we show how to compute the daily dose of a drug component of a MedicationOrder using GELLO:

Let ?med be an instance of the VMR class, then

QuantityPerDayInOriginalUnit(?med) =
?med.repeatNumber * ?med.doseQuantity*(?med.repeatNumber.unit/"per day")

(e.g. 2 tablets 'every 6 hours" = (per 6 hours) * 2 tablets * (per day / per 6 hours) = 2 * 4 tablets per day = 8 tablets per day)

Let assume that the value of the code slot of **MedicationOrder** has been mapped to **ClinicalDrug** codes, and a **ClinicalDrug** ?D has a predicate hasComponent(?x) which is true if ?x is a drug component of ?D. Then the daily dose of drug component ?x (e.g. LISINOPRIL 10MG) for a patient who has been prescribed a MedicationOrder ?med is:

If ?med.code.hasComponent(?x) then

DosePerDay(?med, ?x) = QuantityPerDayInOriginalUnit(?med) * ?x.NDF_Strength/?med.doseForm

(e.g. 8 tablets/day * 10 mg/tablet = 80 mg/day)

From the point of view of a guideline encoder, the use of terms like ?drugComponent.NDF_Strength is similar to a reference to a property of a guideline instance. However, the SAGE execution engine has the responsibility to use the API provided by the external knowledge source (in this case the Apelon TDS server) to fetch the necessary information in evaluating the expression.

4.10 Interchange Format

The SAGE guideline model is represented in Protégé. Protégé supports multiple file formats, including RDF Schema, XML Schema, and CLIPS format. The RDF Schema format represents classes as part of a generated RDF Schema. The XML Schema format uses a fixed schema for Protégé classes, slots, and instances. For details, see http://protege.cim3.net/cgi-bin/wiki.pl?ProtegePluginsLibraryByType#nid3T2

4.10.1 XML Schema

We create an XML-based syntax to serve as the exchange syntax by defining a SAGE Guideline XML schema that constrains the legal XML representation of the guideline instance. Note that the XML Schema is NOT the normative definition of guideline model. XML Schema is not sufficiently expressive (e.g. no multiple inheritances for derived types). It just defines the syntax of the exchange format.

4.10.1.1 Design Decisions

The design decisions for the XML schema are as follows:

- The **Guideline**, **Recommendation_Set**, Variable, and **CEMMetaclass** classes are classes whose instances are elements under the top-level element.

- For each class that is not a terminology code, constraints on data model, or a system class, create a complex type that has a required ID type attribute (to hold the value of instance id) and whose subelements are properties of the class.
- For a class that has multiple superclasses, use annotation to select one parent for the purpose of XML schema complex type. The classes for which a single superclass has to be chosen are:
  - (setParentForSchema "Boolean" "Criterion")
  - (setParentForSchema "ConceptDescriptor" "DataValue")
  - (setParentForSchema "Guideline_Metadata" "Resource_Metadata")
  - (setParentForSchema "OrderSet_Metadata" "Resource_Metadata")
  - (setParentForSchema "Deprecated_Action_Specification" "Action_Specification")
  - (setParentForSchema "RelativeTimeInterval" "TimeInterval")
  - (setParentForSchema "Transition" "Recommendation_Specification")
  - (setParentForSchema "Evidence_Statement" "Guideline_Model_Entity")
  - (setParentForSchema "HL7_Data_Type" ":STANDARD-CLASS")
  - (setParentForSchema "ActMetaclass" ":STANDARD-CLASS")
- There are three special complex types: **_THING** (from which all types are derived), **_STANDARD-CLASS** (from which the metaclasses **HL7_Data_Typ**e, **ActMetaclass**, and **CEMMetaclass** are derived), and **_STANDARD-SLOT** which is the type of whose values are slots themselves.
- Slot values whose value type is “Class” will be constrained to be instances of **_Thing**.
- Cardinality constraints on slots translates into XML cardinality constraints (Slots whose cardinality is multiple become subelements that are “unbounded” (i.e. there may be repeated subelements that have the same tag))
- Values of the following slots are made to be references only (i.e. no expansion of instance values as subelements).
  - *subrecommendations*
  - *recommendation_set*
  - *alternative*
  - *:FROM*
  - *:TO*
- Slot values whose value type is “Symbol” will be made into strings with a list of allowed strings. (Not done yet)
- For each property that has a complex value type (i.e. an association in UML), create a sequence of either one element (for cardinality single slot) or a set of elements (for

cardinality multiple slot), where the element tag is the allowed class of the property value. In the case there are multiple allowed classes, then use the most specific class that is a generalization of all possible allowed classes. In case of an allowed class whose child has multiple inheritances, find the most specific class that is a generalization of all possible allowed classes.

- Special characters such as ":" and " " are converted to "_"
- Changes to Guideline Model:
  - **Evidence_Statement** made a subclass of **Guideline_Model_Entity**
  - Made "**Deprecated_Action_Specification**" a subclass of "**Action_Specification**" (so xml schema can generate appropriate base class for *action_spec*,*then*, and *else* slots).
  - Removed "**Expressiion**" as an allowed class for "*display_data*" slot of "**Display**"
  - Added **Order_Set_Entity** as a superclass of **OrderSet_Metadata**
  - Set minimum cardinality of *steps* slot in **Activity_Graph** class = 1

Known limitations of the current schema-generation software:

- All complex type derivations are by "extensions." Some should be by "restriction."

4.10.1.2 Implementation of XML Schema Generation

The generation of XML schema is based on a set of annotations to Guideline Model. The annotations are instances of annotation classes (i.e. subclasses of Protégé :ANNOTATION class). A Python script generates the schema from the GuidelineModel project. Another script generates the XML export from a particular Protégé guideline knowledge base.

4.10.1.2.1 Annotations on Guideline Model Classes

The annotation classes have the following structures:

KBXMLDescription (top-level class)

XSDStructure (subclass of **KBXMLDescription**)

XSDSpecification (subclass of **XSDStructure**)

slots:

root_name: root token of the XML document (e.g. "guideline")

sections: specifies the "toplevel" classes whose instances are underneath the root of the XML document (this is where we specify that **Guideline**, **Recommendation_Set**, **Variable**, and **CEMMetaclass** are the classes whose instances we are going to expand

ClassXMLAnnotation (subclass of **XSDStructure**) Each instance of this class contains annotations on a class.

slots:

class: reference to the class being annotated

class_type_name: the name to use when declaring the XML schema complex type name for the class. Used to hide Protege-specific classes

parent_for_schema: parent of the complex type being declared (When there are multiple parents for a class, need to select one for the purpose of creating an XML schema)

selected_local_template_slots: local slots of the annotated class that should be used as subelements (Slots that are not inherited from *parent_for_schema*, but include slots from non-parent superclasses)

add_template_slot_constraints: if true, then generate directly overridden template slot facets

An example of **ClassXMLAnnotation** is shown in Figure 37.

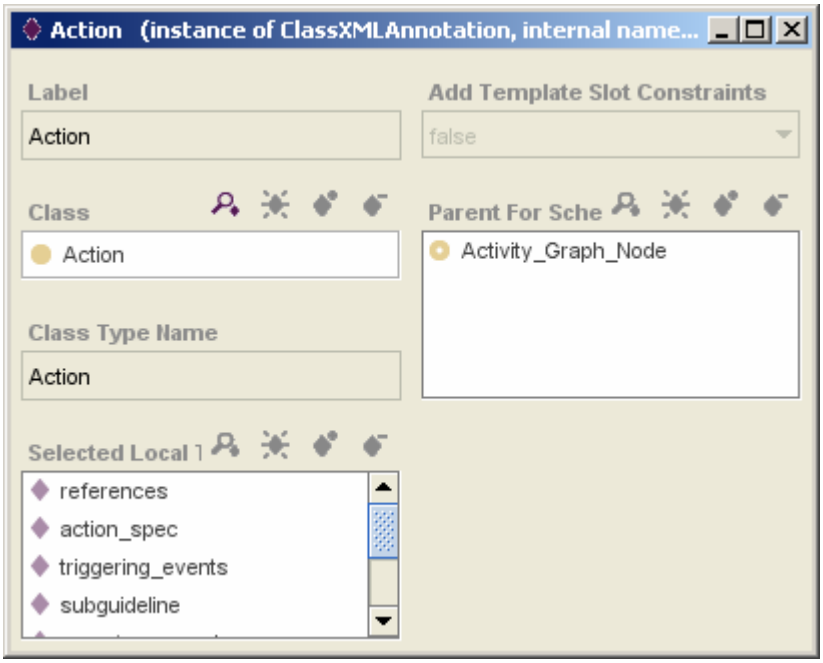


Figure 37. ClassXMLAnnotation for the class "Action."

SlotXMLAnnotation (subclass of **XSDStructure**): a class that allows annotations on a slot

slots:

slot_reference: reference to the slot being annotated

referencep: if true then values of the slot are referenced instead of expanded

SlotConstraintSpecification (subclass of **KBXMLDescription**): a class that specifies the structure of constraints added by a CEM (i.e. facet overrides)

slots:

slot_name: name of slot where a constraint is being placed.

slot_constraint_name: the name of constraint (In Protege term, the name of a facet of a slot)

slot_constraint_values: values of the constraint (i.e. facet values)

4.10.1.2.2 Algorithm for generating XML Schema and XML export

To generate the XML schema, the Python script takes the *root_name* and sections in an instance of **XSDSpecification** to generate the element hierarchy of the schema. Slots of the top-level classes are subelements of the element hierarchy.

After declaring the element hierarchy structure, the Python script goes through all instances of **ClassXMLAnnotation** to generate a complex type for each instance.

The XML-export script uses the *root_name* and sections in the **XSDSpecification** instance to generate the element hierarchy, expanding all instances of the classes specified in the "sections" slot of **XSDSpecification** instance.

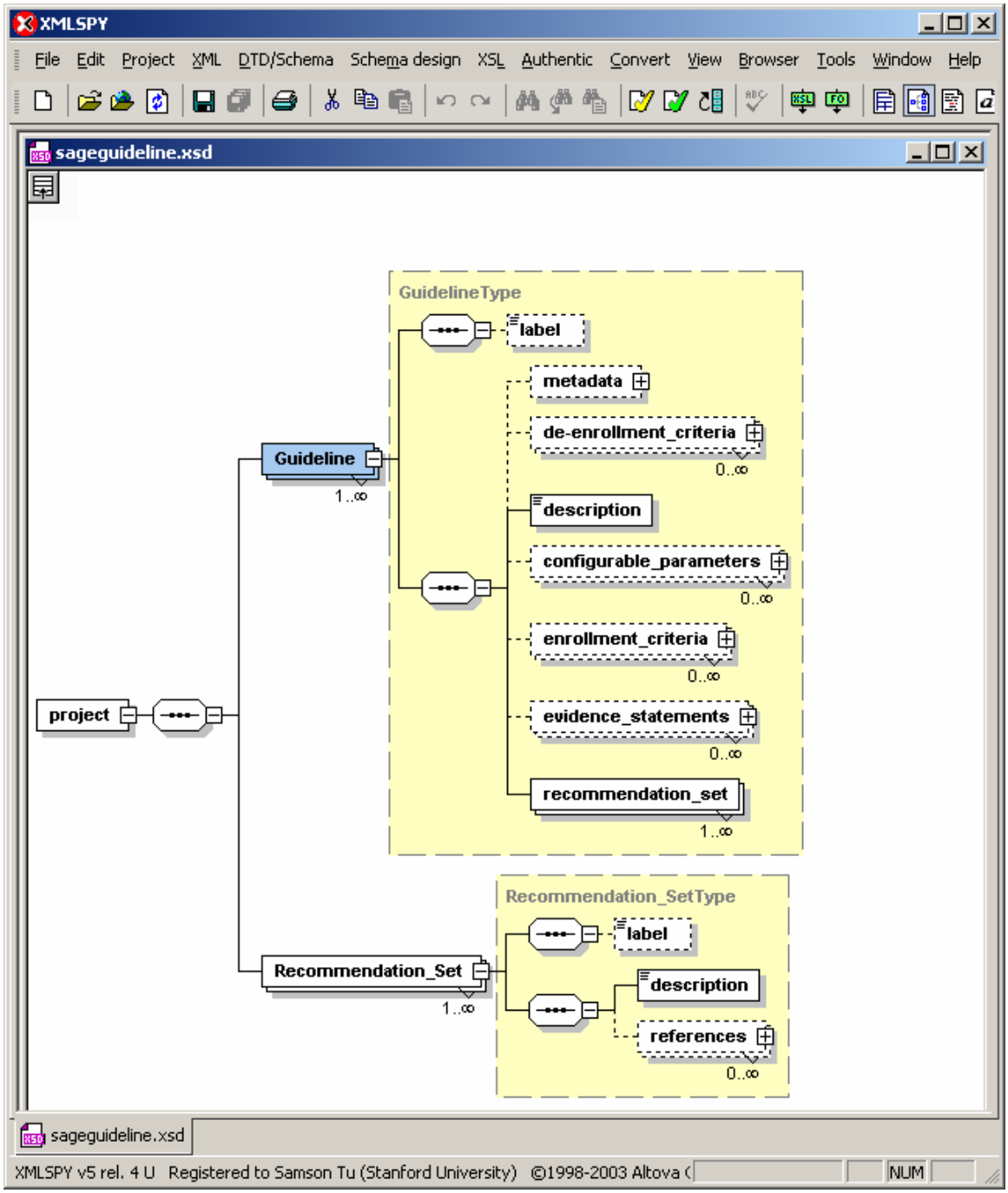


Figure 38. An XMLSpy view of a simplified view of XML Schema for the SAGE Guideline Model.

4.10.2 XML instance

In generating the XML instance from the Protégé knowledge base, I arbitrarily changed all occurrences of "<" in the knowledge base to "LessThan". Ditto with ">."

Instances of non-terminological classes that are not part of Guideline Model are represented as instances of the most specific class in the Guideline Model. These non-Guideline Model classes

are classes that represent constraints on slots in the Guideline Model. All CEMs fall into this category, so are instances of specialized data type classes (e.g. **BloodPressure** subclass of **PhysicalQuantity** data type).

5 Information Models

The healthcare entities that comprise the environment in which a clinical guideline operates are formalized in the SAGE guideline model. We categorize them as Health Organization Model or Patient Data Model.

5.1 Healthcare Organization Model

The SAGE project models the healthcare organization in which a guideline is implemented only to the extent that's necessary for the specification of an executable guideline. Thus, the settings where care takes place and the roles of patients and care providers have been relegated to terminology set. The current model contains an enumeration of clinical and information technology resources and an event model that defines the structure of the events that may trigger the SAGE execution engine.

5.1.1 Resources

See http://smi-web.stanford.edu/projects/sage/modelclasseshtml/index.html for a listing of the resources.

5.1.2 Event Model

The SAGE Guideline Model represents an executable encoding of a guideline. The execution of the guideline is triggered by events in the CIS or by events generated by SAGE itself. We model two types of events: Clinical/administrative events and **TimeDrive** events

5.1.2.1 Clinical/administrative event

A SAGE clinical or administrative event has the form "an agent in some *role* (e.g., Family Medicine Physician) performs an *act* on an *object* (e.g., accessing a resource such as patient medical record) in a *setting* (e.g., Outpatient clinical setting)"

To maintain backward compatibility, we define **Clinical_Event** and **Administrative Event** as separate classes that have identical structure and that have a *SAGE_event_type* slot that points to the hierarchy of event classes. So the instance of **Clinical_Event** (where Outpatient family medicine physician accesses patient record) still points to the class representing the same event.

Nevertheless, the definition of triggering events in **Recommendation_Set_Entities** had to change from class type to instance type. That's one incompatibility between the current version of the Guideline Model and execution engine.

Because we keep the reference to event classes, the execution engine can easily adapt to the new model by accessing the *SAGE_event_type* slot of the new event instances to get the old event class.

5.1.2.2 TimeDrive event

A **TimeDrive_Event** is an event that is generated purely on the basis of passage of time. There are different **TimeDrive** events.

A **Periodic_Time_Event** has a periodic interval specification using the HL7 data type **Period Interval of Time**.

The HL7 **Period Interval of Time** is defined by three attributes:

- period: a quantity of time (e.g., 2 weeks) that specifies the distance between any two time intervals.

- phase: an interval of time (i.e., an interval defined by low and high time-point limits) that specifies the duration and anchor of the interval in a calendar. The way HL7 specifies a time interval is to find an example of the interval to be specified. Thus, encoding a zero-length periodic interval of time “Sunday midnight every 2 weeks” would involve (1) specifying “every 2 weeks” using the period attribute , and then specifying zero-length interval “Sunday midnight” by finding an example of such time interval (e.g., 200601010000;200601010000).

-alignment: is a calendar code (CY, MY, WY, DM etc.) to which periodic intervals are aligned. For example, [20060102; 20060102] aligned to CY means January 02 every year.

See {HL7 V3 standard}/infrastructure/datatypes/datatypes.htm#domain-CalendarCycle for a complete list of calendar codes.

A **Relative_Periodic_Time_Event** is a **Periodic_Time_Event** that occurs periodically after an *anchor* event.

The *event_times* slot of the **Fixed_Time_Event** class specifies a set of **PointInTime** instances when the events should be generated.

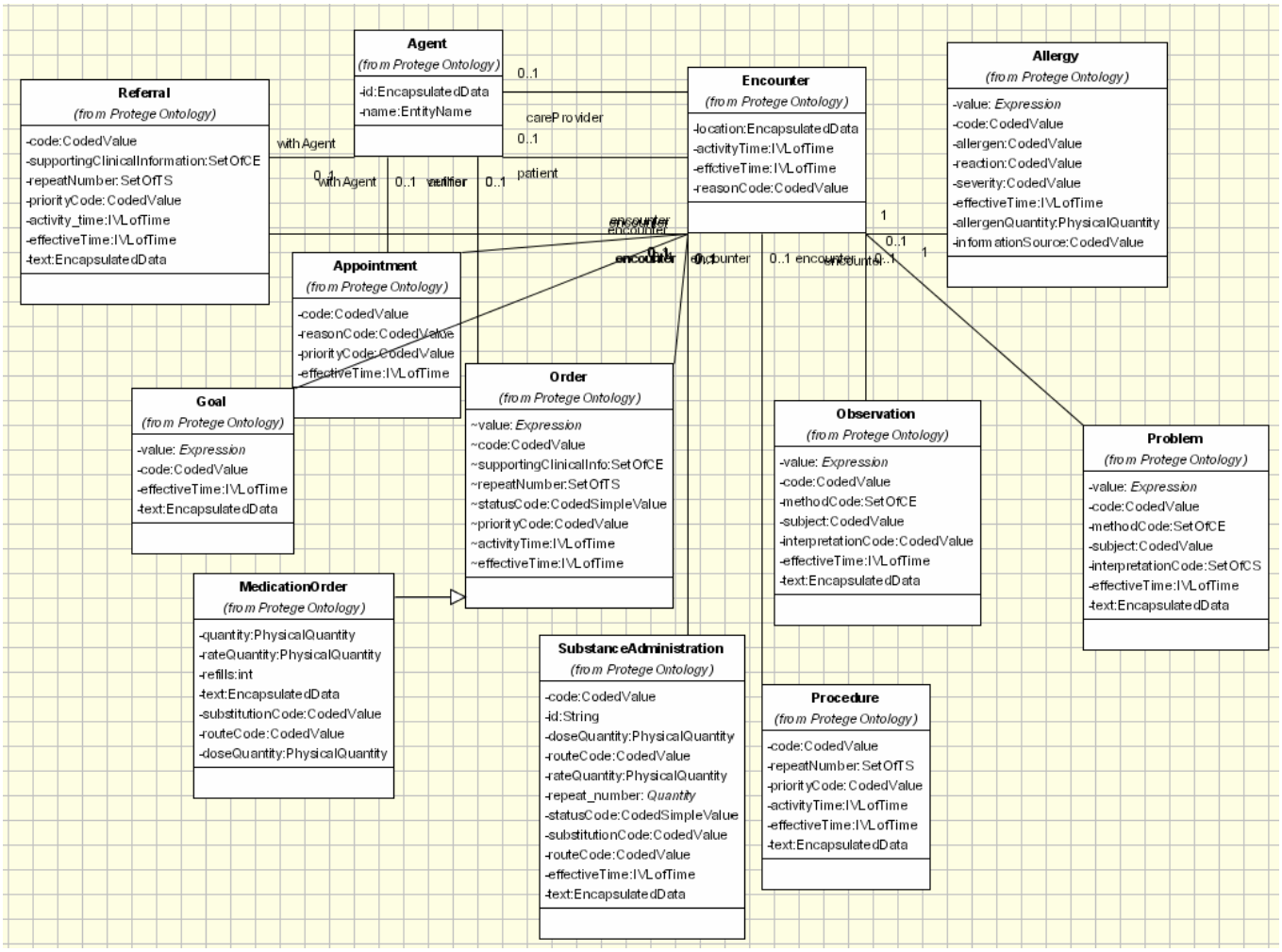
An instance of **Relative_Time_Event** class is an event that occurs at a point in time relative to another event. The *anchor* slot defines the point of reference. The *duration* slot defines the duration of time that should pass before this event is generated.

5.2 Patient Data Models

The SAGE Guideline Model is designed to deliver patient-specific decision support for guideline-based care. For that purpose, it must use patient information and deliver its recommendations in structured format. In SAGE, the format of patient information is defined through a Virtual Medical Record and Clinical Expression Models. The format of patient-specific recommendation output, as defined by subclasses of **Action_Specification** varies. The format include string messages (**Notify**), operations on VMR classes (**Conclude**, **Retract**, **Recommend_VMROrder**, and **Inquire**), and operations on order sets (**Recommend_OrderSet**). The **Display** action outputs the result of evaluating an expression, instances of VMR classes or **Supplemental_Materials**, and what we call Clinical Data Set. In this section, we briefly describe the VMR and Clinical Data Set as examples of patient data models.

5.2.1 Virtual Medical Record (VMR)

The Virtual Medical Record is described in a separate document (****Ref****). In this document we just give an UML diagram for illustrative purpose.



5.2.2 Clinical Expression Models

Given a Virtual Medical Record such as the one shown in Section 5.2.1, there are still many possible variations in represent a particular clinical statement. For example, “Presence of diabetes” can be represented as an Observation whose code is “Diabetes,” or an Observation whose code is “Diabetes” and value is “present.” A *clinical expression model* in SAGE is designed to remove the ambiguity by stating explicitly how patient data should be represented. It does so by adding constraints to a VMR class. For example, a clinical expression model for “Anaphylactic reaction to hepatitis B vaccine” would say that such data will be modeled as instances of **AdverseReaction** class where the *reaction* slot is constrained to be a concept subsumed by “anaphylactic reaction” and the *substance* slot is constrained to be a kind of hepatitis B vaccine.

5.2.3 Clinical DataSet

SAGE project investigated the use of flowsheet in presenting guideline-based recommendations. Several types of clinical data can be displayed in time-trended fashion, e.g. vital signs, assessment, lab tests and medication. For each data item, flowsheet application requires information to be able to initiate lab or medication order, set user input priority and display SAGE concluded recommendations on each item. Creating a central structure for such information would facilitate integration of SAGE DSS with CIS that has flowsheet applications. .

We experimented with a new information model called **Clinical DataSet** in order to define flowsheet-oriented data items. An instance of **Clinical DataSet** holds a collection of **Clinical_DataElement** instances as shown in Figure 39

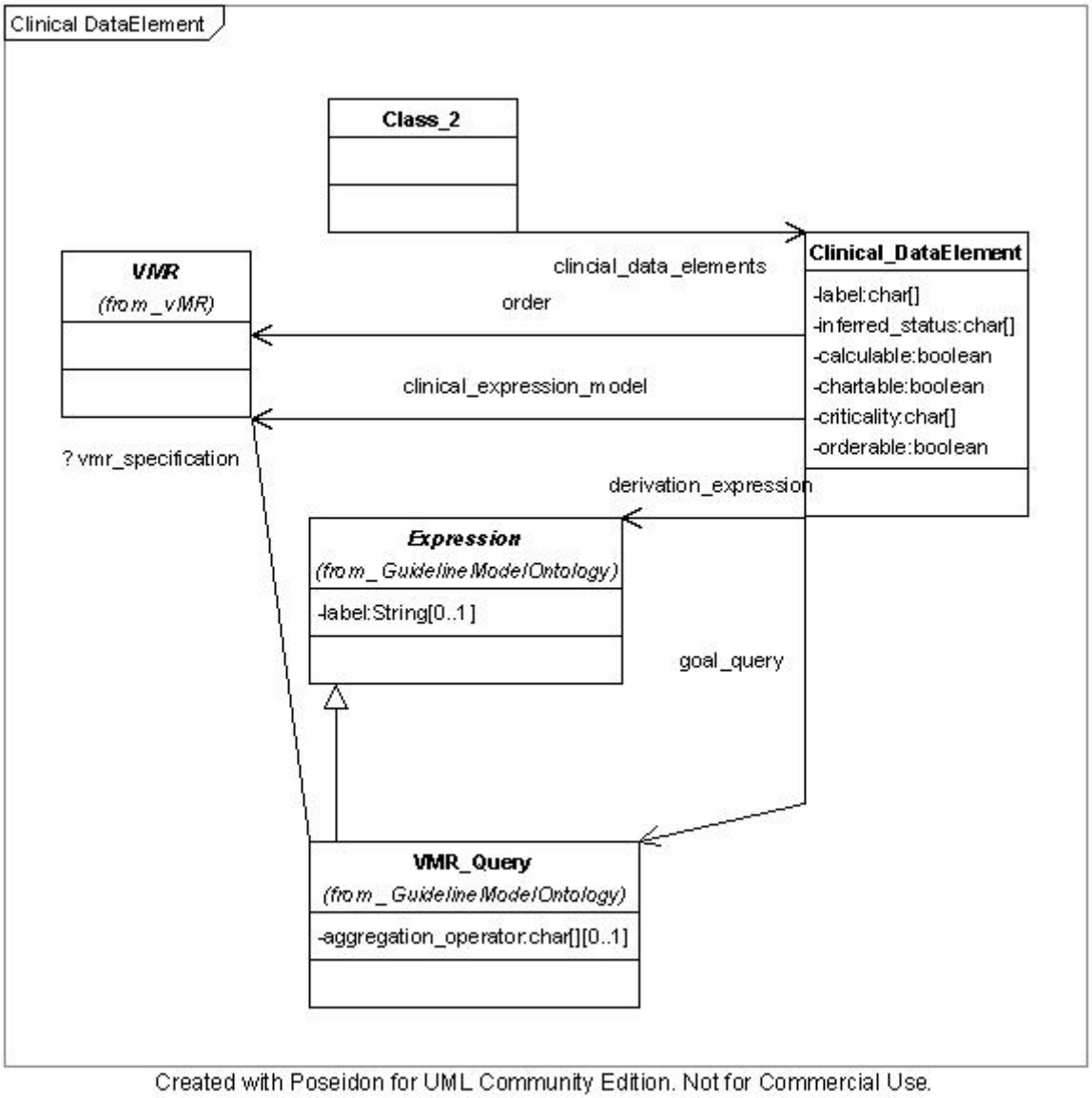


Figure 39 UML diagram showing Clinical DataElement

Figure 40 shows an example of Clinical DataElement for systolic blood pressure.

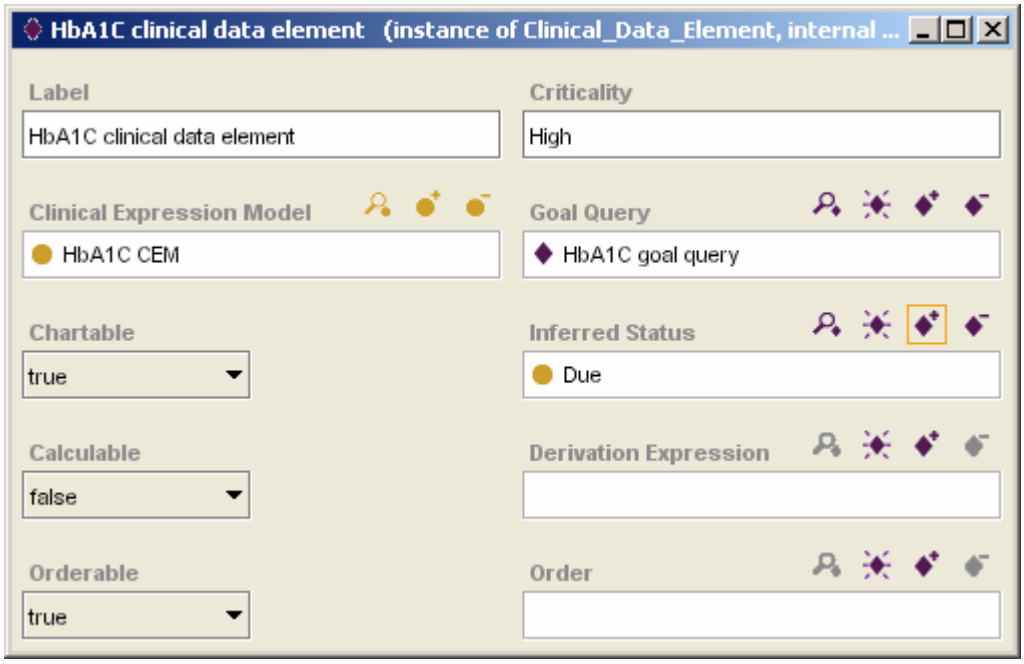


Figure 40 An example Clinical DataElement

Figure 41 shows the use of clinical data elements in Flowsheet.

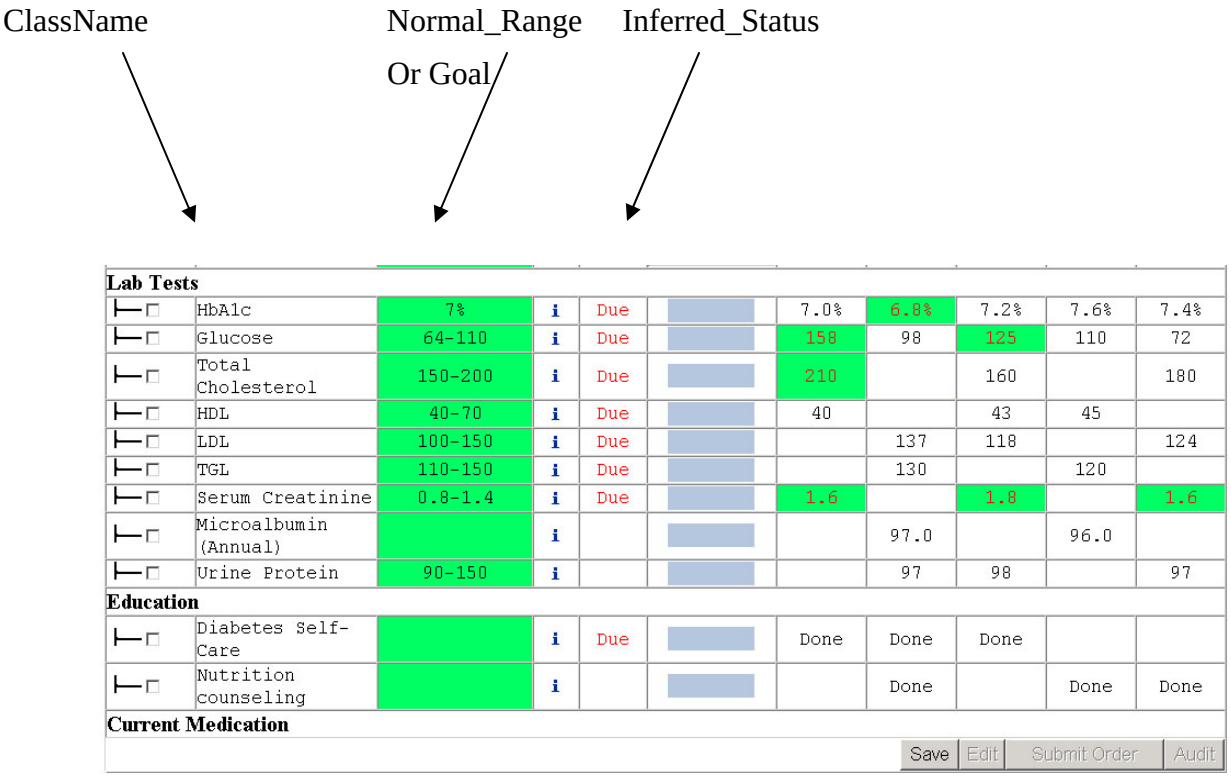


Figure 41 Example of the use of flowsheet to display guideline recommendations

Appendix: Class Hierarchy for SAGE Guideline Model (with links to detailed specification of each class)

An HTML representation of the SAGE Guideline Model class hierarchy (with links to detailed specification of each class in javadoc-like format) can be found at (****Ref****)guidelinemodelhtml\index.html.

Acknowledgement

The SAGE project was supported in part by grant 70NANB1H3049 of the NIST Advanced Technology Program.

6 References

American Diabetes Association (2002). Standards of Medical Care for Patients With Diabetes Mellitus. *Diabetes Care* **25(Supplement 1)**: S33-S49.

Boxwala, A., Tu, S. W., et al. (2001). Towards a Representation Format for Sharable Clinical Guidelines. *J Biomed Inform* **34**(3): 157-169.

Boxwala, A. A., Peleg, M., et al. (2004). GLIF3: A Representation Format for Sharable Computer-Interpretable Clinical Practice Guidelines. *J Biomed Inform* **37**(3): 147-161.

Carter, J., Brown, S., et al. (2002). Initializing the VA medication reference terminology using UMLS metathesaurus co-occurrences. Proc AMIA Symp, 116-120.

Field, M. J. and Lohr, K. N., Eds. (1990). Clinical Practice Guidelines: Directions for a New Program. Washington, DC, National Academy Press.

Fieschi, M., Dufour, J., et al. (2003). Medical decision support systems: old dilemmas and new paradigms? *Methods Inf Med* **42**(3): 190-198.

Fox, J. and Das, S. K. (2000). Safe and Sound. Cambridge, MIT Press.

Grieve, G. and Schadow, G. (2003). UML Implementation Technology Specification - Data Types. www.hl7.org/v3ballot/html/foundationdocuments/itsum/datatypes-its-uml.htm, accessed 2004.

Health Level 7 (2003). HL 7 Reference Information Model. http://www.hl7.org/library/data-model/RIM/modelpage_non.htm, accessed 2003.

Hripcsak, G., Ludemann, P., et al. (1994). Rationale for the Arden Syntax. *Comput Biomed Res* **27**(4): 291-324.

Institute for Clinical Systems Improvement (2002). Health Care Guideline: Immunization. http://www.icsi.org/knowledge/detail.asp?catID=29&itemID=174, accessed 2002.

Johnson, P. D., Tu, S. W., et al. (2000). Using Scenarios in Chronic Disease Management Guidelines for Primary Care. Proc AMIA Symp, Los Angeles, USA, 389-393.

Johnson, P. D., Tu, S. W., et al. (2001a). Achieving Reuse of Computable Guideline Systems. Medinfo, London, UK, 99-1003.

Johnson, P. D., Tu, S. W., et al. (2001b). A Virtual Medical Record for Guideline-Based Decision Support. Proc AMIA Symp, Washington, DC, 294-298.

Kiepuszewski, B., ter Hofstede, A. H. M., et al. (2003). Fundamentals of Control Flow in Workflows. *Acta Informatica* **39**(3): 143-209.

Krall, M. A. and Sittig, D. (2002). Clinician's assessments of outpatient electronic medical record alert and reminder usability and usefulness requirements. Proc AMIA Symp, 400-404.

Object Management Group (2003). UML 2.0 OCL Specification. http://www.omg.org/cgi-bin/doc?ptc/03-10-14.pdf, accessed 2004.

Peleg, M., Boxwala, A., et al. (2000). GLIF3: The Evolution of a Guideline Representation Format. Proc AMIA Symp, Los Angeles, 645-649.

Peleg, M., Tu, S. W., et al. (2003). Comparing Computer-Interpretable Guideline Models: A Case-Study Approach. *J Am Med Inform Assoc* **10**(1): 52-68.

Peleg, M., Boxwala, A. A., et al. (2004). The InterMed Approach to Sharable Computer-Interpretable Guidelines: A Review. *J Am Med Inform Assoc* **11**(1): 1-10.

Pryor, T. and Hripcsak, G. (1993). Sharing MLM's: An Experiment between Columbia-Presbyterian and LDS Hospital. Proc Annu Symp Comput Appl Med Care, 399-403.

Quaglini, S., Stefanelli, M., et al. (2000). Guideline-Based Careflow Systems. *Artif Intell Med* **5**(22): 5-22.

Shahar, Y., Young, O., et al. (2003). DEGEL: A hybrid, multiple-ontology framework for specification and retrieval of clinical guidelines. AIME, Protaras, Cyprus.

Shankar, R., Tu, S. W., et al. (2002). Use of Protégé-2000 to Encode Clinical Guidelines. Stanford University. SMI-2002-0944.

Shankar, R. D., Tu, S. W., et al. (2003). A Knowledge-Acquisition Wizard to Encode Guidelines. Proc AMIA Symp, Washington DC, 1007.

Shiffman, R. (1994). Toward effective implementation of a pediatric asthma guideline: integration of decision support and clinical workflow. Proc Annu Symp Comput Appl Med Care, Washington, DC, 797-801.

Shiffman, R., Shekelle, P., et al. (2003). Standardized reporting of clinical practice guidelines: A proposal from the Conference on Guideline Standardization. *Ann Intern Med* **139**: 493-498.

Shiffman, R. N., Karras, B. T., et al. (2000). GEM: A Proposal for a More Comprehensive Guideline Document Model Using XML. *J Am Med Inform Assoc* **7**: 488-498.

Tu, S. W. and Musen, M. A. (1999). A Flexible Approach to Guideline Modeling. Proc AMIA Symp, Washington DC, Hanley & Belfus, Inc., 420-424.

Tu, S. W., Musen, M. A., et al. (2004). Modeling Guidelines for Integration into Clinical Workflow. Medinfo, San Francisco, CA, USA, 174-178.

W3C (2002). Web Services Activity. http://www.w3.org/2002/ws, accessed 2003.

Wang, A. Y., Sable, J. H., et al. (2002a). The SNOMED clinical terms development process: refinement and analysis of content. Proc AMIA Symp. 2002, 845-849.

Wang, D., Tu, S. W., et al. (2002b). Representation Primitives, Process Models and Patient Data in Computer-Interpretable Clinical Practice Guidelines: A Literature Review of Guideline Representation Models. *Int J Med Inf* **68**(1-3): 59-70.

Workflow Management Coalition (1999). Interface 1: Process Definition Interchange, Process Model. Workflow Management Coalition. WfMC TC-1010-P.