

The EON Guideline Model

Contact information of primary author:

Samson W. Tu

swt@stanford.edu

Stanford Medical Informatics

Stanford University

MSOB X259, 251 Campus Drive

Stanford, CA 94305-5479

Acknowledgements

This document was extracted and adapted from the ATHENA DSS Guide [1], Samson Tu was the primary author of the extracted sections. However, the co-authors of the ATHENA DSS Guide—Robert W. Coleman and Drs. Susana Martins and Mary K. Goldstein—made significant contributions to these sections.

The EON project was funded by NLM Grant LM05708. Dr. Mark Musen was the principal investigator. Project members included Aneel Advani, Amar Das, John Nguyen, Martin O'Connor, Yuval Shahar, Ravi Shankar, and Samson Tu.

Protégé was developed by Stanford Medical Informatics at the Stanford University School of Medicine with support from the Defense Advanced Research Projects Agency (DARPA), the National Cancer Institute (NCI), the National Institute of Standards and Technology (NIST), the National Library of Medicine (NLM), and the National Science Foundation (NSF). Protégé is a national biotechnology resource supported by grant P41 LM007885 from the National Library of Medicine.

The ATHENA Project has been funded in part by Department of Veterans Affairs (VA) Health Services Research and Development (HSR&D) grants CPG 97-006 “Guidelines for Drug Therapy of Hypertension: ‘Closing the Loop’” (1998 – 2000, Principal Investigators, Mary K. Goldstein, MD, and Brian B. Hoffman) and CPI 99-275 “Guidelines for Drug Therapy of Hypertension: Multi-Site Implementation” (2000 – 2005, Principal Investigators, Goldstein and Hoffman). Participating sites for the implementation and evaluation in the second grant phases were VA Palo Alto, VA Durham (site Principal Investigator, Eugene Oddone, MD), and VA San Francisco (site Principal Investigator: Michael Shlipak, MD). Project members have included Robert Coleman, MS Pharm; Susana Martins, MD; and additional participants from VA medical centers in Palo Alto, San Francisco, and Durham. Additional details about the overall projects are available in the Final Reports for grants CPG 97-006 and CPI 99-275, available on request from Dr. Goldstein.

The authors would like to thank Ms. Lesley Evensen for editorial assistance in the early stages of development of the manual, and Ms. Heidi Craig for editorial assistance in the later stages. Views expressed in this document are those of the authors and not necessarily those of the Department of Veterans Affairs or other funding agencies or affiliated organizations.

Acknowledgements	2
I. Introduction	4
II. Protégé	4
II.1. Protégé Knowledge Model	4
II.1.1. Instances, Slots, and Classes	5
II.2. Metaclass in Protégé	7
II.3. Constraints in Protégé: Facets and PAL Constraints	10
II.4. Protégé's File Format	11
II.5. Protégé User Interface	11
II.5.1. Components of the Protégé UI	12
II.6. Two Common Tasks	21
III. EON Models	25
III.1. Patient Data Model	27
III.2. Medical Concept Model	29
III.2.1. Terminology Hierarchies	32
III.2.2. Value_Type Hierarchy	34
III.2.3. Associations	36
III.2.4. Supporting Material	37
III.3. EON Guideline Model	38
III.3.1. Management Guideline	39
III.3.2. Clinical Algorithm	42
III.3.3. Action Specification	51
III.3.4. Drug Usage and Guideline Drug Activities	61
III.4. Expressions	66
III.4.1. Template-based Expression Language	67
III.4.2. PAL-based Expression Language	74
III.4.3. Summary	76
IV. Guideline Interpreter's Use of the Knowledge Base	76
IV.1. Overview	77
IV.2. Structure of Guideline Advisories	80
IV.3. Generation of Guideline Advisories	84

I. Introduction

This document describes the EON guideline model, especially as it was implemented and used in the ATHENA project.

Section II gives a short introduction to Protégé. Because the ATHENA Knowledge Base was developed using Protégé 1.7 and earlier versions, the figures use screen dumps from Protégé 1.7. Between Protégé 1.7 and the current version of Protégé,¹ its user interface (UI) has changed substantially, although the basic operations are the same. The major difference between Protégé 1.7 and the current version is that the *Diagram widget* described in Section II.5.1.6 has been replaced with the *Graph widget*. Please see Protégé documentation for details of the Graph widget.

Section III describes the models and expression languages used in EON. Section IV discusses how the Guideline Interpreter (aka EON Guideline Execution Engine) uses the guideline model to generate patient-specific recommendations.

For an introduction to the process of encoding guidelines and protocols in EON model, see the presentations and workbooks of a Guideline Modeling Workshop conducted at Stanford University on March 23-24, 2006. The workshop was funded by an academic expert supplement grant obtained by The Health Services Research and Development (HSR&D) Center for Health Care Evaluation (CHCE). The workshop website is located at http://www.chce.research.med.va.gov/athena/guideline_modeling_workshop.htm. The guideline used in the workshop can be found at <http://www.chce.research.med.va.gov/athena/atglance.pdf>. The self-contained software used in the workshop can be downloaded from <http://www.stanford.edu/~swt/ATP3Demo.zip>.

II. Protégé

Protégé [2, 3] is an integrated software tool used by system developers and domain experts to develop knowledge-based systems. Created at Stanford Medical Informatics, Stanford University, Protégé is the tool used to develop the ATHENA Knowledge Base. Extensive documentation for Protégé is available on the Protégé website: <http://protege.stanford.edu>.

The following subsections give a basic introduction to Protégé. We will discuss Protégé's knowledge model [4], the integrity constraints that one can place in the knowledge base to make sure that it is well-formed, Protégé's file format, and, finally, its graphical user interface. More details can be found at: http://protege.stanford.edu/doc/users_guide/index.html.

II.1. Protégé Knowledge Model

¹ As of November 30, 2006, Protégé 3.1.1 is the current version.

II.1.1. Instances, Slots, and Classes

In Protégé, the basic unit of knowledge representation is a *frame*, a structure that holds specific types of information. Frames can be used to hold information about concrete individuals in the domain (e.g., *George Washington* or *JNC 6 guideline*), in which case, the frames are *instance frames* (or, more simply, *instances*). Frames can also be used to represent a named collection of individuals (e.g., *American Presidents*) or an abstract concept (e.g., the concept of a beta adrenergic antagonist drug), in which case, the frames are *classes*. Properties of instances—e.g., *eligibility_criteria* (shown as Eligibility Criteria in Figure 1),² which define the target population of a guideline—are themselves frames called *slots*. Slots may have *values*. Slot values may be drawn from one of Protégé’s primitive types—float, integer, string, symbol, or Boolean—or they may be instances or classes.³

² For each slot name displayed on the instance form, Protégé automatically replaces the underscore with a blank and capitalizes the first letter of each word. Thus, the slot *eligibility_criteria* is displayed as “Eligibility Criteria” on the instance form of the *ATHENA_Management_Guideline* class.

³ In the rest of the document, we may use *attribute* synonymously with *slot*. They both represent properties or characteristics of an entity.

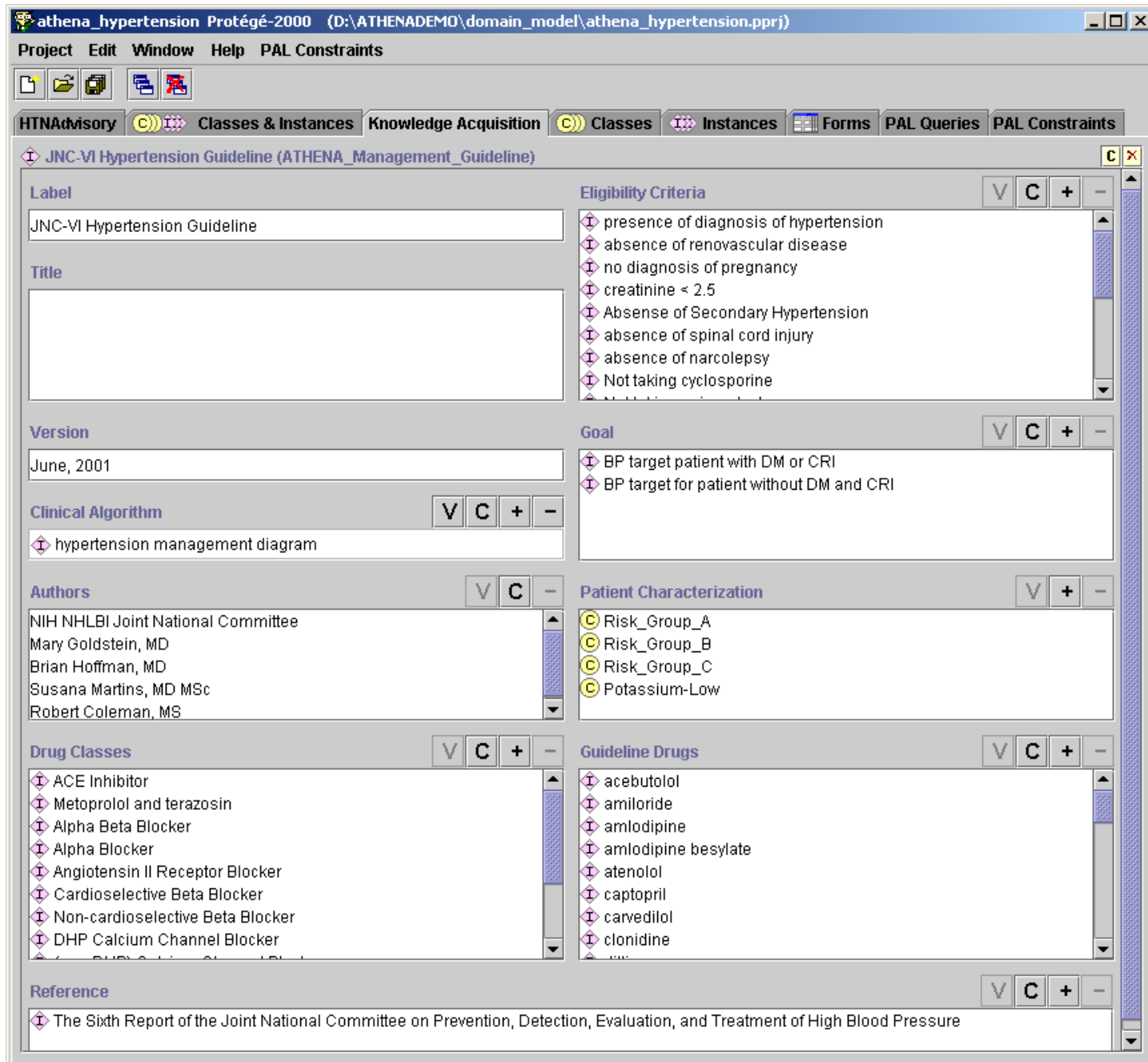


Figure 1 - Partial view of the JNC-VI Hypertension Guideline instance

In Protégé, classes are organized into classification hierarchies where children classes are specializations of parent classes. Furthermore, *template slots* associated with a class define the properties that instances of the class may have. Figure 2 shows the definition of the ATHENA_Management_Guideline class in Protégé. The left pane shows the class hierarchy of concepts in the ATHENA Knowledge Base. The middle pane shows instances (JNC-VI Hypertension Guideline) of the selected class (ATHENA_Management_Guideline), and the right

pane shows the template slots associated with the selected class. Note that slots such as `patient_characterization`, `eligibility_criteria`, `authors`, `version`, and `goal` are precisely the slots that have values in Figure 1.

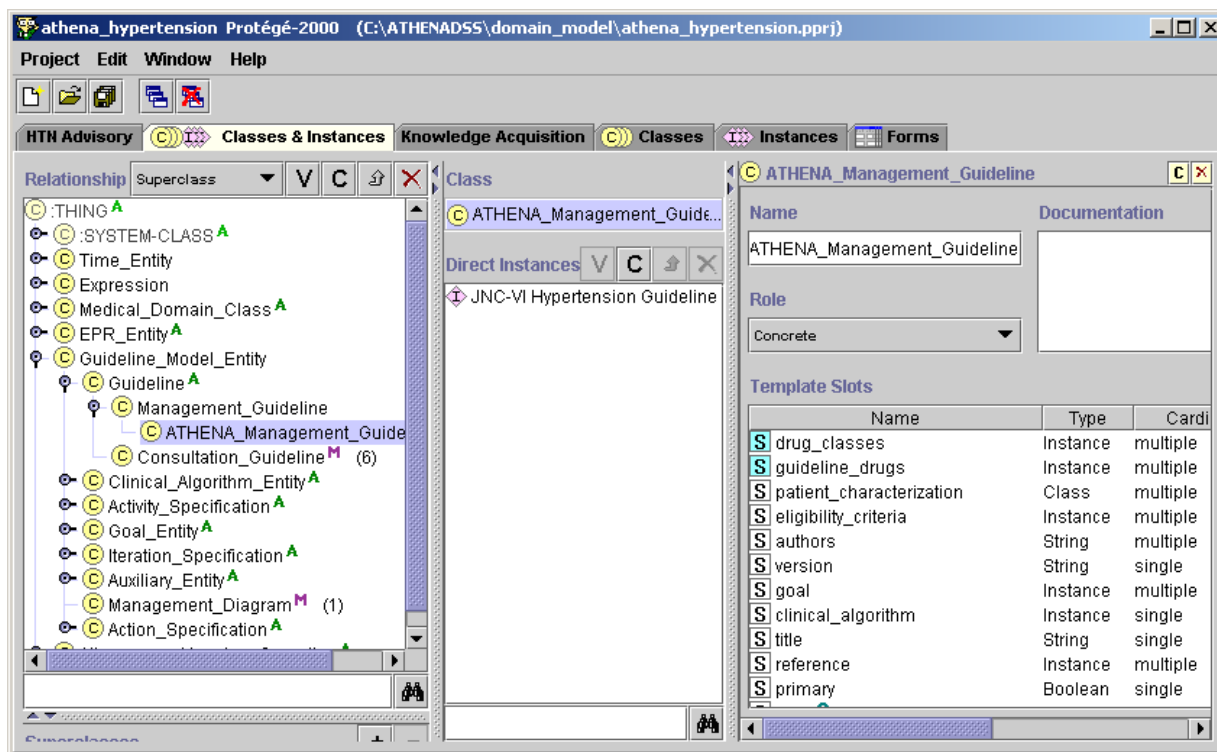


Figure 2 - Protégé graphical user interface showing the class hierarchy and the definition of a class

II.2. Metaclass in Protégé

In Protégé, classes are themselves instances of other classes. Classes whose instances are classes are called *metaclasses*. Protégé defines a metaclass called `:STANDARD-CLASS` as the default metaclass for all classes. Users can define their own metaclasses by subclassing `:STANDARD-CLASS`. Figure 3 shows some of the metaclasses that have been defined for the EON/ATHENA system. This subsection will describe the reasons for users to define metaclasses. Details of the individual metaclasses will be discussed in Subsection III.2.

There are two reasons for users of Protégé to define their own metaclasses:

1. Users can associate slots with a metaclass, as with regular classes. In Figure 4, the `Diagnostic_Term_Metaclass` has a `DiagnosticCriteria` slot. Thus, instances of `Diagnostic_Term_Metaclass`—e.g., the CRI (Renal Insufficiency) class (Figure 5)—may have values for the `DiagnosticCriteria` slot. The Guideline Interpreter would evaluate the `DiagnosticCriteria` slot value of CRI to determine whether a patient has renal insufficiency.

- Another reason to have metaclasses is that, by making a collection of classes instances of a metaclass, special reasoning can be carried out for the collection. Thus, for example, by making drug concepts (such as lisinopril and atenolol) instances of the Medication_Metaclass, we can write decision criteria that range over these drug concepts (i.e., instances of Medication_Metaclass). This makes it possible to determine, say, whether a bad drug partner has already been prescribed for a possible drug recommendation.

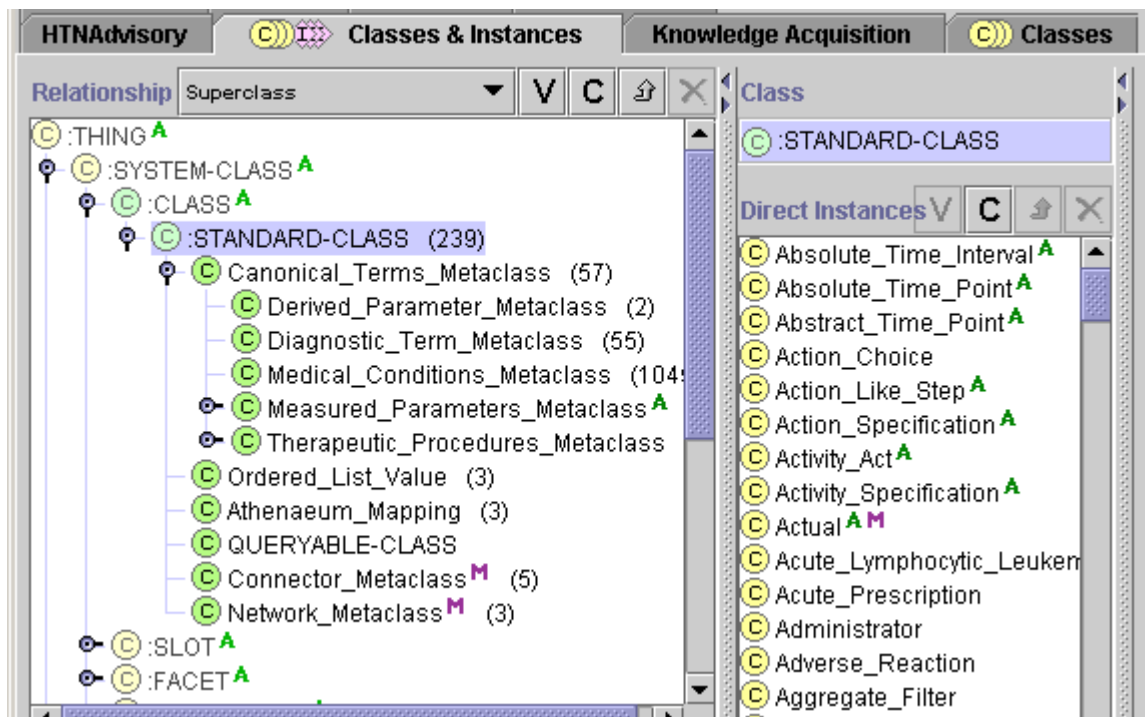


Figure 3 - The metaclass hierarchy in Protégé, showing classes (such as Absolute_Time_Interval) as instances of the :STANDARD-CLASS metaclass

The screenshot shows a window titled "Diagnostic_Term_Metaclass". It has three main sections: "Name", "Documentation", and "Constraints". The "Name" section contains a text field with "Diagnostic_Term_Metaclass". The "Role" section has a dropdown menu set to "Concrete". The "Template Slots" section is a table with the following data:

Name	Type	Cardinality	Other F
S DiagnosticCriteria	Instance	single	classes={Criterion}
S PrettyName	String	single	
S Synonyms	String	multiple	
S :NAME	String	single	
S :DOCUMENTATION	String	multiple	
S :ROLE	Symbol	single	allowed_values=/&h

Figure 4 - The definition of Diagnostic_Term_Metaclass. DiagnosticCriteria is a slot whose value is specific to instances of this metaclass.

The screenshot shows a window titled "CRI (Diagnostic_Term_Metaclass)". It has sections for "Name", "Documentation", "Role", "PrettyName", "Synonyms", and "DiagnosticCriteria". The "Name" section contains a text field with "CRI". The "Role" section has a dropdown menu set to "Concrete". The "PrettyName" section contains a text field with "Renal Insufficiency". The "Synonyms" section is empty. The "DiagnosticCriteria" section contains a list with one item, "CRI", which is preceded by a diamond icon.

Figure 5 - The definition of the CRI class (an instance of the Diagnostic_Term_Metaclass). It has a PrettyName and a criterion defining Renal Insufficiency (in the DiagnosticCriteria slot) in terms of other data.

II.3. Constraints in Protégé: Facets and PAL Constraints

In Protégé, slots are themselves frames that have properties. The properties of a slot, called *facets*, express constraints on possible values of slots. Figure 6 shows the facets of the `eligibility_criteria` slot as they appear in Protégé's UI. We can see that:

- values of the slot are constrained to be of the value type, Instance;
- the values must be instances of the allowed class, Criterion; and
- the slot may have zero or more values, since the *Cardinality multiple* checkbox is checked and there is no entry in the *Cardinality at least* box.

The Minimum and Maximum facets apply only to slots whose value types are integers or floating-point numbers. The Template Values of a slot are values inherited by all instances of the classes to which the slot is attached. (For example, if we associate a criterion as a template value of the eligibility criteria slot in the `Management_Guideline` class, then all guidelines—be they hypertension or screening guidelines—will have that criterion as one of their eligibility criteria.) The Default value of a slot is the value that a slot is initially given when an instance is created. The creator of the instance is free to change the initially assigned value. The Inverse Slot of a slot is one that has a reciprocal relationship with the slot (e.g., parent-of and child-of are inverses of each other: if John is the parent of Jim, then Jim is a child of John).

Figure 6 - The facets of the `eligibility_criteria` slot

Facets of a slot express constraints on a single slot. When a constraint is a complex relationship among multiple instances and slots, it can be written as a *Protégé Axiom Language (PAL) constraint* (Figure 7). A PAL constraint uses a logic-based language to express relationships among instances of a Protégé knowledge base that can span across multiple classes and slots.

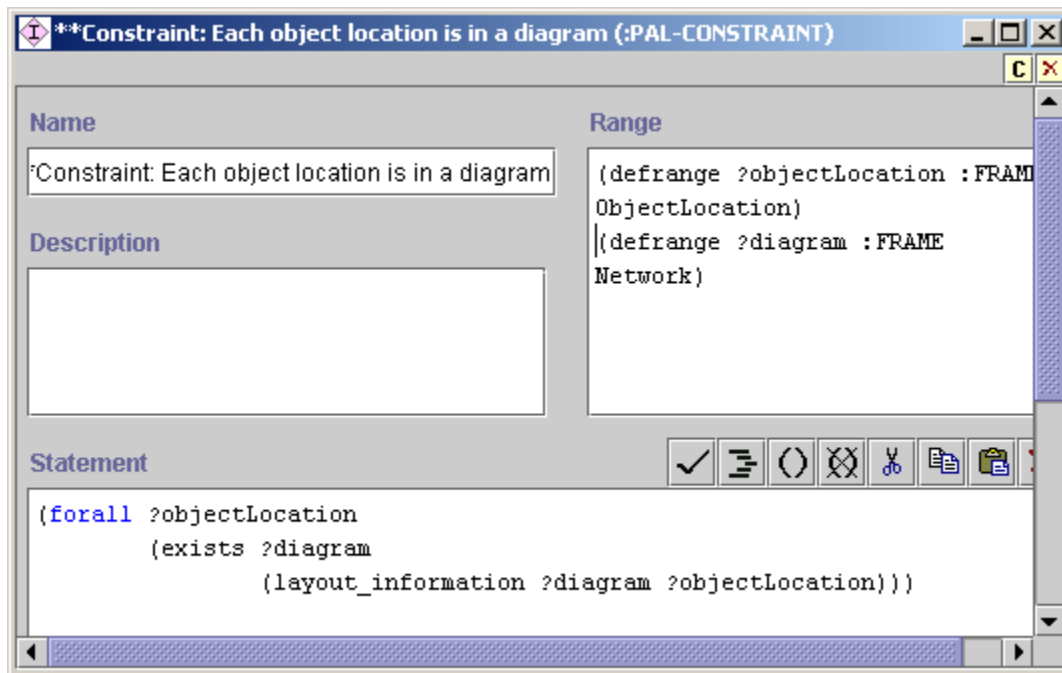


Figure 7 - A PAL constraint expressing the requirement that all instances of ObjectLocation must be part of a diagram, i.e., instances of the Network class

II.4. Protégé's File Format

Protégé saves a knowledge base—called a *project* in Protégé—as a collection of three files. They have the extensions .pont, .pins, and .pprj. The pont file contains the definition of Protégé classes. The pins file contains the definition of and data about instances. The pprj file contains display information for classes and instances as well as information about relationships among different Protégé projects. The same pont and pins files may be used in different pprj files if different displays are desired for the same knowledge base. Thus, in ATHENA, a clinician may view and edit the ATHENA Knowledge Base using athena_hypertension.pprj, which references athena_all.pont and athena_all.pins and which has numerous UI customizations. The Guideline Interpreter, on the other hand—because it doesn't need the display information kept in the athena_hypertension.pprj file—loads the athena_server.pprj project, which also references the same athena_all.pont and athena_all.pins files.

II.5. Protégé User Interface

This subsection introduces the Protégé user interface, describing its components and highlighting some of the common operations. It also gives two examples of working with the Protégé GUI:

1. searching for classes or instances whose display name matches a string, and
2. finding all places where a class or instance is referenced.

More information about using Protégé can be found at:
http://protégé.stanford.edu/doc/users_guide/index.html.

II.5.1. Components of the Protégé UI

The Protégé UI is divided into tabs, each of which provides a view into the content of the knowledge base. This subsection describes how to work with the tabs—called Forms, Classes, Instances, Classes & Instances, and Knowledge Acquisition—occasionally referencing the Protégé User Guide.

II.5.1.1. Forms Tab

The basic UI paradigm in Protégé is that information associated with a frame (see Subsection II.1) are displayed on a *form*. For each class, Protégé generates a prototypical form used to display and edit instances of that class. The Forms tab allows a developer to customize this automatically generated instance form of a class. Figure 8 shows the Forms tab for the ATHENA Knowledge Base. The left-hand side of the tab shows the class hierarchy of the knowledge base. Selecting a class in this hierarchy (e.g., the Scenario class) causes the prototypical form associated with the class to be displayed on the right side of the tab. Each slot of the class is shown with a default *slot widget*, a UI component that displays the slot value and through which the slot value can be edited. For example, in Figure 8, the *label* slot is shown as a text box with the title “Label” above it, the Boolean *new_encounter* slot is displayed as a check box entitled “New Encounter”. By default, slots that have instances as values are displayed with slot widgets that allow a user to view (**V** button), create (**C** button), add (**+** button), or remove (**-** button) instances as slot values.

The prototypical form associated with a class can be customized. By selecting a slot widget on the prototypical form and using the Selected Widget Type drop-down menu, a developer can change the slot widget associated with a slot. Figure 9 shows that the *new_encounter* Boolean slot can be displayed using either ComboBoxWidget or CheckBoxWidget. The Form Browser Key drop-down menu allows a user to select the slot whose value will be used as the display name of an instance. Figure 10 shows that the *label* slot has been selected as the slot whose value is the display name of Scenario instances.

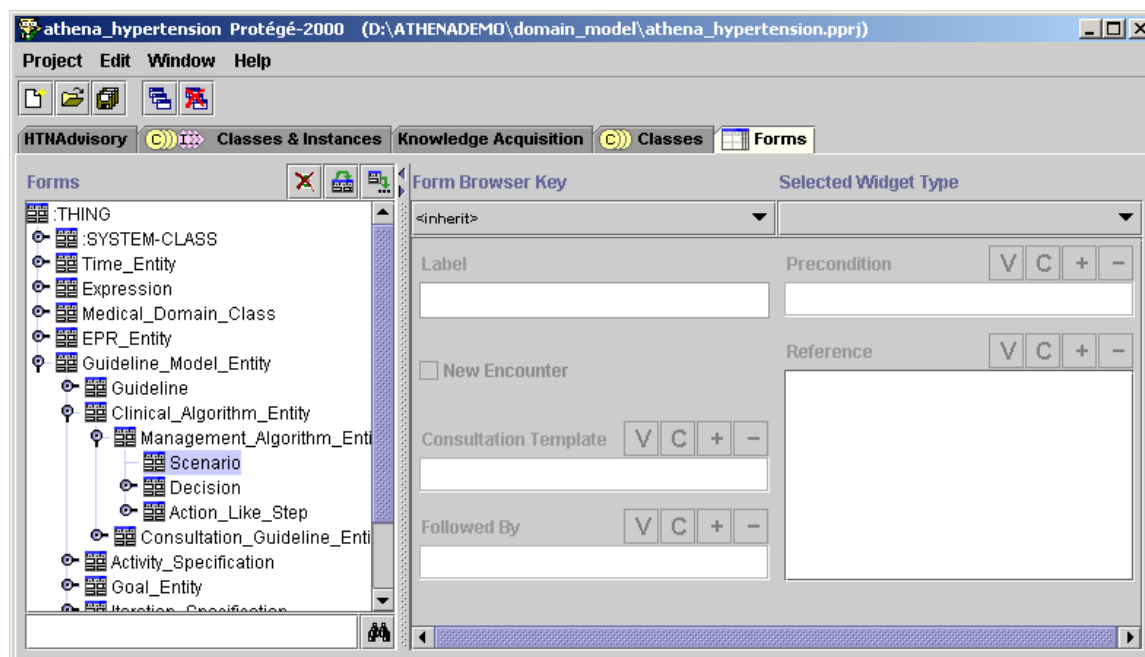


Figure 8 - The instance form associated with the Scenario class. It defines the UI for viewing and editing instances of the class.

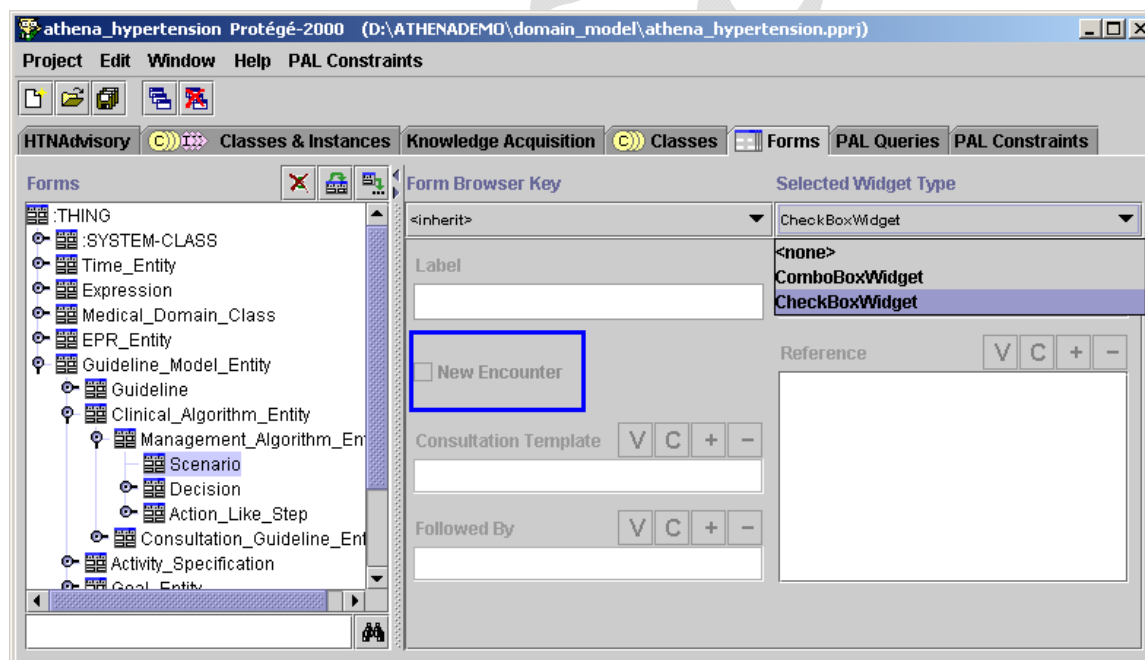


Figure 9 - Changing the slot widget assigned to the new_encounter slot

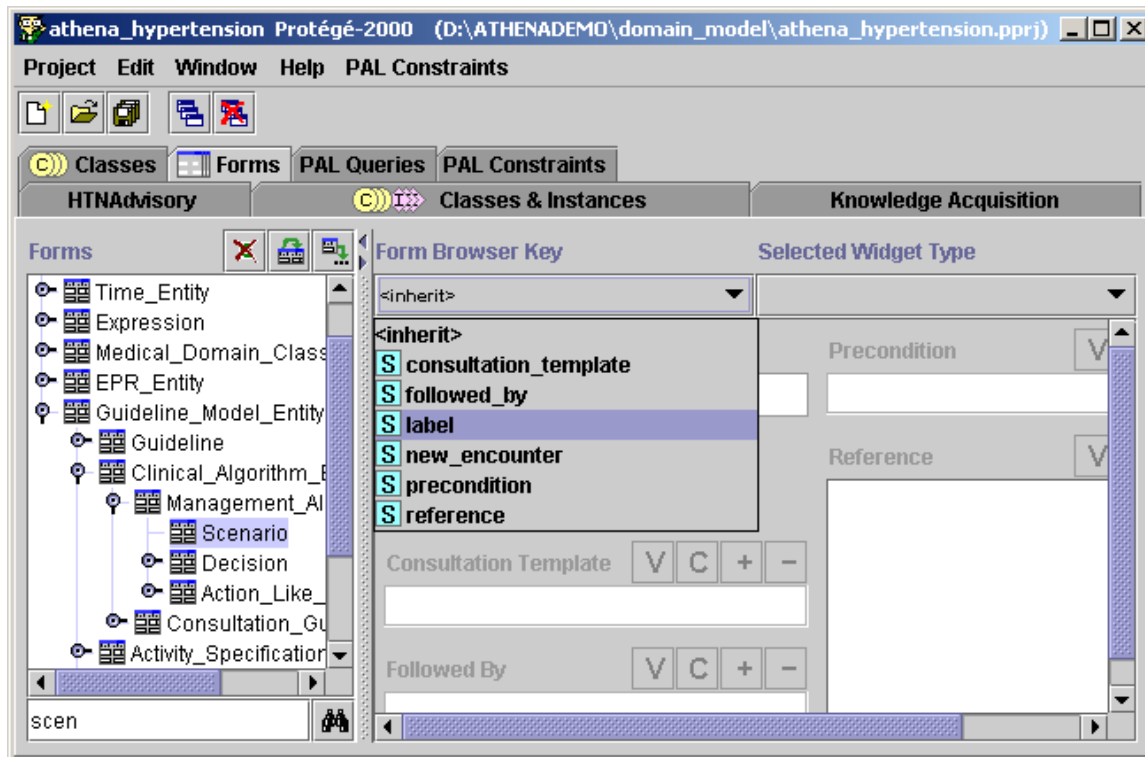



Figure 10 - Selecting the label slot to supply the display name of Scenario instances

II.5.1.2. *Classes Tab*

Quoting the Protégé User Guide:

“The Classes tab provides a single window in which you may view, create, and edit classes, which model concepts in your domain. An example is shown below (Figure 11). The window consists of three panes:

1. The **Class Relationship** pane in the upper left shows classes in a hierarchy and allows you to edit, create, and delete new classes. It also allows you to rearrange the class hierarchy by dragging a class to a replacement superclass.
2. The **Superclasses** pane in the lower left shows the superclasses of the selected class and allows you to add and remove superclasses for a class, as well as jump to a different superclass by clicking on it.
Note: If you cannot see the **Superclasses** Pane, your window may be too small. You can see the pane by enlarging your window or by dragging the slider bar at the bottom of the **Class Relationship** Pane.
3. When a single class is selected, the **Edit** pane on the right contains the Class Form for the selected class. The Class Form allows you to: name the class, choose its role, define constraints, provide a brief note, and, most importantly, define and edit the template slots. The Class Form can also be displayed as a separate window by clicking the View  icon in the Class Relationship pane.”

The Class Form can also be viewed by double clicking on a selected class under the Classes tab.

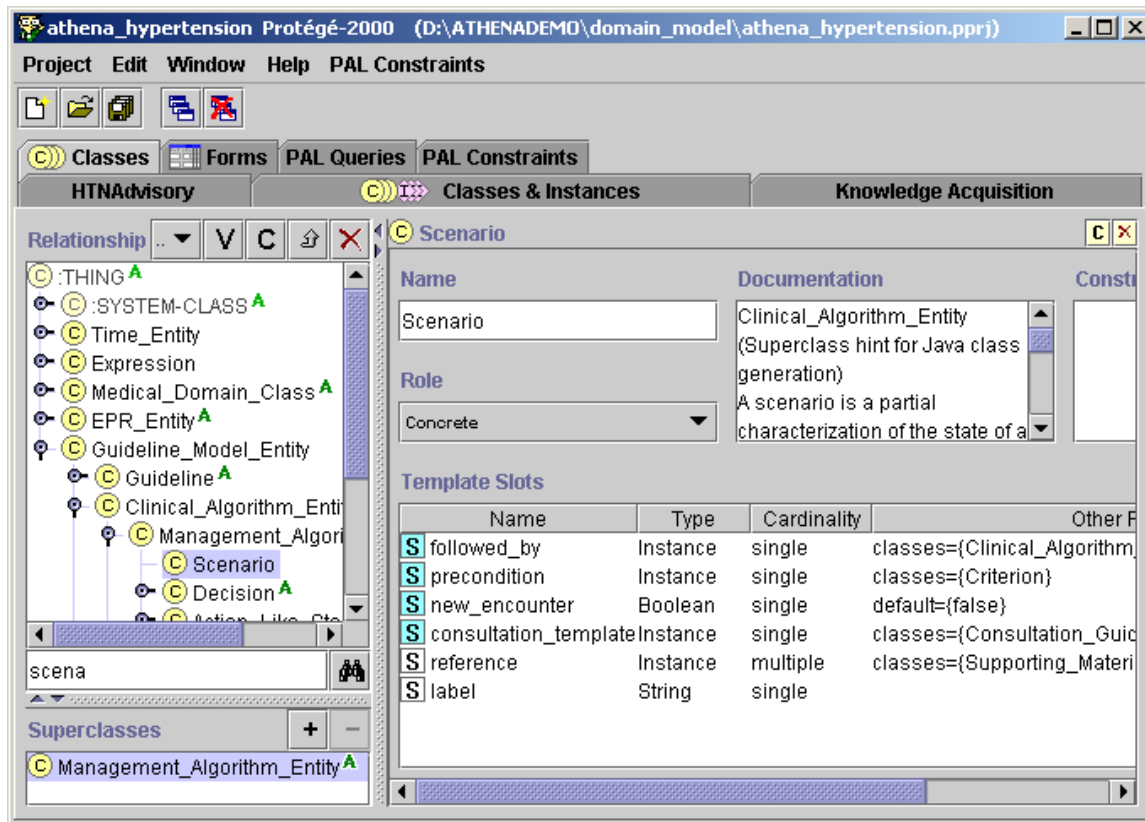


Figure 11 - The Classes tab in Protégé, showing the classes in the ATHENA Knowledge Base and the Class Form for editing the Scenario class

II.5.1.3. *Instances Tab*

Quoting the Protégé User Guide:

“The Instances tab provides a window in which you may view, create, and edit instances. (Classes model concepts in your domain, slots model properties of classes and any relationships between them, and instances model the actual data.) An example of the Instances Tab is shown below (Figure 12). The window consists of three panes:

1. A **Class pane** at the upper left shows the classes in a superclass/subclass relationship. The Instances Tab lets you view classes, but you cannot edit or rearrange them. ...
2. The **Direct Instances pane** in the center shows all the direct instances, if any, for the selected class, and allows you to view, edit, create, and delete direct instances.
3. When a single instance is selected, the Edit pane on the right contains the **Instance Form** for the selected instance. The Instance Form displays all the slots which apply to the instance, and allows you to edit them. The Instance Form can also be displayed as a separate window by clicking the View **V** icon in the Direct Instances pane.”

Double clicking on an instance in the Direct Instance pane under the Instances tab causes the Instance Form of that instance to pop up in a separate window. Elsewhere in this manual, an instance form is also called a “template”.

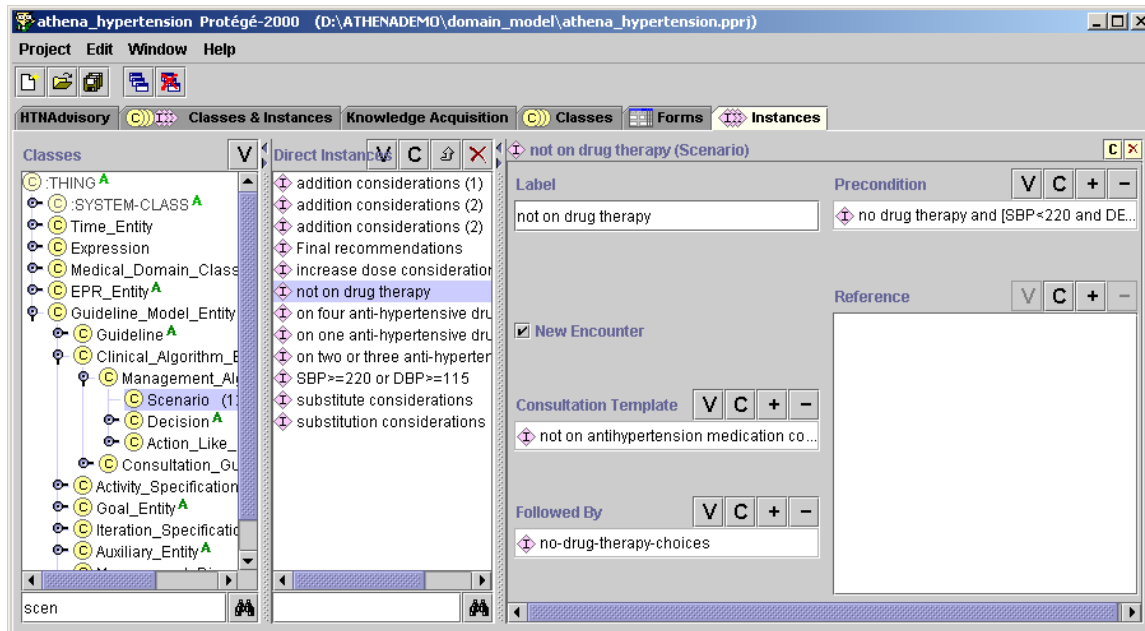


Figure 12 - The Instances tab, showing the Instance Form associated with the “not on drug therapy” instance of Scenario

II.5.1.4. Classes & Instances Tab

The Classes & Instances tab combines most of the functionalities of the Classes tab and the Instances tab in a single tab. Like the Instances tab, the Classes & Instances tab has three main panes: Class Relationship pane as it exists in the Class tab, Direct Instances pane, and the form for the selected class or instance. When a class is selected in the Class Relationship pane, but no instance is selected in the Direct Instances pane, then the form pane displays the class form associated with the selected class (Figure 13). When a class is selected in the Class Relationship pane and an instance is selected in the Direct Instances pane, then the form pane displays the Instance Form associated with the selected instance (Figure 14).

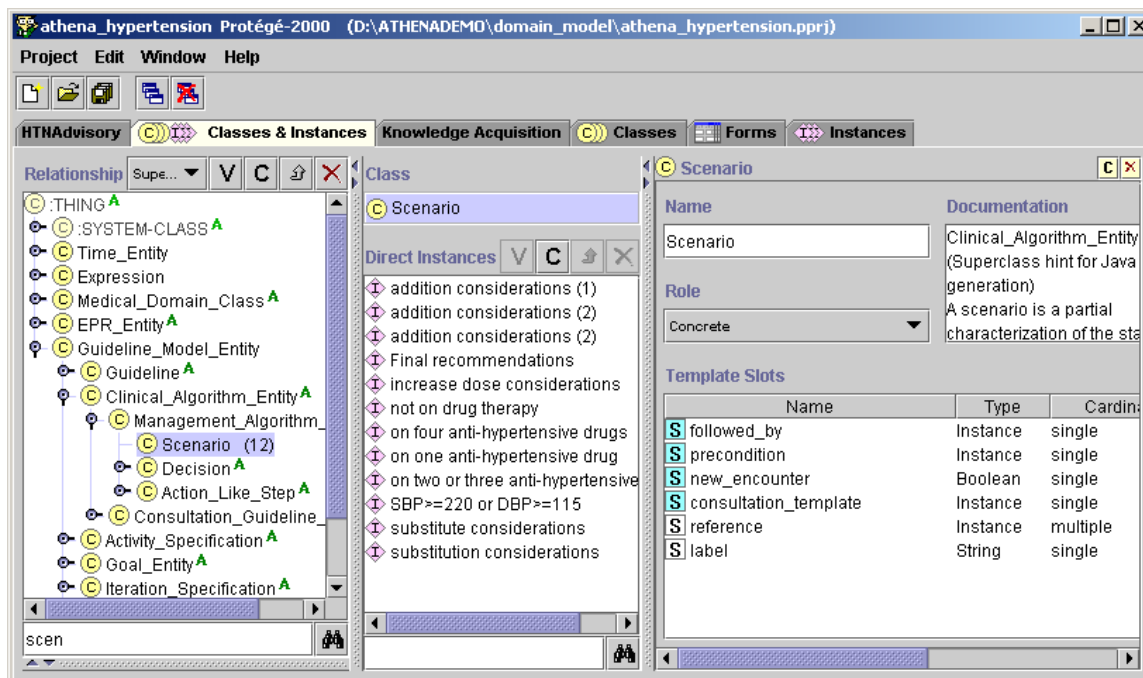


Figure 13 - The Classes & Instances tab. It shows a class hierarchy in the Relationship pane on the left, a Direct Instances pane in the center, and a Class Form for class Scenario on the right.

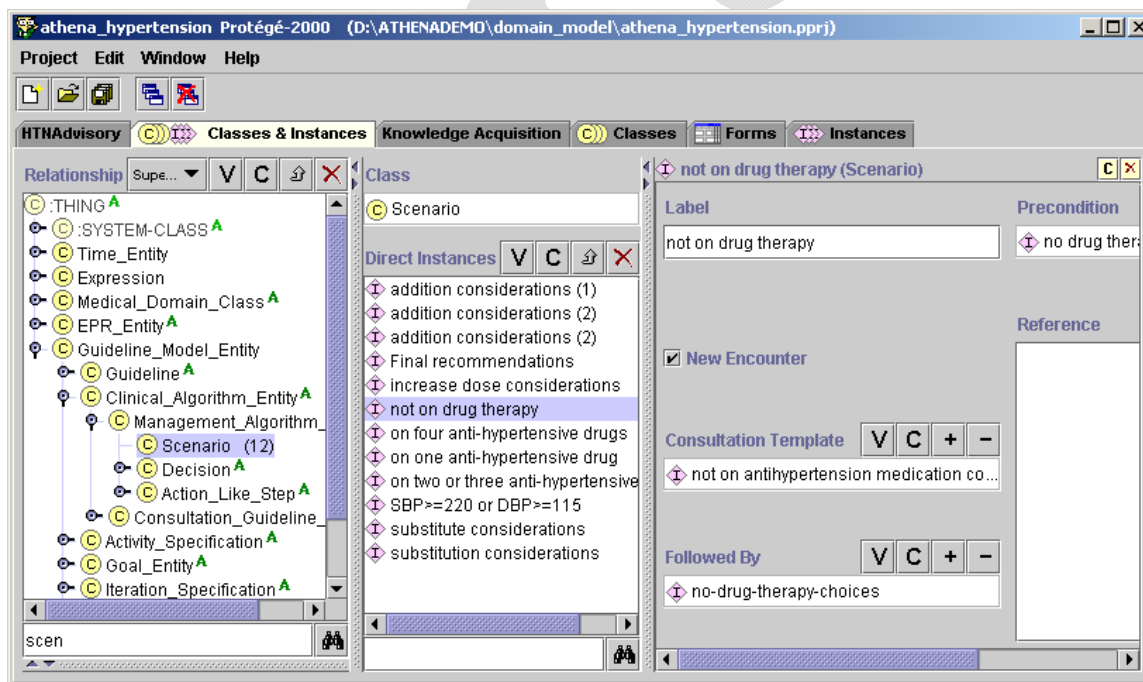


Figure 14 - The Classes & Instances tab, as in Figure 13, except that on the right it is displaying an Instance Form for the “not on drug therapy” instance

II.5.1.5. Knowledge Acquisition Tab

The Knowledge Acquisition tab is designed to facilitate navigation in a Protégé knowledge base by specifying an instance as the entry point for browsing the knowledge base. For the ATHENA Knowledge Base, the JNC-VI Hypertension Guideline instance of the ATHENA_Management_Guideline class is the starting point for browsing the ATHENA Knowledge Base (Figure 15).

The screenshot shows the Protégé-2000 interface with the 'Knowledge Acquisition' tab selected. The main window title is 'athena_hypertension Protégé-2000'. The 'Classes & Instances' tab is active, showing the 'JNC-VI Hypertension Guideline (ATHENA_Management_Guideline)' instance. The interface is divided into several sections:

- Label:** JNC-VI Hypertension Guideline
- Title:** (Empty text box)
- Version:** June, 2001
- Clinical Algorithm:** hypertension management diagram
- Authors:** NIH NHLBI Joint National Committee, Mary Goldstein, MD, Brian Hoffman, MD, Susana Martins, MD MSc, Robert Coleman, MS
- Drug Classes:** ACE Inhibitor, Metoprolol and terazosin, Alpha Beta Blocker, Alpha Blocker, Angiotensin II Receptor Blocker, Cardioselective Beta Blocker, Non-cardioselective Beta Blocker
- Eligibility Criteria:** presence of diagnosis of hypertension, absence of renovascular disease, no diagnosis of pregnancy, creatinine < 2.5, Absence of Secondary Hypertension, absence of spinal cord injury, absence of narcolepsy, Not taking cyclosporine
- Goal:** BP target patient with DM, CHF or CRI, BP target for patient without DM, CHF, and CRI
- Patient Characterization:** Risk_Group_A, Risk_Group_B, Risk_Group_C, Potassium-Low
- Guideline Drugs:** acebutolol, amiloride, amlodipine, amlodipine besylate, atenolol, captopril, carvedilol

Figure 15 - The Knowledge Acquisition tab, showing the Instance Form of the JNC-VI Hypertension Guideline instance of the ATHENA_Management_Guideline

The ATHENA_Management_Guideline class was designed to make most important entries in the ATHENA Knowledge Base visible on a single form.

II.5.1.6. Diagram Widget

To graphically display and edit the clinical algorithm associated with a guideline, the ATHENA Knowledge Base uses the diagram widget to represent the steps and connections among the steps (Figure 16). For the Management_Diagram class, the Steps slot is configured such that the associated diagram widget shows the allowed classes of the slot (Choice_Step, Action_Choice, Scenario, and Case_Step). They are available in the drawing palette of the diagram. For example, a user can create an instance of Scenario in the drawing canvas by selecting and dragging a Scenario icon from the drawing palette to the drawing canvas. Similarly, a connection between two icons (representing instances) can be created by selecting and dragging the connection arrows and attaching the ends of each arrow to the appropriate icons.⁴

⁴ The diagram widget described here works only in Protégé 1.9 or earlier. Since Protégé 2.0, a new graph widget has replaced the diagram widget.

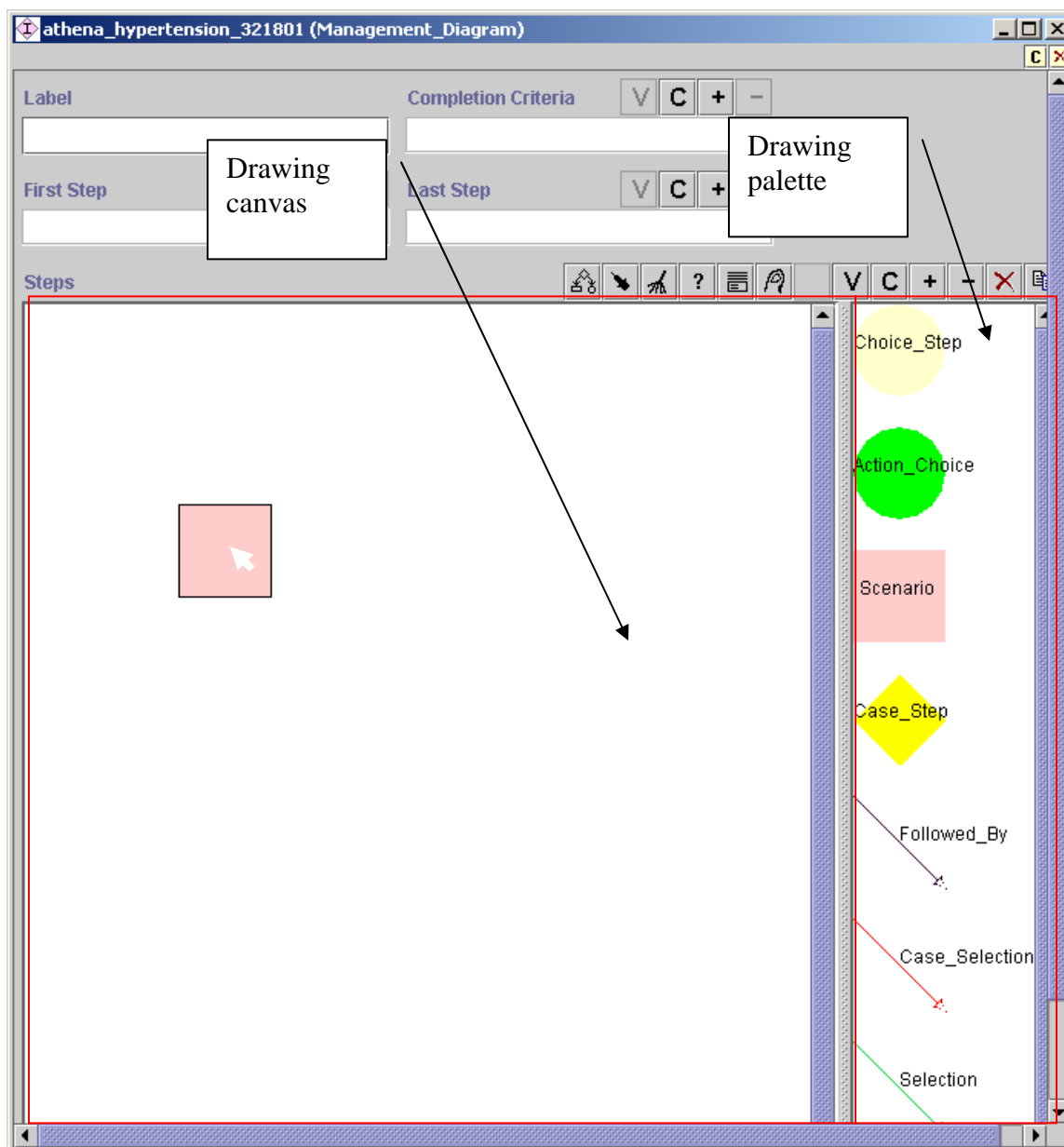


Figure 16 - The diagram interface for creating a clinical algorithm. A user creates an object in the diagram by selecting and dragging an icon from the drawing palette to the drawing canvas.

Figure 17 shows a portion of the main clinical algorithm of the ATHENA Knowledge Base.

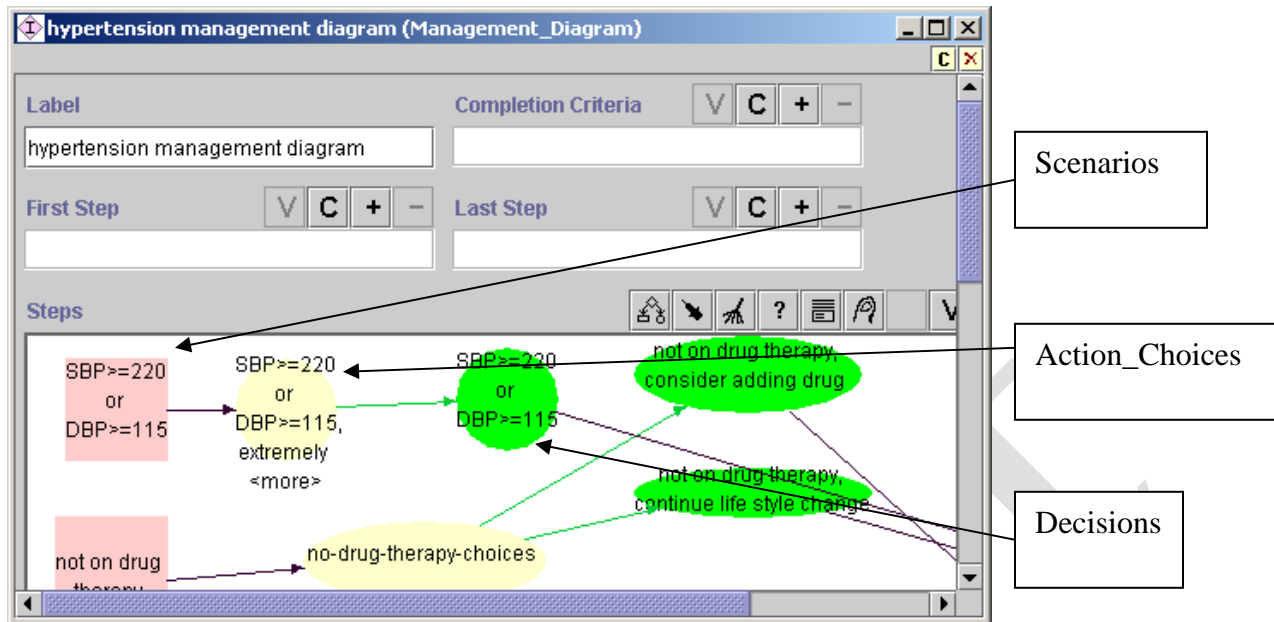


Figure 17 - Hypertension management diagram, showing Scenarios, Decisions (Choice_Steps), and Action_Choices

II.6. Two Common Tasks

The following subsections describe two tasks that a user will commonly carry out in Protégé. One is using the search function to find existing classes or instances. The other is finding references to classes, slots, or instances in one or more places throughout the ATHENA Knowledge Base.

II.6.1.1. Using the Search Function to Find Classes or Instances

In Protégé, a user can search for existing classes or instances by their display names. This is done using the binocular search pane available in a number of Classes & Instances panes and dialog boxes (Figure 18). The binocular search pane consists of the search-string text field and the binocular search button.

Suppose a user is looking for a class with a name that begins with “anti”. He or she can enter the search string “anti” in the search-string text field and click the binocular button (Figure 18). If there is one match, that class will be highlighted. If there is more than one match, Protégé will show a dialog box containing all matches (Figure 19), and the user will be able to select one. String comparisons in searches are *not* case sensitive.

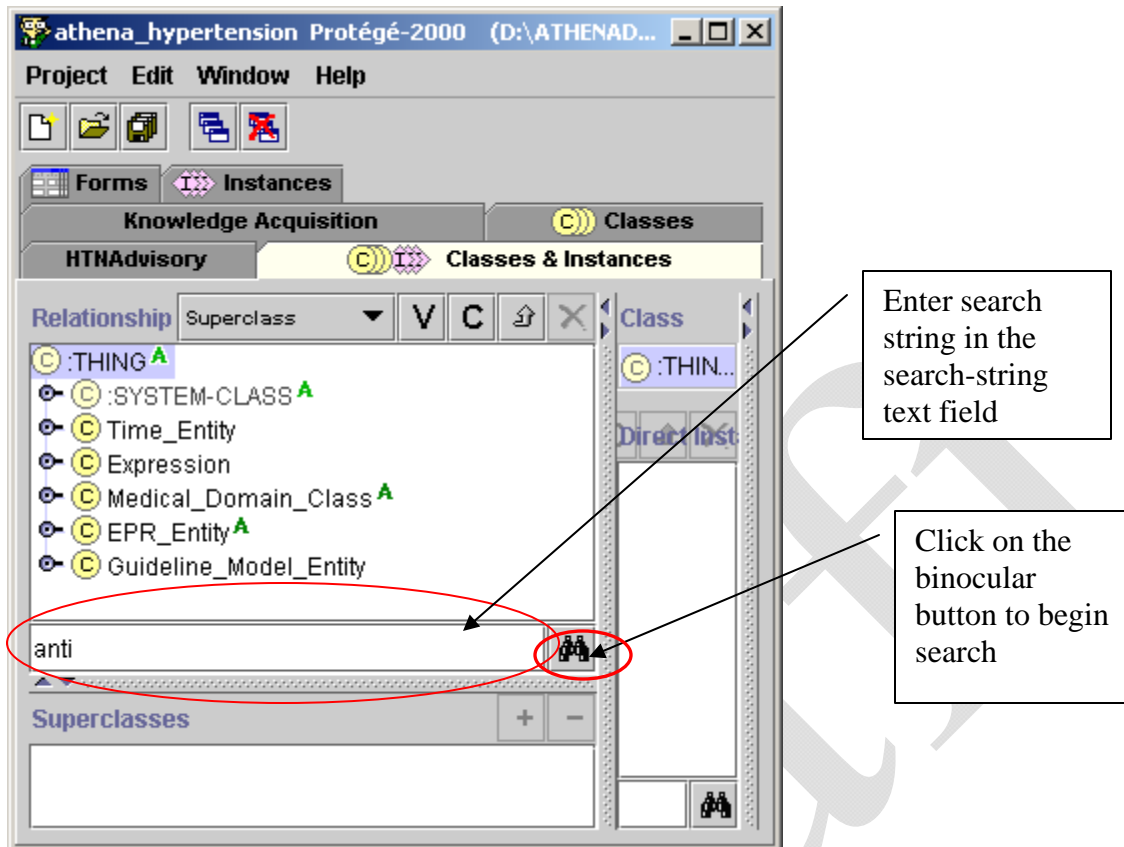


Figure 18 - Using the binocular search pane to look for classes with a name beginning in “anti”

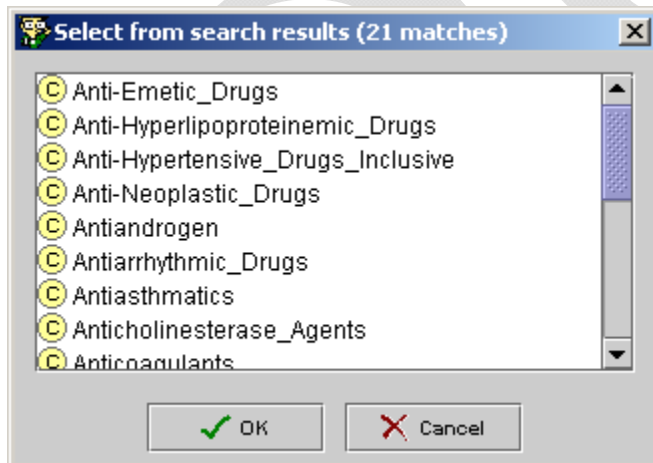


Figure 19 - Results for the search string “anti”

If you want to find classes with names containing “anti” in any position, you can add the wildcard character * at the beginning of the search string. Protégé will return all classes with names containing “anti” (Figure 20).

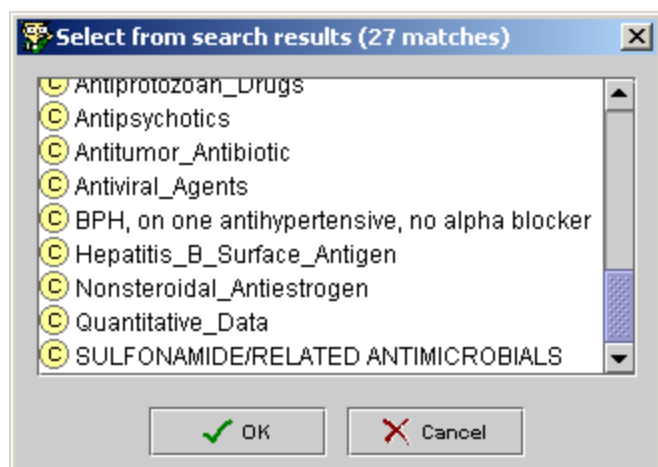


Figure 20 - Search results for the search string “*anti”

II.6.1.2. *Finding References to Classes, Slots, or Instances throughout the Knowledge Base*

A class or an instance may be used in one or more places in the knowledge base. If a user wants to modify or delete a class or an instance, it is essential to know all places this class or instance is used. He or she can use the references buttons, highlighted in Figure 21, to find all references to a particular class or instance.

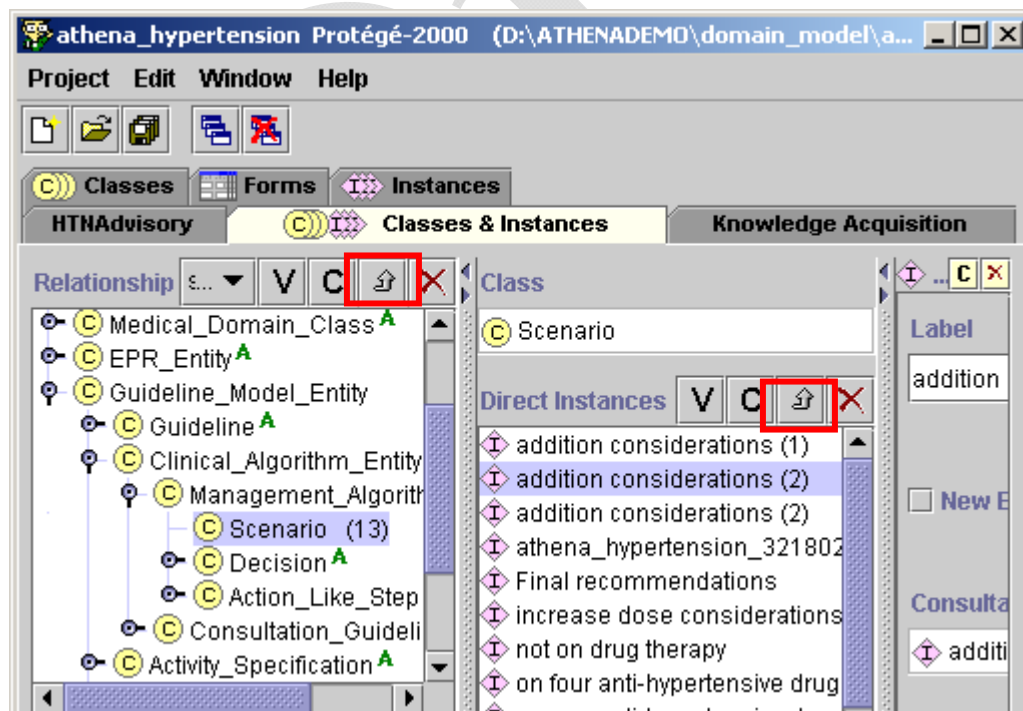


Figure 21 - References buttons (up arrow, between C and X buttons) in the Classes & Instances tab

If you select the Scenario class and click on the references button in the class hierarchy pane, for example, a window showing all references to the Scenario class will pop up (Figure 22). The window shows that: the Scenario class is an allowed class of the first_object slot in the Followed_By class; it is the direct instance of the :STANDARD-CLASS metaclass; it is the direct subclass of the Management_Algorithm_Entity class; it is the direct type of a number instances; and it is the slot value type of the previous_scenarios slot.

Frame	Slot	Facet
Followed_By	first_object	allowed-classes
:STANDARD-CLASS	:DIRECT-INSTANCES	
Management_Algorithm_Entity	:DIRECT-SUBCLASSES	
addition considerations (1)	:DIRECT-TYPE	
addition considerations (2)	:DIRECT-TYPE	
addition considerations (2)	:DIRECT-TYPE	
athena_hypertension_321802	:DIRECT-TYPE	
Final recommendations	:DIRECT-TYPE	
Increase dose considerations	:DIRECT-TYPE	
not on drug therapy	:DIRECT-TYPE	
on four anti-hypertensive drugs	:DIRECT-TYPE	
on one anti-hypertensive drug	:DIRECT-TYPE	
on two or three anti-hypertensive drugs	:DIRECT-TYPE	
SBP>=220 or DBP>=115	:DIRECT-TYPE	
substitute considerations	:DIRECT-TYPE	
substitution considerations	:DIRECT-TYPE	
previous_scenarios	:SLOT-VALUE-TYPE	

Figure 22 - Window showing all references to the Scenario class

From the window that shows all references to the Scenario class, you can select a class, slot, or instance (e.g., Management_Algorithm_Entity class in Figure 23), and click on the references button on the form to open another window that shows all references to the selected frame (e.g., Figure 24 shows all references to the Management_Algorithm_Entity class in the knowledge base).

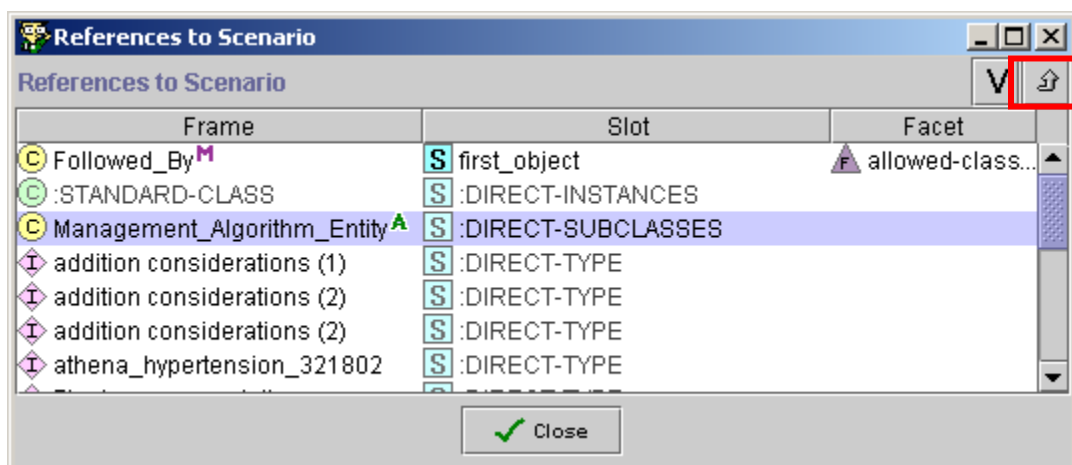


Figure 23 - Selecting a frame (e.g., Management_Algorithm_Entity class) in the References to Scenario window, and clicking on the references button to find all references to the selected frame

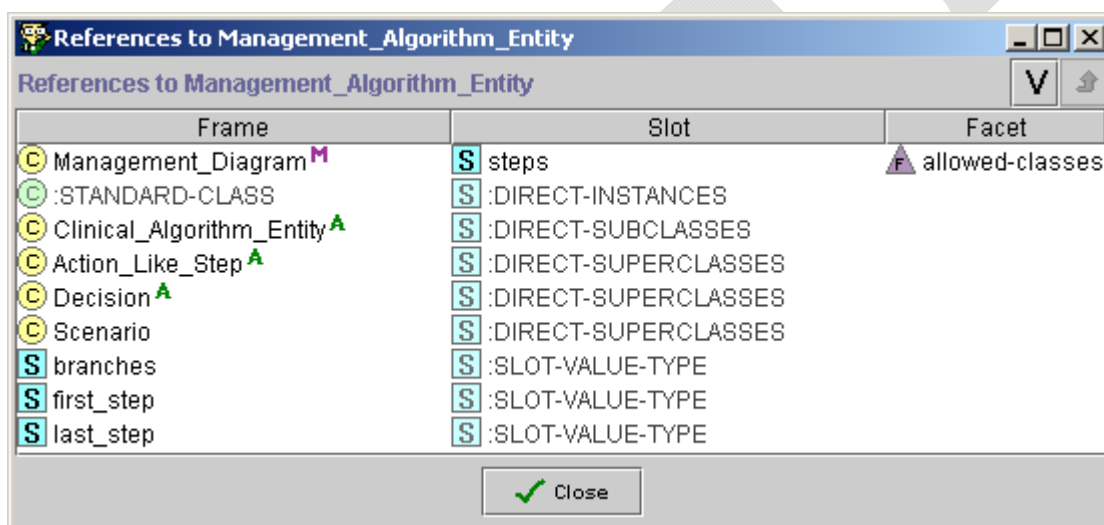


Figure 24 - All references to Management_Algorithm_Entity

III. EON Models

EON models, consisting of an extensible set of Protégé projects, structure the knowledge and data so they can be used to encode guidelines and protocols in a form suitable for generating patient-specific recommendations. The EON models include:

1. *Patient Data Model* (EPR_Entity hierarchy) – defines the structure of patient information used by the EON system. It converts the clinical data from the electronic medical record (or other source) into a format the Guideline Interpreter can process.

2. *Medical Concept Model* (Medical_Domain_Class hierarchy) – primarily defines the medical vocabulary used in encoding guidelines. The terms in the medical concept model are mapped to standard terminologies or to the terminology used in the host information system.
3. *EON Guideline Model* (Guideline_Model_Entity and Expression hierarchy) – defines the structure of a computable EON guideline.

As shown in Figure 25, the EON Guideline Model contains:

1. *Eligibility criteria* – define the target population of the guideline. For example, a guideline on Hypertension would have a target population of patients with high blood pressure as an eligibility criterion.
2. *Goals* (e.g., target blood pressures) – specify patient states the guideline aims to help achieve.
3. *Abstractions about patients* (e.g., the risk group to which a patient belongs) – represent interpretations about a patient's medical condition.
4. *A clinical algorithm* – organizes a collection of patient scenarios, decisions to be made, and possible action choices. Each action choice includes decision criteria for giving preference to the actions specified in the choice. The actions may include sending a message, referring a patient, evaluating activities to recommend, or starting, stopping and modifying activities.
5. *Activity specification* – represents an action that can take place over time (e.g., taking a medication to manage chronic problems). Activity specifications have properties, such as compelling indications, relative indications, relative contraindications, and absolute contraindications, that can be used to determine whether the activity is appropriate. Drug_Usage and Guideline_Drug are the two classes of activities heavily used in the ATHENA Knowledge Base. Instances of Drug_Usage (e.g., ACE Inhibitors) often contain information on classes of drugs, while instances of Guideline Drug contain specific information about a particular drug in a class (e.g., Captopril).
6. *Computable expressions* (e.g., eligibility criteria) – expressions, written using the expression languages available in the EON Guideline Model. They can be evaluated using coded patient data to infer valid statements about a patient.

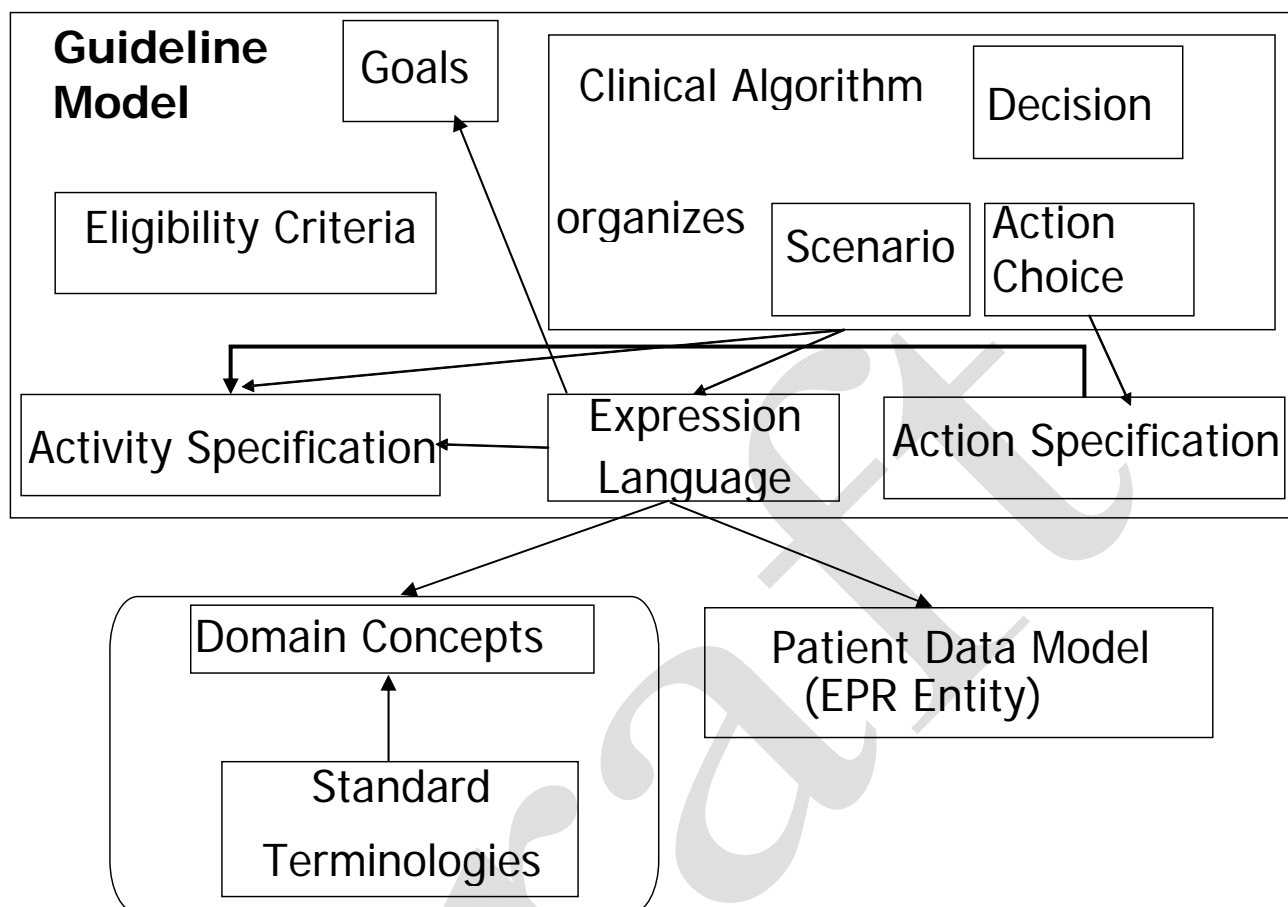


Figure 25 - Overview of the relationships among components of EON models

The following subsections describe the EON models and ATHENA Knowledge Base in more detail. They identify three levels: those features of the knowledge base that have impact on the display of the ATHENA GUI (as of November 2005), those features that are implemented by the EON Guideline Interpreter (as of November 2005) but are not displayed on the ATHENA GUI, and those features of the EON Guideline Model that are not fully implemented by the Guideline Interpreter. The subsections cover the patient data model, the guideline concept model, the EON Guideline Model, and the EON expression language.

III.1. Patient Data Model

The EON patient data model defines the structure of the patient data used by the EON system. The Guideline Interpreter, for example, uses the data to determine whether a patient satisfies the eligibility criteria of the guidelines represented in the ATHENA Knowledge Base. Patient data are modeled as static (i.e., unchanging), as time stamped, or as having a time interval during which the information they represent is valid. For example, laboratory test results may be modeled as instances of `Numeric_Entry`, which has a code, a numeric value, and a time stamp.

The Guideline Interpreter evaluates decision criteria using patient data represented in the format of the patient data model. The EON patient data model, represented by subclasses of the *EPR_Entity* class (see Figure 26), consists of the following classes:

- *Encounter class* – represents records of the encounters a patient has with health-care providers.
- *Patient class* – represents non-varying demographic information about specific patients (e.g., sex and date of birth).
- *Note_Entry class* – describes time-stamped non-numeric observations made by clinicians.
- *Numeric_Entry class* – represents results of quantitative measurements.
- *Adverse_Event class* – models adverse reactions to specific substances.
- *Condition class* – represents medical conditions that persist over intervals of time (i.e., it has a start time and may have a stop time).⁵
- *Medication class and Procedures class* – two intervention classes, they model drug prescriptions and other medical procedures that have been recommended, authorized, or used.

All entities in the patient data model are assertions about the demographic and clinical conditions of specific patients. The model is not designed to replicate all the contents of an electronic medical record. Rather, it includes only those distinctions relevant to modeling guidelines and protocols.

⁵ The Condition class is not used in ATHENA.

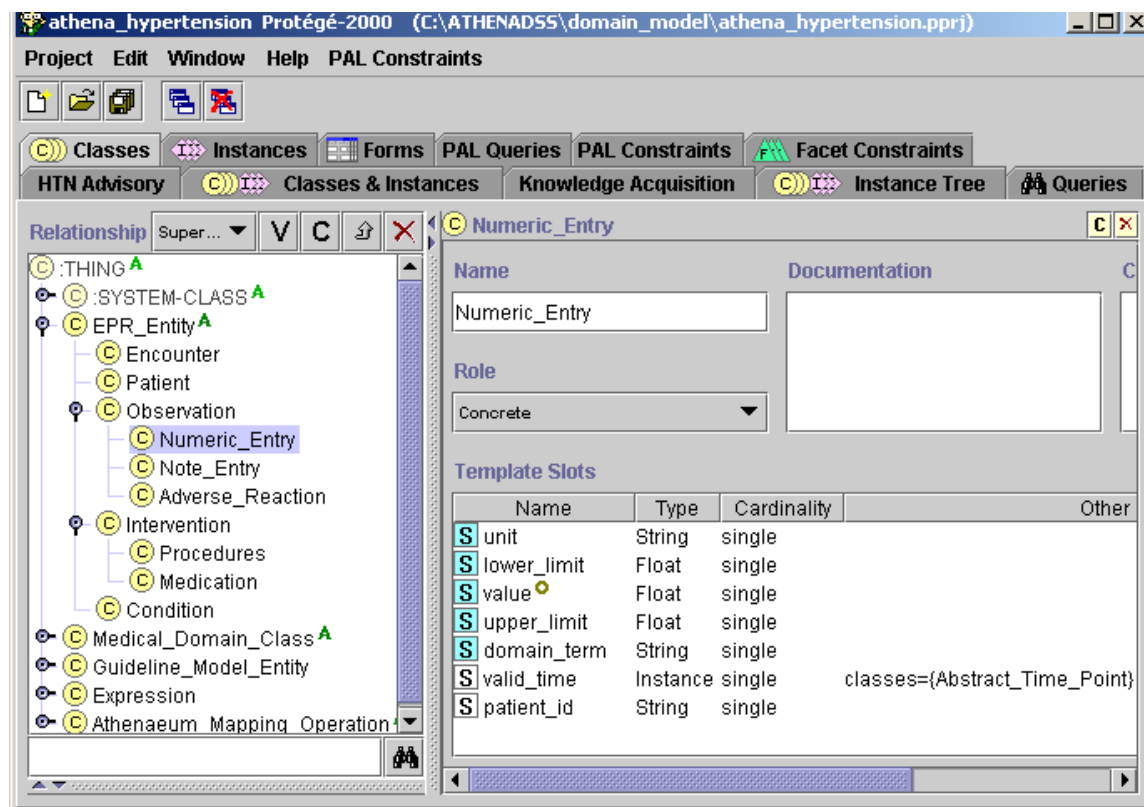


Figure 26 - The classes in the EON patient data model (EPR_Entity)

III.2. Medical Concept Model

The medical concept model contains classes that represent:

1. terminological concepts necessary to encode patient information and guideline statements,
2. references to supporting material, and
3. relationships between different drugs, and between drugs and medical conditions.

These concepts and relationships are organized in a taxonomic hierarchy (see Figure 27). The particular collection of classes represented in the current EON medical concept model is a historical legacy that does not conform to any external standard such as SNOMED Clinical Term.

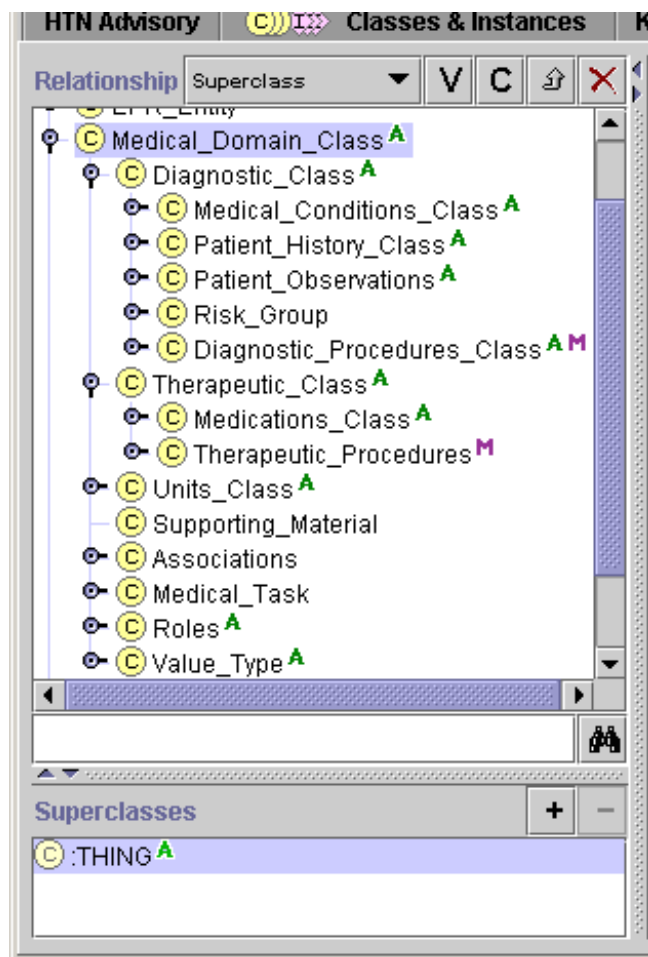


Figure 27 - The top-level classes in the medical concept model (subclasses of Medical_Domain_Class)

Classes in the Medical_Domain_Class hierarchy are instances of user-defined metaclasses. The important metaclasses to understand are:

1. *Canonical_Terms_Metaclass* – Canonical_Terms_Metaclass is the root of most metaclasses used in the Medical_Domain_Class hierarchy. (The only exception is Ordered_List_Value, see below). Medical concepts represented in the hierarchy should be instances of this metaclass or its subclasses. The Canonical_Terms_Metaclass provides a PrettyName slot that can be used to provide a more human-understandable name for the class. The Synonyms slot is not used.
2. *Medical_Conditions_Metaclass* – Instances of Medical_Conditions_Metaclass represent findings, diagnosis, and problems. They should be subclasses of the Medical_Conditions_Class in the Medical_Domain_Class hierarchy (Figure 28). A term representing a finding or diagnosis, such as Atrial_Fibrillation, may be mapped to patient

data by making codes used in data subclasses of the guideline concept (e.g., the ICD9 codes 427.31 and 427.32). Like the *Canonical_Terms_Metaclass*, only the *PrettyName* slot is used in the current system. Note that the current ATHENA Knowledge Base has PAL criteria (see III.4.2) whose variables range over instances of *Medical_Conditions_Metaclass*. (So if a finding is not an instance of *Medical_Conditions_Metaclass*—e.g., it is an instance of *Canonical_Terms_Metaclass*—it will not be considered when the system looks for, for instance, the contraindications of a drug.)

3. *Diagnostic_Term_Metaclass* – *Diagnostic_Term_Metaclass* provides a second way to relate guideline concepts to terms used in data. Instances of *Diagnostic_Term_Metaclass* have a *DiagnosticCriteria* slot that takes a Boolean criterion. If, for a patient, the criterion evaluates to *true*, then the EON Guideline Interpreter concludes that the finding is present for the patient. For example, in the ATHENA Knowledge Base, “hypertension without comorbidities that compellingly indicates thiazides or beta blocker” is represented as an instance of *Diagnostic_Term_Metaclass*, with *DiagnosticCriteria* “presence of hypertension and absence of myocardial infarction, diabetes, heart failure, and isolated systolic hypertension”.
4. *Derived_Parameter_Metaclass* – An instance of a *Derived_Parameter_Metaclass* is used to represent a concept whose value may come from multiple sources. In ATHENA, *Treatment_Systolic_BP* and *Treatment_Diastolic_BP* are instances of the *Derived_Parameter_Metaclass* where the *definition* slot specifies an ordered list of queries that can be used to obtain values for the two parameters. For example, the value for *Treatment_Systolic_BP* is derived by querying sequentially for: *MD_Typical_Systolic_BP* (the blood pressure a clinician uses for the purpose of managing hypertension), *MD_Clinical_Systolic_BP* (the most recent blood pressure measurement entered by a clinician through the ATHENA DSS interface), and *DB_Systolic_BP* (the most recent blood pressure stored into the VistA database). The Guideline Interpreter uses the first data returned by this ordered list of queries. Thus, *MD_Typical_Systolic_BP*, if present, will be used instead of the *DB_Systolic_BP* blood pressure data in the database.
5. *Interval-Valued_AtomicTest_Metaclass* – Instances of *Interval-Valued_AtomicTest_Metaclass* are used to represent test results, such as the level of serum creatinine, that may have upper and lower limits of normal. The Guideline Interpreter also uses the *precision* slot to determine the number of digit to the right of the decimal point in displaying values of the test result.
6. *Medications_Metaclass* – Instances of the *Medications_Metaclass* are the generic drug names such as terazosin and lisinopril. Only the *PrettyName* slot is used. Again, because current PAL criteria use variables that range over instances of *Medications_Metaclass*, a drug that is not an instance of this metaclass will be missed in the evaluation of PAL criteria.

7. *Ordered_List_Value* – Instances of the *Ordered_List_Value* metaclass have pointers to the next and previous instances in a list. This metaclass is used to model the sequence relationship among *High_Dose*, *Medium_Dose*, and *Low_Dose*.

Other metaclasses, such as *Drug_Category_Metaclass*, play no role in the interpretation of the knowledge base.

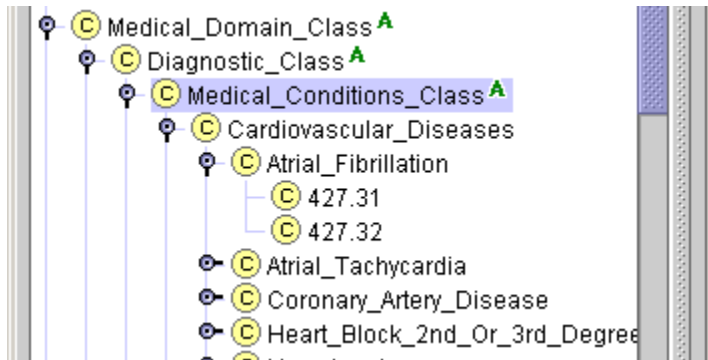


Figure 28 - Part of the ATHENA Medical Conditions Class hierarchy. Those classes in the hierarchy that have no subclasses should be either mapped to terms used in patient data or defined using *Diagnostic_Term_Metaclass*.

III.2.1. Terminology Hierarchies

The *Diagnostic_Class*, *Therapeutic_Class*, *Medical_Task*, *Value_Type*, and *Units_Class* hierarchies supply controlled terminologies for the formal encoding of guideline knowledge in the ATHENA Knowledge Base and for patient data (Figure 29). *Units_Class* is not used by the Guideline Interpreter, which assumes that the data and knowledge base are using consistent units of measure.

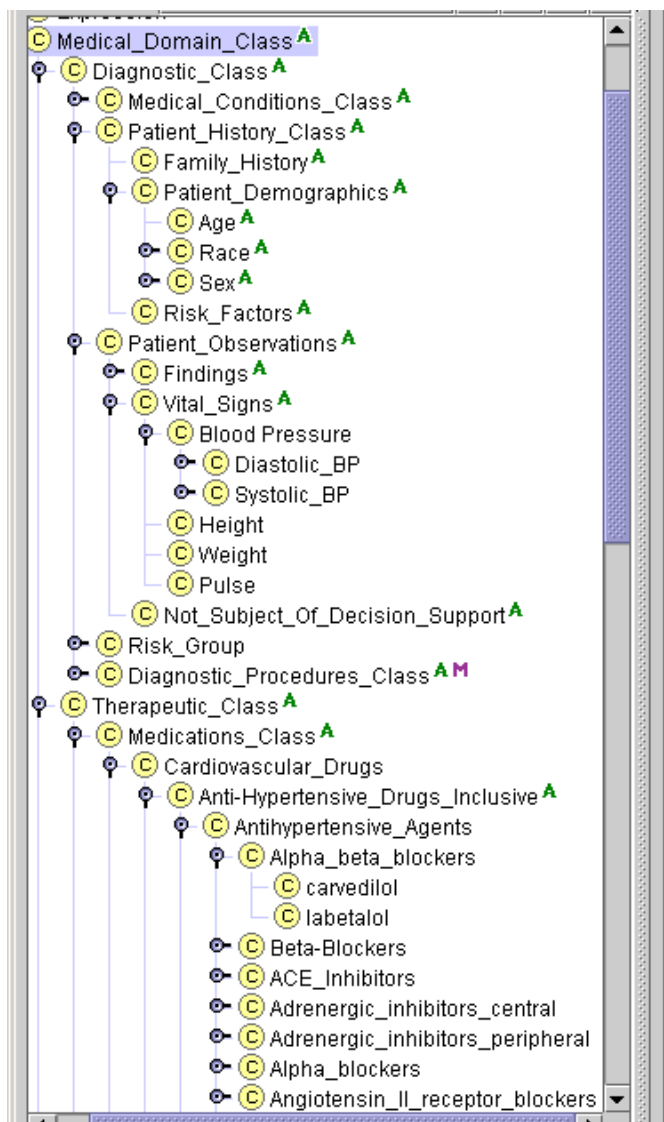


Figure 29 - Terminology classes in the Medical_Domain_Class hierarchy of the ATHENA Knowledge Base

With the exception of the Value_Type hierarchy, which is explained later in this subsection, the terminological hierarchies can be organized into any form by clinicians and knowledge engineers maintaining the ATHENA Knowledge Base, as long as a concept (represented by a class):

1. is mapped to a term in the patient data, with the mappings defined by the ATHENAeum_Mapping_Operation classes (e.g., Creatinine class is mapped to the CREATININE string);⁶
2. has subclasses (e.g., ICD9 codes) that are mapped directly to patient data;

⁶ The mappings of drug names are maintained in the SQL database without reference to mappings in the ATHENA Knowledge Base.

3. is defined as instances of `Diagnostic_Term_Metaclass` or `Derived_Parameter_Metaclass`; or
4. has subclasses that are mapped (as in case 1 or 2, above) or defined (as in 3).

In addition, the concept should be an instance of an appropriate metaclass (e.g., a generic drug class like *lisinopril* should be an instance of the `Medications_Metaclass` and a finding such as *fever* should be an instance of `Medical_Conditions_Metaclass`). The terminology hierarchy may contain additional classes (e.g., `Patient_Observation`) that are used to group concepts for navigational purposes; they play no role in the decision-support system.

The ATHENA Knowledge Base organizes medical conditions and drug classes into classification hierarchies as shown in Figure 28 and Figure 29. The medical condition classes either: have—at the most specific level—ICD9 codes that correspond directly to data in the VA database, or are defined using `Diagnostic_Term_Metaclass` or `Derived_Parameter_Metaclass`. The drug class hierarchy has, at the most specific level, generic drug names—such as *labetalol* in Figure 29—that are mapped to drug terms in VA prescriptions. (The drug mapping is maintained in an ATHENEON database table.) In addition, the ATHENA Knowledge Base enumerates the vitals, laboratory test results, and demographic terms used in encoding the hypertension guidelines. These terms are mapped to terms used in the VA patient database.

III.2.2. Value_Type Hierarchy

The `Value_Type` hierarchy defines concepts that are used in the EON system as enumerations of mutually exclusive categorical values. A complete hierarchy listing is shown in Figure 30.

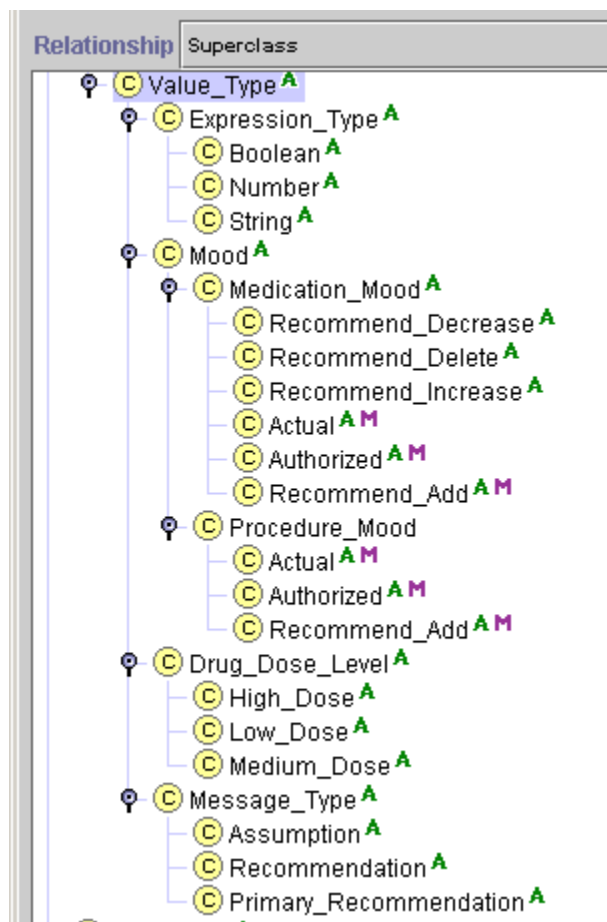


Figure 30 - The set of system-recognized terms, as organized in the Value_Type hierarchy

Medication Mood

As the Guideline Interpreter generates recommendations about drug usage, it needs to perform additional tasks based on these recommendations (or the lack thereof). For example, in order to generate patient-specific messages associated with the recommendation to add a drug,⁷ developers of the ATHENA Knowledge Base must encode messages that are based on the system's drug recommendations (i.e., the collateral actions associated with an instance of Drug_Usage). In effect, these recommendations must be represented as data upon which additional reasoning can be based. Accordingly, the Guideline Interpreter generates instances of the Medication class to represent its recommendations. To distinguish among system-generated Medication instances from patient data, the Guideline Interpreter uses Medication_Mood values (see Figure 30):

- The Authorized mood value represents the medication that is prescribed.
- The Actual mood value represents the medication that is currently being taken by the patient.

⁷ These patient-specific messages associated with a drug recommendation are displayed in the ATHENA GUI as the Info button messages next to a drug recommendation.

- The Recommend_Decrease, Recommend_Delete, Recommend_Increase, and Recommend_Add mood values represent possible recommendations for a drug.

Drug_Dose_Level

The possible values of Drug_Dose_Level in ATHENA are High_Dose, Low_Dose, and Medium_Dose. They are instances of the Ordered_List_Value metaclass that allows the specification of an ordered list (Figure 31). Thus, Low_Dose's *next* attribute value is Medium_Dose, whose *next* attribute is High_Dose. Similarly, the *previous* attribute of High_Dose is Medium_Dose; and the *previous* attribute of Medium_Dose is Low_Dose. These terms are used in the Guideline_Drug class to specify ordinal levels of drug doses. The actual ranges for the dose levels are defined in the dose_level_ranges slot of Guideline_Drug class. The Guideline Interpreter classifies a patient's medication dose according to the dose levels specified in instances of the Guideline_Drug class to determine whether the current daily dose is at the maximum level. The values of Drug_Dose_Level can be changed (e.g, Level_1, Level_2, etc. instead of Low_Dose, Medium_Dose and High_Dose), as long as the values are instances of Order_List_Value metaclass and the ranges are defined in instances of Guideline_Drug.

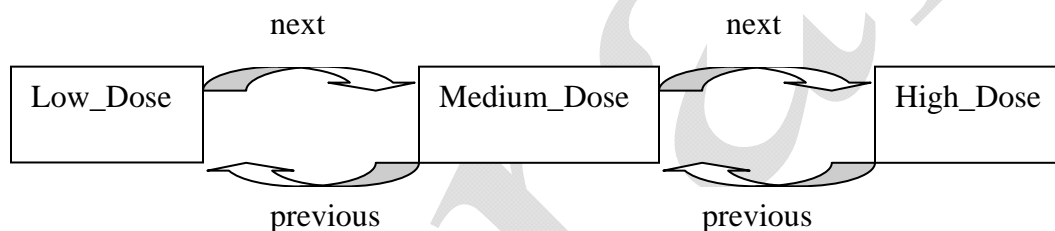


Figure 31 - Use of Ordered_List_Value metaclass to create a sequence of ordinal drug dose levels

Message Type

Subclasses of this class define the possible values for the message_type slot of the On_Screen_Message class (see Figure 30 and Subsection III.3.3). In ATHENA, the values are Assumption, Recommendation, and Primary_Recommendation. The Guideline Interpreter uses these values to annotate messages sent to the client and the current ATHENA GUI, using the message-type information to display messages in different locations. The developer of the ATHENA Knowledge Base can change the values of these message types as long as the client program knows how to interpret the values.

III.2.3. Associations

Figure 32 shows an example of an instance of `Drug_Indication_Relation` that holds supporting material for diabetes with proteinuria as a competing indication for the use of ACE inhibitors. In the ATHENA system, the GUI uses these instances to pop up HTML pages. The instances are not used by the Guideline Interpreter.



III.2.4. Supporting Material

37

d:\ATHENADEMO\doc\ATHENA\EvidenceDisplayATHENA\Diabetes_ACE\SS_Diabetesand ACE.html).

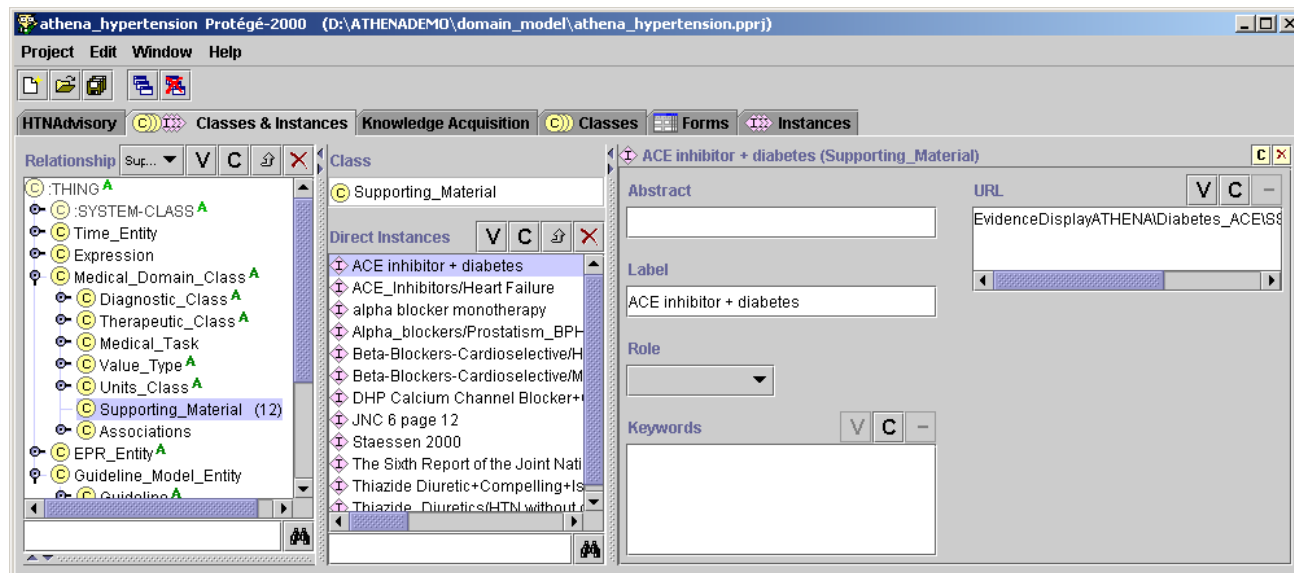


Figure 33 - Example of a Supporting_Material instance

III.3. EON Guideline Model

The introduction to Subsection III describes the overall conceptualization of the EON Guideline Model. This subsection describes the details of the EON Guideline Model as it is used in the ATHENA Knowledge Base. It addresses individuals charged with maintaining the ATHENA Knowledge Base or with creating a guideline knowledge base like the ATHENA Knowledge Base.

Figure 1 shows the main components of the ATHENA Knowledge Base as represented in the instance of ATHENA_Management_Guideline class. The ATHENA_Management_Guideline class, created to facilitate navigation in the Protégé tool, is an ATHENA-specific extension of the EON Guideline Model. The ATHENA_Management_Guideline class is a subclass of the Management_Guideline class. The Guideline Interpreter reasons with instances of the Management_Guideline. (ATHENA_Management_Guideline being a subclass of Management_Guideline, instances of ATHENA_Management_Guideline are automatically instances of the Management_Guideline class.)

The description of the EON Guideline Model and the ATHENA Knowledge Base will be divided into four parts:

1. properties of the Management_Guideline class other than the clinical algorithm,
2. the clinical algorithm,

3. the actions (e.g., substituting drugs, raising doses, and sending messages) that the DSS can recommend to the end users
4. the clinical interventions (i.e., Drug_Usage and Guideline_Drug classes) whose properties (e.g., indications and dose range) are used by the Guideline Interpreter to generate recommendations.

The topics covered by these subsections represent the four areas that a developer of a knowledge base has to conceptualize the guideline knowledge in creating a computable guideline knowledge base in the EON format.

III.3.1. Management Guideline

The *label* attribute is a short name for the guideline. It also serves as the identifier for the Guideline Interpreter to access the correct instance of Management_Guideline (i.e., JNC-VI Hypertension Guideline, in the case of ATHENA Knowledge Base).⁸

The *eligibility_criteria* attribute defines the target population of a guideline encoded in EON. In applying the eligibility criteria, the Guideline Interpreter identifies a patient as *eligible* if none of the eligibility criteria explicitly rule out him or her (i.e., a criterion evaluates to *false*). Thus, the Guideline Interpreter will apply the guideline to the patient even if there is insufficient information to rule out a patient (i.e., if the criteria evaluate to *unknown*). Section III.4 describes the formats of criteria.

⁸ Section IV. describes the *setGuideline* method in the PCASession interface, which a client program uses to specify the applicable guideline. The Guideline Interpreter uses the value of the *label* attribute to match for the string specified in the argument of the *setGuideline* method.

Management_Guideline (instance of :STANDARD-CLASS)

Name
Management_Guideline

Documentation
Management guidelines model decisions and actions that lead to dependent changes in patient states over time. A management guideline has one or more goals.

Constraints

Role
Concrete

Template Slots

Name	Cardina...	Type	Other Facets
assume_if_no_goal...	single	Symbol	allowed-values={make_alternative_assumption...
authors	multiple	String	
clinical_algorithm	required...	Instance of Managemen...	
eligibility_criteria	multiple	Instance of Criterion	
goal	multiple	Instance of Conditiona...	
goals	multiple	Instance of Goal	
label	required...	String	
patient_characteriz...	multiple	Class with superclass...	
primary	single	Boolean	default=true
references	multiple	Instance of Supporting...	
title	single	String	
version	single	String	

Figure 34 - The definition of the Management_Guideline class

The *goal* attribute is used to specify the goals a guideline establishes for each patient. Values of the attribute should be a list of Conditional_Goal instances. Figure 35 shows an example of a conditional goal. The selection_criterion attribute (Selection Criterion in Figure 35) is a Boolean criterion that, if evaluated to *true* for the patient, determines whether the criterion_to_achieve (Criterion To Achieve) applies to the patient.

Figure 35 - Conditional_Goal example in the ATHENA Knowledge Base

Goals-Conditional goals in the Guideline Interpreter should be mutually exclusive. That is, only one conditional goal should be applicable to a patient. The criterion to achieve can be a complex conjunction or disjunction (*and* and *or*) of other criteria. In the ATHENA Knowledge Base, the goals are target systolic and diastolic blood pressures.

The goal criteria can be used in decision-making through instances of the *Goal_Criterion* class. Figure 36 shows the goal criterion used in ATHENA. It is simply a reference to an instance of *Management_Guideline*. The Guideline Interpreter evaluates the conditional goals associated with the guideline, and returns *true*, *false*, or *unknown*. If the goal criterion evaluates to *unknown*, then the Guideline Interpreter generates alternative recommendations based on assumptions it makes about the patient's status (i.e., the assumption that the patient satisfies the goal or that the patient does not satisfy the goal).

Figure 36 - Goal_Criterion used in the ATHENA Knowledge Base

Since version 2.08, *Management_Guideline* has the *goals* attribute, which can have one or more instances of the *Goal* class. Instances of the *Goal* class has one or more *conditional_goals*. The idea is that a guideline may set one or more collections of conditional goals (e.g., goals for BP, for LDL, etc.).

There is also a new *SelectedGoal_Criterion* class with which a guideline encoder can check to see whether particular instance of *Goal* is satisfied.

The *patient_characterization* attribute is the place to specify abstractions about a patient case that the developer of the knowledge base wants the Guideline Interpreter to return as part of the recommendation. In the ATHENA Knowledge Base, the specified abstractions are risk groups used to classify patients (e.g., risk group A being no risk factor and no target organ damage or clinical cardiovascular disease). The values of this attribute are instances of the *Diagnostic_Term_Metaclass*. Currently, the ATHENA GUI does not display these patient abstractions that the Guideline Interpreter computes and returned as part of the recommendation.

The *assume if no goal value* attribute controls whether the Guideline Interpreter makes alternative assumptions when the goal of the guideline evaluates to unknown. Possible attribute values are *make alternative assumptions*, *assume satisfied*, *assume unsatisfied*, *make no assumption*.

The *title*, *version*, and *authors* attributes are not used by the DSS, but should be used to annotate the knowledge base.

III.3.2. Clinical Algorithm

Individuals charged with creating a guideline knowledge base like the ATHENA Knowledge Base—the developers of the knowledge base—can specify the flow of the decisions and actions of a guideline in the *clinical_algorithm* attribute of the *Management_Guideline* class. The value of this attribute should be an instance of the *Management_Diagram* class. Protégé provides a graphical tool for viewing and editing the algorithms (Figure 37). A clinical algorithm in the EON system basically functions to organize guideline recommendations into a collection of starting scenarios (pink squares in Figure 37), such that a patient case falls into exactly one of them. Associated with each scenario is a *consultation guideline* where the developer can specify actions that should be performed for all patients classified into the scenario (actions such as showing warning messages about current medications).

Instances of the *Followed_By* class specify possible paths of an algorithm (black arrows in Figure 37). Control of flow in an algorithm is achieved by using instances of *Case_Step* (yellow diamond in Figure 37), by which exactly one path is selected (i.e., deterministic choice); or by using instances of *Choice_Step* (light yellow ovals), where more than one choice is possible (i.e., non-deterministic choices). Instances of *Action_Choice* (green ovals) follow a choice step. In action choices, the developer specifies the criteria for ruling out or for preferring the choice; and *Action_Specification* defines the recommended actions for that choice. Subclasses of *Action_Specification* fall into two categories: (1) those that perform actions directly (mostly sending messages to the user) and (2) those that do something to an “activity” (e.g., recommending adding or stopping the use of a drug, evaluating the set of drugs to add, and increasing the dose of current medications).

Figure 37 is a partial view of a simple clinical algorithm in the ATHENA Knowledge Base. It shows two possible starting scenarios. The first one (characterized by $SBP \geq 220$ or $DBP \geq 115$) leads to an Action_Choice step where an urgent attention message is sent. The second scenario includes cases in which the patient is not taking any antihypertensive medication, as the patient's blood pressures are not in the range of those in the first scenario. This scenario leads to four possible management choices: adding an antihypertensive drug, recommending lifestyle changes, prescribing an ACE inhibitor, and prescribing a beta-blocker for secondary prevention. If a specific drug should be added but none has been indicated (see branch of the *no indicated drug?* case step), the algorithm makes default choices; otherwise the algorithm continues to a scenario where messages are generated that suggest considering the addition of drugs.

The Guideline Interpreter uses a clinical algorithm by:

1. determining which starting scenario into which the patient case falls,
2. following the paths from that scenario, then
3. stopping when it reaches a terminal node (i.e., a node with no outgoing link) or when it encounters case steps or choice steps for which there is no preferred alternative.

As the Guideline Interpreter traverses the clinical algorithm, it constructs a recommendation that is eventually presented to the end user of the DSS.



Figure 37 - Part of a clinical algorithm in the ATHENA Knowledge Base

The components of the clinical algorithm are described in more detail, below.

Scenario

Figure 38 shows the details of the *not on drug therapy* scenario of the clinical algorithm. Because *new_encounter* is true (shown as *New Encounter* in the figure), the Guideline Interpreter will treat the scenario as a possible starting point in the clinical algorithm. The type of patient for whom a scenario is appropriate is defined by the criterion in the *precondition* slot of the scenario (shown as *Precondition*). At the start of its operation, the Guideline Interpreter collects all instances of Scenario class whose *new_encounter* slot is checked, and uses the value of their *precondition* slot to determine which scenario to use for the patient. For example, in Figure 38, the clinical algorithm's *new_encounter* slot is checked, so the *not on drug therapy* scenario will be used as a starting point in generating the guideline advisory if its precondition (shown as *no drug therapy and [SBP<220 and DE..* in Figure 38) evaluates to *true*. The *followed_by* slot (shown as *Followed By* in Figure 38) links the scenario to the next node in the clinical algorithm. The ATHENA algorithm contains five *new_encounter* scenarios, which are the potential starting points for a patient case: one based on blood pressure (SBP \geq 220mmHg or DBP \geq 115mmHg) and four based on the number of antihypertensive medications prescribed for the patient (0, 1, 2 or 3, and 4). In Figure 37, starting scenarios are the two orange rectangles: SBP \geq 220 or DBP \geq 115, and *not on drug therapy*.

In general, anyone creating a guideline knowledge base should ensure that the definitions of scenarios have preconditions that use commonly available data, and the classification of patients into scenarios should be unambiguous. Furthermore, one scenario should lead to management decisions that differ from decisions resulting from other scenarios. However, some overlap is acceptable (e.g., secondary prevention decisions are common to all ATHENA scenarios).

Figure 38 - *Not on drug therapy* scenario example in ATHENA

In addition to determining the starting point for generating a guideline advisory, the second function of a scenario is to specify actions that should be performed for patients who have been classified to meet the preconditions of the scenario (**Error! Reference source not found.**). The *Consultation_Guideline* has a *steps* slot where these scenario-based actions are specified. The

steps of a Consultation_Guideline consist of instances of Consultation_Branch_Step (**Error! Reference source not found.**) and Consultation_Action_Step (**Error! Reference source not found.**).

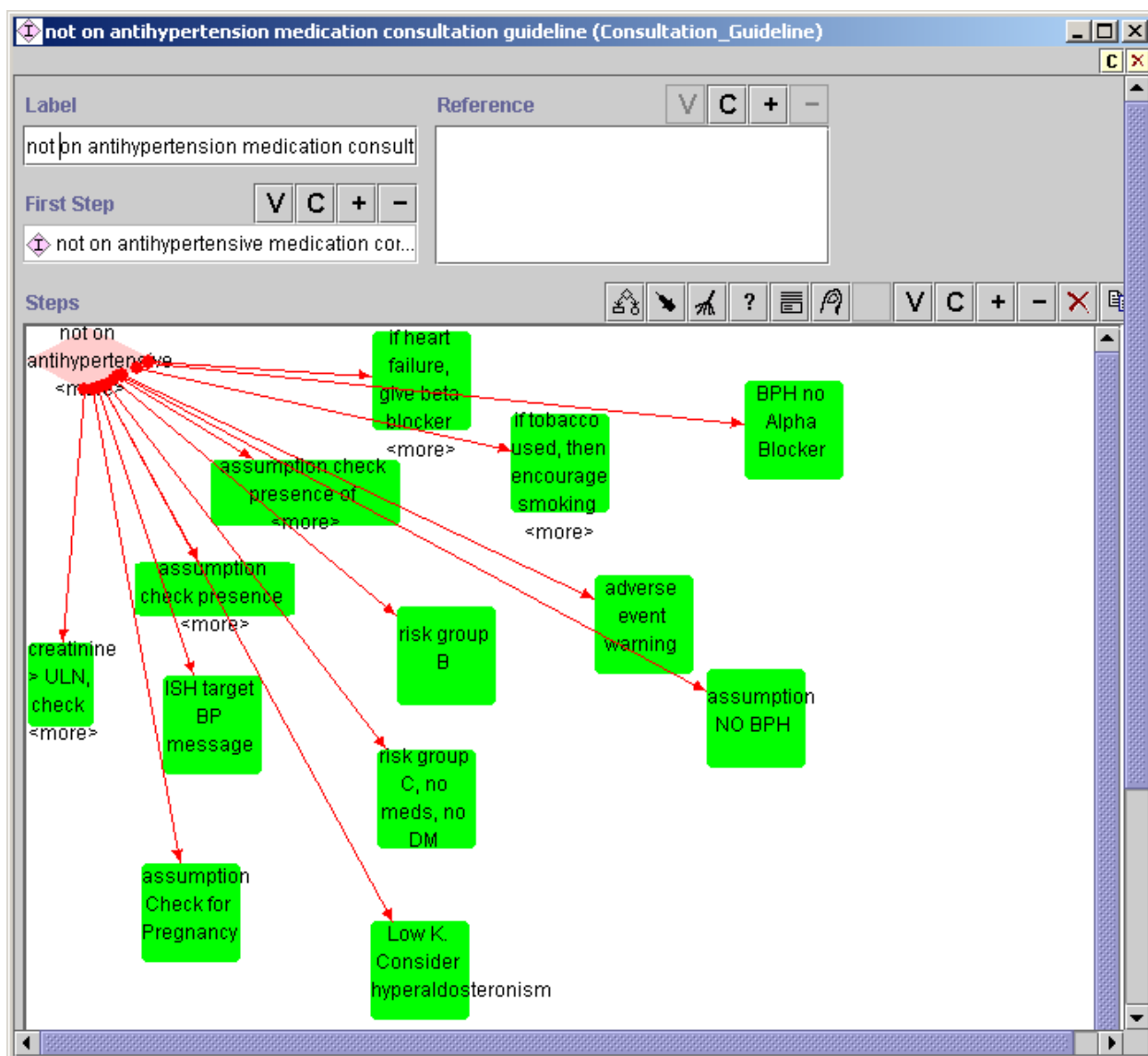


Figure 39 - Consultation_Action_Step generating messages within Consultation_Guideline.
For instance, it encourages smoking cessation when the patient is a smoker.

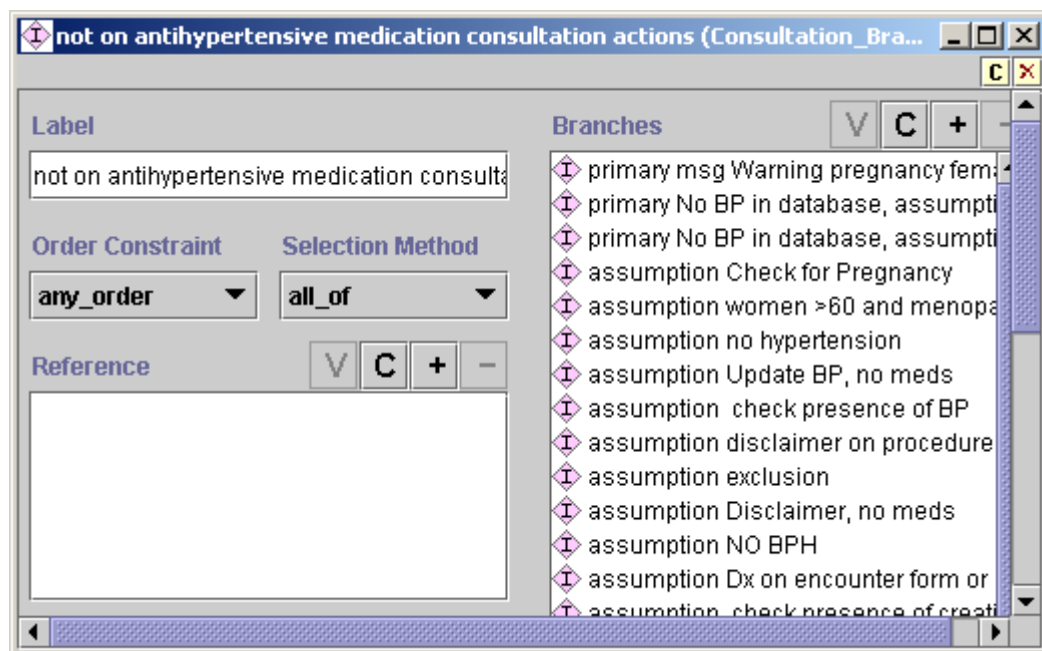


Figure 40 - Consultation_Branch_Step example

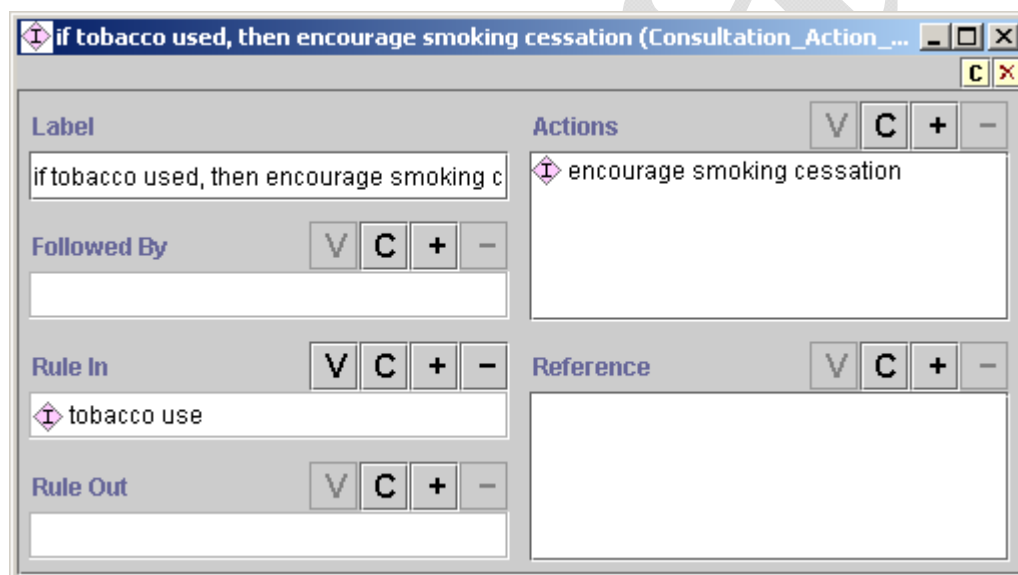


Figure 41 - Consultation_Action_Step example showing the “encourage smoking cessation” action if the “tobacco use” rule_in condition (shown as Rule In) is true

The Consultation_Guideline class was designed to allow interactive presentation of and requests for information. Each instance of Consultation_Branch_Step represents a potential decision point for determining what information to present or to request. In the current implementation of the Guideline Interpreter, however, the Consultation_Guideline allows presentation only of textual messages. An instance of Consultation_Guideline should start with an instance of Consultation_Branch_Step (specified in the first_step slot). Possible values of the *branches* slot

(shown as Branches) are other instances of Consultation_Branch_Step and Consultation_Action_Step. An instance of Consultation_Action_Step has a rule-in and a rule-out condition, where an action step is marked as preferred if it is not ruled out and is ruled in. ATHENA GUI presents to the user only messages that are preferred. The *actions* slot (shown as Actions) in Consultation_Action_Step should contain instances of On_Screen_Messages or its subclasses. In the ATHENA Knowledge Base, these messages give warnings about specific conditions a patient may have and assumptions the system makes.

Case Step

A decision represents a choice from a set of competing alternatives. The EON Guideline Model supports various ways of describing alternatives, and also the corresponding selection mechanism. In the current model are two subclasses of decisions: Case_Step models decisions that are resolved by evaluating an expression and using the result to select one alternative; and Choice_Step models decisions for which one or more alternatives may be preferred, leaving the choice to the end user. Thus, ATHENA DSS may recommend that, according to the guidelines, both Beta Blocker and ACE Inhibitor are good choices for a patient, leaving the actual decision for a clinician to make.

Figure 42 shows an instance of Case_Step that contains an expression (with label *no indicated drug and cannot increase dose?*). The expression, in this case, evaluates to true, false, or unknown. If the expression evaluates to true, then the system steps to the choice step to consider default drug recommendations. If the expression evaluates to false, then the system goes to a scenario where, through the use of a consultation guideline, messages are generated. If the expression evaluates to unknown, because no branch for the unknown value is specified, the Guideline Interpreter will not continue on this path.

The system determines which branch to take based on the value specified in the link between the case step (yellow diamond in Figure 42) and its branches (the red arrows). The link is an instance of the Case_Selection class (Figure 43), which has slots for the source and target of the link (first_object/First Object and second_object/Second Object, respectively) and the *value*/Value slot.⁹ If the expression in the first_object slot evaluates to the value in the Value slot, the object in the second_object slot should be activated.

⁹ The description of the attributes of the Case_Selection class applies only to the Protégé 1.7 version of the system. In Protégé 2.0 and later, the attributes of the Case_Selection class differ.

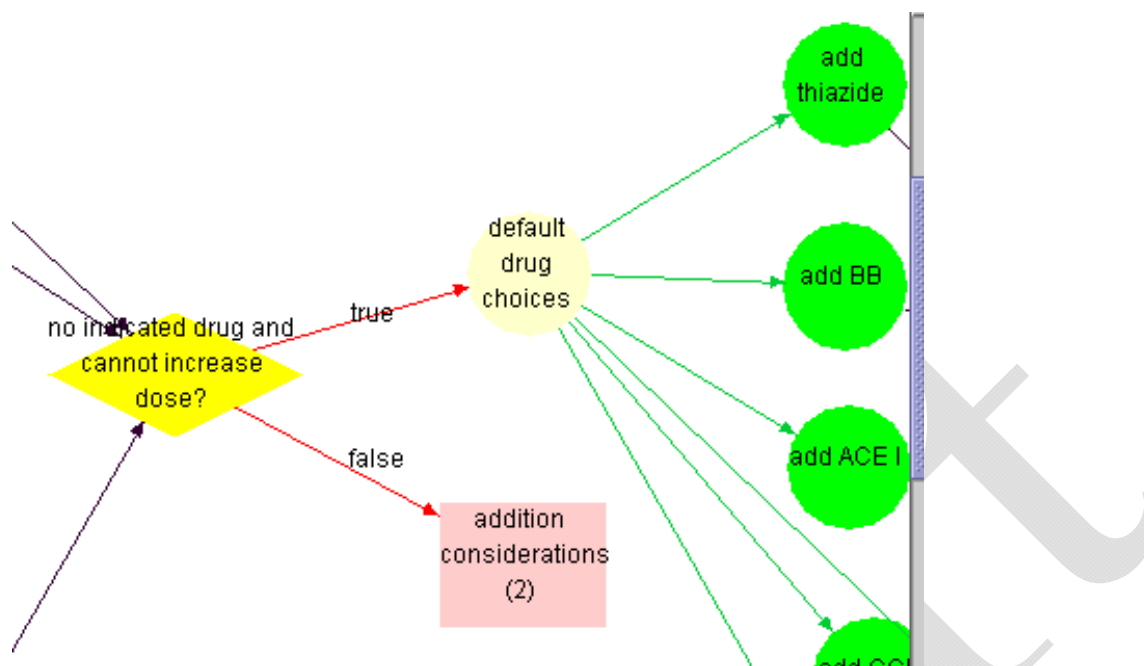


Figure 42 - Illustration of the use of Case_Step (yellow diamond). If an expression in Case Step evaluates to true (i.e., no drug has been recommended based on indications), the Guideline Interpreter evaluates the alternatives in the *default drug choices* step (yellow oval). If the expression evaluates to false, a scenario (*addition considerations (2)*) is used to generate appropriate messages. The red arrows branching from the case step (labeled *true* and *false* in the diagram) are instances of the Case_Selection class shown in Figure 43 - Case_Selection instance. It specifies that if the expression in the source (First Object) evaluates to the value in the Value slot, then take the step in the target (Second Object). Figure 43.

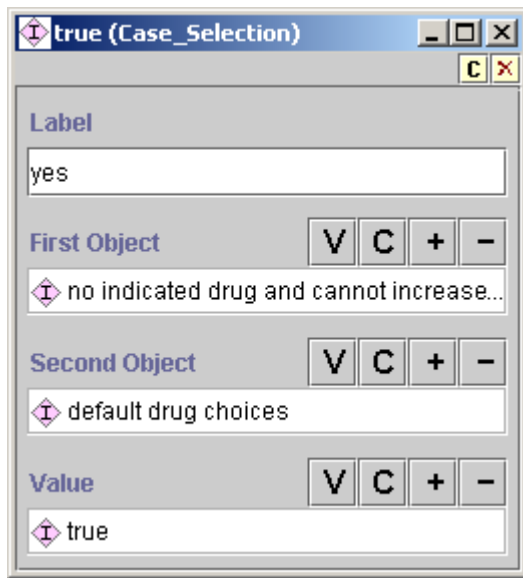


Figure 43 - Case_Selection instance. It specifies that if the expression in the source (First Object) evaluates to the value in the Value slot, then take the step in the target (Second Object).

Choice Step and Action Choice

Choice_Step and the associated Action_Choice are instances that model non-deterministic choices (i.e., more than one choice can be presented to the end user). An instance of the Choice_Step class (Figure 44) contains only links to the previous step and to the alternatives in the choice (*branches* slot, displayed as Branches). An Action_Choice instance (Figure 45) contains decision criteria (strict rule-in and strict rule-out conditions) for determining a preference for actions specified in the *actions* slot (shown as Actions). If a strict rule-out condition evaluates to true, then an alternative is rejected. If the strict rule-out condition is false or unknown and a strict rule-in condition evaluates to true, then the alternative is marked as preferred. If neither evaluates to true, then the preference for the choice can be determined by a default preference associated with the action choice. The current Guideline Interpreter is configured so that, if there is no default preference when neither *strict_rule_in* nor *strict_rule_out* conditions evaluate to true, the action choice and subsequent steps are not further pursued.

no-drug-therapy-choices (Choice_Step)

Label V C + -
no-drug-therapy-choices

Branches V C + -
 not on drug therapy, continue life style
 not on drug therapy, consider adding
 Beta blockers secondary prevention
 ACE I secondary prevention

Previous Scenarios V C + -
 not on drug therapy

Figure 44 - An instance of Choice_Step. It contains only links to the previous step and alternatives in the choice (Branches slot).

not on drug therapy, consider adding drug (Action_Choice)

Label V C + -
not on drug therapy, consider adding drug

Default Preference
 [Dropdown menu]

Followed By V C + -
 no indicated drug?

Rule In Condition V C + -
 [Empty text field]

Rule Out Condition V C + -
 [Empty text field]

Start Constraint V C + -
 [Empty text field]

Strict Rule In Condition V C + -
 BP not adequately controlled based or...

Strict Rule Out Condition V C + -
 BP adequately controlled

Actions V C + -
 evaluate new drug to prescribe
 Adding drug, DBP<100 and SBP<160
 Adding drug, SBP>=160 or DBP>=100
 Adding drug, marginally elevated BP

Reference V C + -
 [Empty text field]

Figure 45 - Example of Action_Choice. Decision criteria (strict rule-in and strict rule-out conditions) determine preferences for actions specified in the Actions slot.

III.3.3. Action Specification

Figure 46 shows the list of possible action specifications (Action_Specification) with instances that individuals building the ATHENA Knowledge Base can add to the *actions* slot of Action_Choice instances (shown as Actions in Figure 45). The action specifications can be placed into two broad categories: those generating textual output (shown as messages in the ATHENA Client) and those involving adding, deleting, or substituting activities (drug recommendations).

Messages

In the first category of Action_Specification are those action specifications in the EON Guideline Model that essentially send textual messages to the user. They include On_Screen_Message and its subclasses, and also—because of the lack of complete implementation in the current ATHENA version of the Guideline Interpreter—Procedure, Collect_Patient_Data (and its subclasses), Present_Data, Schedule, Referral, Acute_Prescription, Modify_Activity, Step_Down_Activity, Step_Up_Activity, Stop_Activity, and Evaluate_Current_Activity classes. If instances of these classes are used (e.g., Schedule and Referral), the current Guideline Interpreter simply inserts the *label* and *description* slot values of these instances into the recommendation. In order to issue a scheduling recommendation, an instance of the Schedule class should be created. This should be done in case the Guideline Interpreter changes in the future, and also because the ATHENA GUI may be able to use the information that the scheduling recommendation is an instance of Schedule class.

Instances of On_Screen_Message class (Figure 47) contain the text of any message that will be part of a patient advisory displayed by an ATHENA Client. The class also has a message_type slot (shown as Message Type) whose possible values are specified as classes in the Value_Type hierarchy (Section III.2.1). The message types are used by the ATHENA Client to determine the display location of the messages.

Instances of the Conditional_On_Screen_Message have a single rule-in condition. If the rule-in condition is true, the message is added to the advisory.

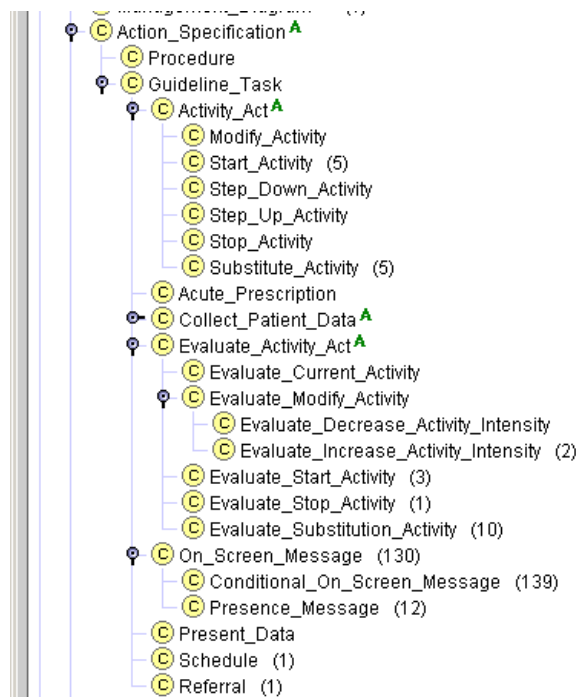


Figure 46 - Different types of Action_Specification in the EON Guideline Model

Figure 47 - Example of Conditional_On_Screen_Message

The Presence_Message class of messages responds to limitations in the previously described message classes. The class makes it possible both to check for the existence of certain conditions (e.g., allergy to a particular drug or the “active and not refilled” status of a medication) and to include the names of these conditions in dynamically generated messages (rather than the static

text strings of both On_Screen_Message and Conditional_On_Screen_Message). There are two usages of Presence_Message, both of which meet the need for dynamically generated messages.

The first usage is shown in Figure 48, where the instance checks for the presence of medications for which drug_name is a subclass of Antihypertensive_Agents, and assessed_status (shown as Assessed Status) is “active and not refilled.” If there are such Medication instances, then the message “Prescription(s) for <drug_name> has(have) NOT BEEN FILLED RECENTLY” will be generated. The drug names of the queried medication instances will substitute the <drug_name> phrase.

Figure 48 - An instance of Presence_Message, showing a query for medications that have the property of “active and not refilled”

The second usage of Presence_Message is shown in Figure 49. The instance specifies a query for instances of Adverse_Reaction, where the domain_term (shown as Domain Term) is a subclass of Alpha_blockers. The Guideline Interpreter will prefix the following string to the message specified in the message body: “The patient had *reaction1, reaction2...* reported as an adverse reaction/allergy to *allergic substance*.” The italicized string will be replaced with actual reactions and the allergic substance found in the patient data. For example, for the instance specified in Figure 49, the generated string might be: “The patient has dizziness and nasal congestion reported as an adverse reaction/allergy to terazosin. Use clinical judgment to determine its relevance in this patient.”

The prefixed string is generated in the Guideline Interpreter code and cannot be changed without changing and recompiling the Guideline Interpreter.

ADR to alpha blocker (Presence_Message)

Label
ADR to alpha blocker

Message Type
[V] [C] [X]

☒ **Presence**

Entry Type
Adverse_Reaction

Domain Term
Alpha_blockers

Assessed Status

Message
Use clinical judgment to determine its relevance in this patient.

Figure 49 - An instance of Presence_Message, showing a query for an adverse reaction to alpha blockers

Drug Recommendations

In the second category of Action_Specification are recommendations to add, delete, substitute, or modify prescribed drugs (Figure 50). Subsection III.3.4 describes the properties of drug classes and generic drugs that are used in generating drug-related recommendations. Below, the action specifications of this category are described.

Activity_Act
 Start_Activity
 Substitute_Activity
Evaluate_Activity_Act
 Evaluate_Modify_Activity
 Evaluate_Decrease_Activity_Intensity
 Evaluate_Increase_Activity_Intensity
 Evaluate_Start_Activity
 Evaluate_Stop_Activity
 Evaluate_Substitution_Activity

Figure 50 - List of actions that modify drug prescriptions

The difference between Activity_Act and Evaluate_Activity_Act is that, for both Start_Activity (Figure 51) and Substitute_Activity (Figure 52), the drug class or generic drug to add or

substitute is always marked as preferred in the recommendation, regardless of whether the drug class or generic drug has a specific indication or not. Thus, ACE Inhibitor may have no specific indication (i.e., Diabetes and Proteinuria/Renal Manifestations or Heart Failure), but the Guideline Interpreter may nevertheless recommend it to be added in some circumstances (e.g., when using it as the default drug to add). In subclasses of Evaluate_Activity_Act, on the other hand, the Guideline Interpreter evaluates the indications, contraindications, drug partners, and complications of the selected drug classes, but it does not mark any of the evaluated drug classes as preferred. Instead, the results of evaluating these properties of selected drug classes are made available as part of an advisory.

For example, an instance of Start_Activity to start a Cardioselective Beta Blocker may, when applied to a particular patient, result in the following recommendation:

Add evaluation

activity_to_start: Cardioselective Beta Blocker
 relative_contraindications: Obstructive_Pulmonary_Disease(492.8, 496.)
 preference: preferred
 messages:
 name: add beta blocker, obstructive pulmonary disease
 text: Cardiovascular benefits of beta blocker therapy may outweigh the increased risk of bronchospasm in this patient
 action_spec_class: On_Screen_Message

An Evaluate_Activity_Act to start Cardioselective Beta Blocker, when applied to the same patient case, will result in the same recommendation, except that the *preference* attribute will not have a value. (The current ATHENA GUI does not display this preference information.)

Figure 51 - An example of Start_Activity. The excluded_subactivities/Excluded Subactivities slot is not used.

Figure 52 - An example of Substitute_Activity

In Substitute_Activity, the drug to be deleted is specified using a PAL_Query (see III.4.2 for details about PAL query). Figure 53 shows an instance of PAL_Query that references a PAL query expression (not shown). The query expression, upon evaluation, returns an instance of Medication for verapamil, if verapamil is one of the active prescriptions of the patient. Then, based on the value of key_slot (shown as Key Slot), returns the drug_name of that instance (i.e., the string *verapamil*).

Figure 53 - PAL_Query that returns the drug_name of the query for verapamil medication

Among the subclasses of Evaluate_Activity_Act is Evaluate_Start_Activity (see Figure 54). For each instance of Drug_Usage enumerated in the alternatives slot, the Guideline Interpreter will

evaluate its indications, contraindications, drug partners, drug partner to avoid, complication factor, formulary preferred drug,¹⁰ and collateral actions and make them available in the advisory.

Figure 54 - Example of Evaluate_Start_Activity

Figure 55 shows an example of Evaluate_Stop_Activity, which is another subclass of Evaluate_Activity_Act. Given the example, the Guideline Interpreter will:

1. Find all instances of Medication (activity_class/Activity Class slot) whose domain term (domain_term/Domain Term) is subsumed by the Antihypertensive_Agents slot.
2. For these Medication instances, find instances of Drug_Usage (activity_spec/Activity Spec slot) whose Drug_Class_Name slot value subsumes the drug names of the Medication instance.

¹⁰ A formulary is a list of prescription drugs that a health-care institution or health plan has approved for use by doctors.

3. Evaluate the indications, contraindications, drug partners, drug partners to avoid, complication factor, formulary preferred drug, and collateral actions of the drug usage as part of the recommendation to discontinue this medication.

Using a PAL query is the alternative to this complicated sequence of operations.

Figure 55 - An instance of Evaluate_Stop_Activity

Figure 56 shows a use of Evaluate_Substitute_Activity that does not use a PAL query to specify which drug is to be deleted in the substitution. The usage is exactly like Evaluate_Stop_Activity (Figure 55), except that—in addition to finding the Drug_Usage instance for the medication to be deleted (e.g., furosemide) and evaluating its properties—the Guideline Interpreter evaluates, for the drug class to be added (e.g., thiazide diuretics), the indications, contraindications, drug partners, drug partners to avoid, complication factor, formulary preferred drug, and collateral actions. The Guideline Interpreter returns a substitution record for an alternative drug class only if the drug class (i.e., instances of Drug Usage) has at least one compelling indication.

The screenshot shows a software window titled "substitute thazide for furosemide (Evaluate_Substitution_Activity)". The window is divided into several sections:

- Label:** A text field containing "substitute thazide for furosemide".
- Description:** An empty text field.
- Activity Class:** A text field containing "Medication".
- Activity Spec:** A section with a "V" button, a "+" button, and a "-" button. Below them is a text field containing "Drug_Usage" with a yellow circle icon.
- Domain Term:** A section with a "V" button, a "+" button, and a "-" button. Below them is a text field containing "furosemide" with a yellow circle icon.
- Activity Spec Key:** A section with a "V" button, a "C" button, a "+" button, and a "-" button. Below them is a text field containing "Drug_Class_Name" with a blue square icon.
- Activity To Stop:** A section with a "V" button, a "C" button, a "+" button, and a "-" button. Below them is an empty text field.
- Alternatives:** A section with a "V" button, a "C" button, a "+" button, and a "-" button. Below them is a text field containing "Thiazide Diuretic" with a pink diamond icon.
- Reference:** A section with a "V" button, a "C" button, a "+" button, and a "-" button. Below them is an empty text field.
- Responses:** A section with a "V" button, a "C" button, a "+" button, and a "-" button. Below them is a list of responses, each preceded by a pink diamond icon:
 - patient non-compliant
 - BP not representative
 - different drug change
 - drug not tolerated
 - missing diagnosis
 - Don't agree with drug recommendation
 - No drug change
 - retitration

Figure 56 - Example of Evaluate_Substitution_Activity usage, without employing a PAL query

Figure 57 shows an example of using a PAL query to find the drug to delete in the substitution evaluation. The PAL query (not shown)¹¹ finds antihypertensive medications that are to be discontinued. Then the Guideline Interpreter finds the Drug_Usage instances associated with the drugs to be discontinued¹² and evaluates their properties. Finally, the Guideline Interpreter evaluates the properties of the drug classes to be added in potential substitution. The Evaluate_Substitution_Activity results in recommendations to discontinue some medications and start others. Because of the current implementation, Evaluate_Substitution_Activity should have Activity Spec "Drug_Usage", Activity Class "Medication", Activity Spec Key "Drug_Class_Name", and Alternatives instances of Drug Usage. Again, the Guideline Interpreter returns a substitution record for an alternative drug class only if the drug class (i.e., instances of Drug Usage) has at least one compelling indication. (Substitute Activity will returns a substitution recommendation record regardless of indications or contraindications of the drugs

¹¹ The PAL query should have "drug_name" in the *key_slot* so that the query will return a list of drug names to delete.

¹² These Drug_Usage instances are those whose Drug_Class_Name slot values (e.g., ACE Inhibitor) subsume the names of the drug to be discontinued (e.g., lisinopril).

involved, although the recommendation record will have evaluation information about each drug.)

Figure 57 - Example of Evaluate_Substitution_Activity usage, employing a PAL query to determine which drugs should be substituted

Evaluate_Increase_Activity_Intensity (Figure 58) illustrates how to specify the evaluation of a drug dose for possible increase. The PAL query in the activities_to_modify/Activities To Modify slot returns the names of drugs that have a daily dose below the maximum level and do not have any contraindications or bad drug partners. The Guideline Interpreter then finds the Guideline_Drug instances (activity_spec/Activity Spec slot) whose generic_drug slot values match those of the medications. From the dose-level table in these Guideline Drug instances, the Guideline Interpreter determines the level to which the medications' daily doses should be increased. The current ATHENA GUI does not make use of this dose-level information.

consider increasing dose (Evaluate_Increase_Activity_Intensity)

Label
consider increasing dose

Description

Activity Class
Medication

Attribute
drug_daily_dose

Activity Spec V + -
C Guideline_Drug

Activities To Modify V C + -
I non-contraindicated drugs whose dos...

Activity Spec Key V C + -
S generic_drug

Excluded Activities V + -

Reference V C + -

Responses V C + -
 I patient non-compliant
 I BP not representative
 I drug not tolerated
 I different drug change
 I Don't agree with drug recommendation
 I missing diagnosis
 I No drug change
 I retitration

Figure 58 - Example of Evaluate_Increase_Activity_Intensity

III.3.4. Drug Usage and Guideline Drug Activities

Drug Usage

The Drug_Usage class contains declarative information about properties of a drug class, such as Cardioselective Beta Blocker (Figure 59). The drug_class_name slot links this drug-usage instance to the corresponding drug class in the medical concept model's medication hierarchy. The formulary_preferred_drug_in_class slot (shown as Formulary Preferred Drug in Class in the figure) links this drug-usage instance to the generic drugs that can be prescribed for this drug class.

Cardioselective Beta Blocker (Drug_Usage)

Label
Cardioselective Beta Blocker

Drug Class Name
Beta-Blockers-Cardioselective

Compelling Indications
☒ Hypertensive_Without_Comorbidities
☒ ISH on HCTZ and BP not controlled
☒ MI (secondary prevention) BP controlled
☒ MI (BP not controlled)

Absolute Contraindications
☒ Bronchospastic_Disease
☒ Heart block without pacemaker
☒ Unspecified Heart Block and no pacemaker

Drug Partners
☒ Thiazide_Diuretics

Side Effects

Complication Factor
☒ Heart_Failure

Formulary Preferred Drug Class
☒ atenolol
☒ metoprolol

Relative Indications
☒ Atrial_Tachycardia
☒ Angina
☒ Atrial_Fibrillation
☒ Hyperthyroidism

Relative Contraindications
☒ Depression
☒ Obstructive_Pulmonary_Disease
☒ sick sinus syndrome and no pacemaker
☒ On Amiodarone

Drug Partners To Avoid
☒ guanabenz
☒ guanfacine
☒ verapamil
☒ beta-blockers-ISA
☒ Beta-Blockers-non-cardioselective

Collateral Actions
☒ alpha blocker monotherapy rec ADD
☒ Add Beta blocker and DM
☒ Add Beta Blocker to Diltiazem (no MI)
☒ Add Beta blocker and PVD

Components

Figure 59 - Details of a Drug_Usage instance, showing the declarative properties of a drug class that can be used in the ATHENA DSS

Algorithm for evaluating a drug class

In evaluating a drug class to add or delete:

1. The Guideline Interpreter determines whether the patient has any of the compelling and relative indications, absolute and relative contraindications, drug partners to avoid, or complication factors that are specified in the drug-usage instance of the drug class.
2. If the patient has at least one complication factor, the Guideline Interpreter will not generate any recommendation concerning the specified drug.
3. If the drug class is not absolutely contraindicated and there is at least one compelling or relative indication, then—unless the patient is already taking a drug of this class—the Guideline Interpreter will include the drug class among the ones it recommends adding.
4. If the recommendation includes adding a drug class, the Guideline Interpreter checks the `formulary_preferred_drug_in_class` slot (shown as Formulary Preferred Drug in Class) to see whether there is a more specific formulary preferred drug. These drugs are instances of `Guideline_Drug` class (Figure 62). If there is only one guideline drug specified in the Formulary Preferred Drug in Class slot and it is not ruled out as a result of evaluating the rule-out condition, then that drug is used as the formulary preferred drug. If there is more than one formulary preferred drug that is not ruled out, then the system evaluates the rule-in criteria. If at least one guideline drug is ruled in, then the ruled-in drugs are considered to be preferred. If no drug is ruled in, then the system returns all drugs that are not ruled out as possibilities that a clinician can select.
5. After determining whether a drug should be added or deleted or have its dose increased, the system—as part of the evaluation of a drug class—assesses the `Collateral_Action` instances in the `collateral_actions` slot of a `Drug_Usage`.

Figure 60 shows an instance of `Collateral_Action` class. It contains a `mood/Mood` slot that specifies the context (`Recommend_Add`, `Recommend_Delete`, or `Recommend_Increase_Dose`) where the actions specified in the `actions/Actions` slot are applicable. The *actions* slot should contain instances of `Action_Specification` that generate messages (e.g., Figure 61). In ATHENA, these collateral action messages are displayed as drug-specific info-button messages (Subsection **Error! Reference source not found.**).

Add Beta blocker and DM (Collateral_Action)

Label
Add Beta blocker and DM

Mood
Recommend_Add A M

Actions
Add Beta blocker and DM

Figure 60 - An example of Collateral_Action associated with a drug class

Add Beta blocker and DM (Conditional_On_Screen_Message)

Description
Add Beta blocker and DM

Label
Add Beta blocker and DM

Message
Beta blockers may diminish symptoms of and recovery from hypoglycemia and should be used with caution in patients who are prone to hypoglycemia.

Message Type
Recommendation A

Rule In Condition
presence of diabetes mellitus

Reference

Figure 61 - An example of Message associated with a drug recommendation

Guideline Drug Activities

As described earlier in this subsection, instances of the Guideline_Drug class make it possible to specify preferred drugs in the formulary. The second role of Guideline_Drug is to encode dose information, such that the system can determine when to recommend increasing the dose of a drug. Figure 62 shows an instance of the Guideline_Drug class. It contains: a reference to the

generic drug metoprolol in the medication hierarchy, a table of dose-level ranges (dose_level_ranges/Dose Level Ranges slot), and the specification of the maximum recommended dose level (max_recommended_dose_level/Max Recommended Dose Level slot). The dose-level range table is a set of Range_Mapping_Entry instances that specifies the lower and upper dose limits for a nominal dose level (Figure 63). The system would recommend increasing the dose of a drug only if the current daily dose is in a range below the maximum recommended dose level.

Currently, the Guideline Interpreter ignores the starting_dose, dose_strength_unit, drug_usage, duration_constraint, prescribable_items, and collateral_actions slots of the Guideline_Drug class.

The screenshot shows a window titled "metoprolol (Guideline_Drug)". It contains several slots and controls:

- Label:** metoprolol
- Generic Drug:** metoprolol (with a yellow circle icon and a green 'A' icon)
- Starting Dose:** 50.0
- Dose Strength Unit:** mg (with a yellow circle icon and a green 'A' icon)
- Max Recommended Dose Level:** High_Dose (with a yellow circle icon and a green 'A' icon)
- Rule In:** presence of MI (with a yellow circle icon and a green 'A' icon)
- Rule Out:** (empty)
- Drug Usage:** (empty)
- Duration Constraint:** (empty)
- Collateral Actions:** (empty)
- Dose Level Ranges:** A table with columns: abstract_val..., lower_limit, upper_limit.

abstract_val...	lower_limit	upper_limit
Low_Dose	0.0	50.0
Medium_Dose	51.0	200.0
High_Dose	201.0	300.0
- Prescribable Items:** (empty)
- Reference:** (empty)

Figure 62 - An instance of Guideline_Drug class

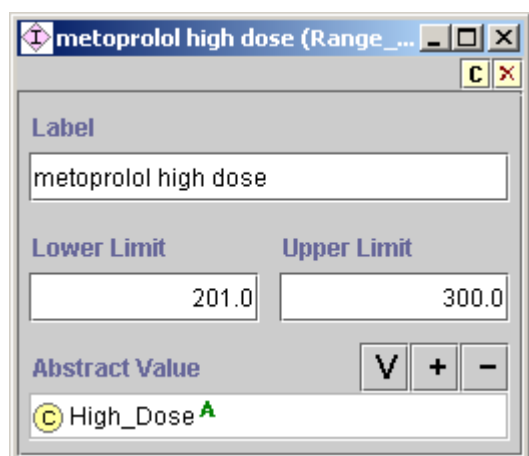


Figure 63 - An instance of Range_Mapping_Entry, showing the lower and upper limits of “high dose” for metoprolol.

III.4. Expressions

The primary mechanism through which the EON Guideline Model interacts with the patient data model and medical concept model involves decision criteria and other expressions written in the EON’s expression languages. These decision criteria and expressions are what make encoded guideline knowledge bases, like that of ATHENA, “computable” (i.e., they can be used to automatically match patient data to conditions in a guideline).

Guideline authors and developers can write expressions and decision criteria in the EON Guideline Model in one of two languages.¹³ Expressions and decision criteria in these two languages are represented as Protégé classes and instances (Figure 64).

¹³There is a third language, but it has never been used in the ATHENA and is mostly untested. In this language, instances of the Temporal_Query class hold ChronusII queries and expressions. The Guideline Interpreter simply passes the queries and expressions to ChronusII for evaluation.

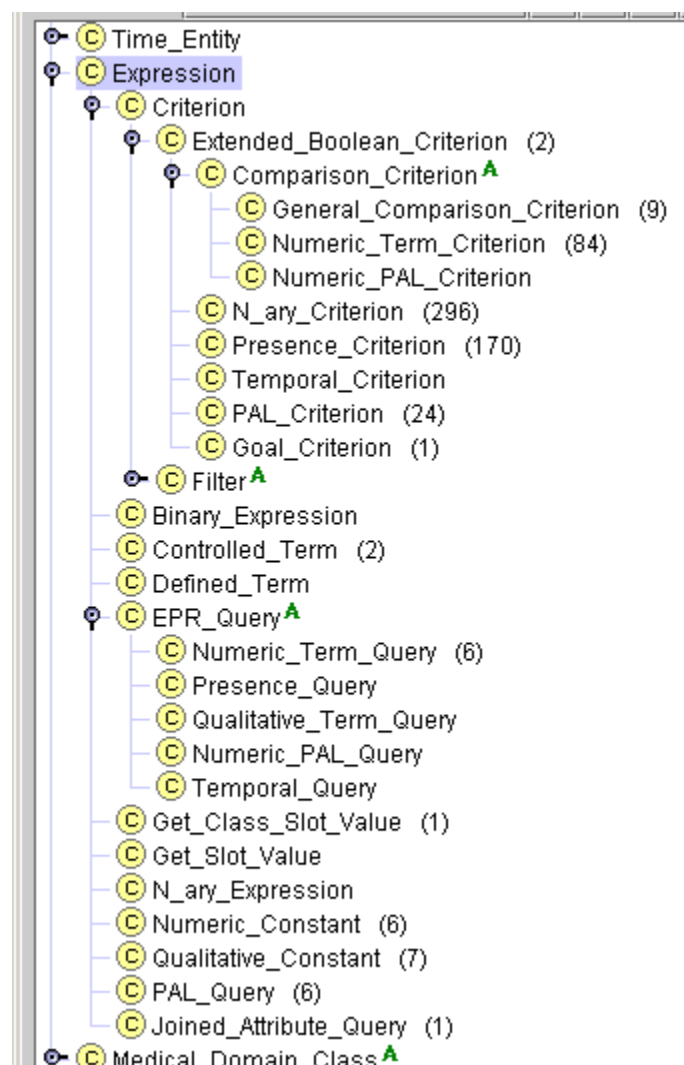


Figure 64 - Classes that make up expression languages in EON

III.4.1. *Template-based Expression Language*

The first expression language consists of a set of object templates, allowing a guideline author to encode common but relatively simple criteria by filling in forms generated by Protégé. Expressions in this language are instances of those subclasses of the Expression class that are not Numeric_PAL_Criterion, PAL_Criterion, Temporal_Criterion, Temporal_Query, and PAL_Query.

Consider the example: “(Presence of diabetes mellitus) and (Minimum creatinine within last two months is greater than upper limit of normal)”. This example can be encoded using a nested series of object templates described below.

Criteria encoded in this template-based language evaluate to *true*, *false*, or *unknown*. For example, in the absence of any serum creatinine result, the criterion that compares the serum creatinine value to its upper limit may evaluate to *unknown*. However, if a patient does not have a medical problem, this fact is usually not explicitly recorded. When checking for the presence or absence of medical problems specified in the medical concept model, the Guideline Interpreter assumes that, if a problem is not explicitly recorded, the patient does not have that problem. Accordingly, these criteria evaluate to *false* when there is no record of a medical condition specified in these criteria.

In most of the EON Guideline Model, a criterion makes a difference when it evaluates to *true*. For example, strict rule-out and strict rule-in conditions apply only when the conditions evaluate to *true*. Thus, in most cases, the *unknown* truth value¹⁴ has the same effect as the *false* truth value. In three places, however, *unknown* makes a difference:

1. In checking eligibility for guidelines, *unknown* has the same effect as *true*. In other words, a patient is not eligible for a guideline only if at least one eligibility criterion evaluates to *false* (see Section III.3.1).
2. If a goal criterion evaluates to *unknown*, then the Guideline Interpreter generates alternative recommendations based on assumptions that the goal is first satisfied and then unsatisfied (see Section III.3.1).
3. In a Case_Step, flow of control is based on the value of the expression specified in the step. If the value of the expression is *unknown*, and if there is a branch that matches the *unknown* value, then that branch will be taken. Otherwise, the Guideline Interpreter will stop following the path.

Below, selected classes in the template-based language are described in detail. The construction and usage of the classes other than those described are similar.

Presence Criterion

Consider: (*Presence of diabetes mellitus*). This expression can be encoded using an instance of the *Presence_Criterion* class Figure 65. *Presence_Criterion* allows the specification of the presence or absence of patient data model instances (e.g., *Note_Entry* instances) that correspond to domain terms from the medical concept model (e.g., *diabetes mellitus*).

The *period* (shown as *Period*) attribute specifies how far back to look for the record. In the absence of a value for the *period* attribute, the Guideline Interpreter looks at all past data available to it. The *mood* (shown as *Mood*) attribute is used only with Medication records, and is explained in Subsection III.2.2. The *filter* (*Filter*) attribute specifies additional conditions that the

¹⁴ In logic, a truth value is a value indicating to what extent a statement is true. In classical logic, a proposition can only be *true* or *false*. This guide is departing from the requirements of classical logic.

data being queried must satisfy (e.g., the daily dose of a medication is not at the maximum recommended level).¹⁵

The evaluation of a *Presence_Criterion* instance returns *true* or *false*. For a criterion with the *presence* (shown as Presence) flag checked, the criterion is true if and only if there exists an entry of the *entry_type* (Entry Type) that satisfies the *mood*, *period*, and *filter* constraints specified in the *Presence_Criterion* instance.

Figure 65 - An instance of *Presence_Criterion* that is checking whether there has ever been a note entry of diabetes mellitus diagnosis

General Comparison Criterion and Numeric Comparison Criterion

Next consider the condition: *Minimum serum creatinine within last two months is greater than upper limit of normal*. An individual maintaining the ATHENA Knowledge Base can encode this condition as an instance of the *General_Comparison_Criterion*. Figure 66 shows the *entry_type*, *domain_term*, and *valid_window* attributes, whose values are used by the Guideline Interpreter to get all numeric entry instances whose domain term is Creatinine and whose timestamp is *within last 2 months*. The aggregation operator, *minimum*, is used to identify the minimum among the queried values. The *value* (Value) attribute specifies that the minimum value has to be greater than the upper limit of normal of Creatinine. Following are explanations of each attribute:

- *Entry_type* (Entry Type) – The attribute can be *Numeric_Entry* or *Note_Entry*.
- *Domain_term* (Domain Term) – This is a class from the domain concept model.

¹⁵ The *filter* attribute was created before Protégé Axiom Language (PAL) criteria (Subsection III.4.2)—which allow the specification of complex conditions—were added to the EON Guideline Model. The *filter* attribute should not be used in future work.

- *Aggregation_operator (Aggregation Operator)* – Possible values are minimum, maximum, most recent, average, and count (the number of returned values).
- *Valid_window (Valid Window)* – The valid_window attribute specifies how far back to look for the value being queried. It is an instance of Relative_Time_Interval_Definite (Figure 67).¹⁶
- *Operator* – Possible values are >, >=, <, <=, =, eq, and neq; eq and neq are used to compare non-numeric values.
- *Assume_if_no_value (Assume If No Value)* – Possible values are no_assumption, assume_satisfied, assume_unsatisfied, and use_default. If this slot is not filled in, or if no_assumption is specified, evaluation of the criterion makes no assumption and the result may be *true*, *false*, or *unknown*. Assume_satisfied or assume_unsatisfied mean the criterion is assumed to be true or false, respectively, if there is no value for the domain term. Use_default means the default value specified in the default_value (Default Value) slot should be used. If no assumption is specified and there are no data for evaluating the criterion, *unknown* is the result of evaluating the criterion.

The screenshot shows a software interface for defining a comparison criterion. The title bar reads "creatinine > upper limit of normal (General_Comparison_Criterion)". The interface is divided into several sections:

- Label:** A text field containing "Minimum creatinine within last 2 months >".
- Entry Type:** A dropdown menu showing "Numeric_Entry".
- Domain Term:** A dropdown menu showing "Creatinine".
- Operator:** A dropdown menu showing ">".
- Aggregation Operator:** A dropdown menu showing "minimum".
- Assume If No Value:** An empty dropdown menu.
- Value:** A text field containing "Creatinine upper limit of normal".
- Default Value:** An empty text field.
- Valid Window:** A text field containing "within last 2 months".
- Mood:** An empty text field.

Each of these fields has a small "V" button and a "+" or "-" button next to it, likely for validation or editing.

Figure 66 - An instance of General_Comparison_Criterion, stating that the minimum creatinine value within the last two months is greater than the upper limit of normal

¹⁶ There are other types of time intervals and time points that can be specified using the classes in the Time_Entity hierarchy in the EON Guideline Model, but Relative_Time_Interval_Definite is the only one necessary for the ATHENA system. Other classes will not be documented in this manual.

The screenshot shows a dialog box titled "within last 2 months (Relative_Time_Interval_Definite)". It has a standard Windows-style title bar with a question mark icon and standard window controls. The dialog is divided into several sections. The top section has three labels: "How Many", "Polarity", and "Time Unit". Below "How Many" is a text box containing the number "2". Below "Polarity" is a dropdown menu currently showing "Before". Below "Time Unit" is a dropdown menu currently showing "month". The bottom section has two labels: "Label" and "Relative Time Point". Below "Label" is a text box containing the text "within last 2 months". Below "Relative Time Point" is a dropdown menu currently showing "Today". To the right of the "Relative Time Point" dropdown are four buttons: "V", "C", "+", and "-".

Figure 67 - An instance of *Relative_Time_Interval_Definite*, specifying the time interval “two months before today”

The *value* (Value) slot of a *General_Comparison_Criterion* should be an instance of *Expression*, whose subclasses are:

- *Binary_Expression* – subtraction or division
- *N_ary_Expression* – multiplication or addition of multiple arguments
- *Controlled_Term* – reference to controlled terminology; e.g., the Male class from the medical concept model
- *Numeric_Constant* – an instance with a number in its *value/Value* slot
- *Qualitative_Constant* – an instance with an arbitrary string in its *value/Value* slot
- *Get_Class_Slot_Value* and *Get_Slot_Value* – classes with instances specifying queries in Protégé to get a slot value either from a class or from an instance
- *EPR_Query* – Queries for patient data in the formats of the *EPR_Entities* (see Subsection III.1)

The definitions of most *Expression* classes are straightforward. Here, only *Get_Class_Slot_Value* and the types of *EPR_Query* are illustrated. Both *Get_Class_Slot_Value* and *Get_Slot_Value* are designed to query values from the Protégé classes and instances so they can be compared with patient data. Figure 68 shows an instance of *Get_Class_Slot_Value* that queries for the value of the *UpperLimitOfNormal* slot from the *Creatinine* class. (Recall that *Interval-Valued_AtomicTest_Metaclass* metaclass, described in Subsection III.2.1, allows class instances of this metaclass to specify upper and lower limits of normal.)

Creatinine upper limit of normal (Get_Class_Slot_Value)

Label
Creatinine upper limit of normal

Class Reference
Creatinine

Slot
UpperLimitOfNormal

Figure 68 - An instance of Get_Class_Slot_Value. It obtains the value of the UpperLimitOfNormal slot from the Creatinine class.

Among the subclasses of the EPR_Query class, Numeric_Term_Query and Qualitative_Term_Query may be useful. An instance of Numeric_Term_Query is shown in Figure 69. It specifies a query for the most recent value of the MD_Typical_Systolic_BP. MD_Typical_Systolic_BP is the systolic blood pressure a clinician enters into ATHENA to update an advisory. The query returns a numeric result.

An instance of Qualitative_Term_Query analogously has the same format but returns a string as its result.

MD_Typical_Systolic_BP (Numeric_Term_Query)

Label
MD_Typical_Systolic_BP

Entry Type
Numeric_Entry

Numeric Domain Term
MD_Typical_Systolic_BP

Mood

Aggregation Operator
most_recent

Period

Figure 69 - A Numeric_Term_Query that asks for the most recent value of the MD_Typical_Systolic_BP

N ary Criterion

Instances of `N_ary_Criterion`¹⁷ allow other criteria to be specified in Boolean combinations (*and*, *or*, and *not*). Figure 70 illustrates how to combine `Presence_Criterion` and `General_Comparison_Criterion`, discussed above, to form a conjunctive n-ary criterion using the *and* operator.

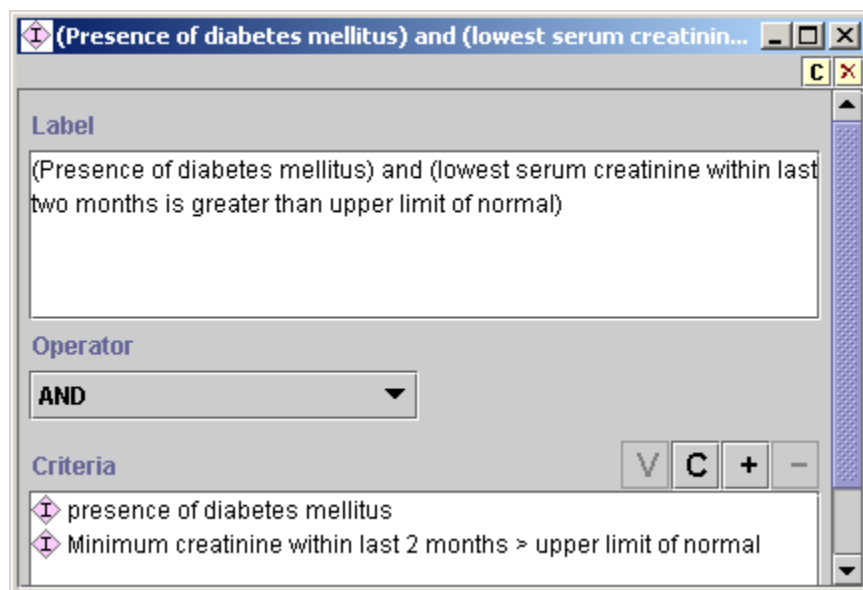


Figure 70 - An instance of `N_ary_Criterion` that requires both of its constituent criteria to be true, as indicated by the AND operator

Recall that criteria expressed in the template-based expression language can evaluate to *true*, *false*, or *unknown*. Accordingly, the truth tables used to derive the truth values of `N_ary_Criterion` take the *unknown* truth value into account. The following truth tables are used:

AND	true	false	unknown
true	true	false	unknown
false	false	false	false
unknown	unknown	false	unknown

OR	true	false	unknown
true	true	true	true
false	true	false	unknown
unknown	true	unknown	unknown

NOT	true	false	unknown
	false	true	unknown

¹⁷ By analogy with unary, binary, etc., “n-ary” means “any number of”. Here it refers to the fact that the *and* and *or* operators can take any number of arguments.

Other criteria templates in the EON Guideline Model:

- support simplified numerical value comparisons (Numeric_Term_Criterion, as illustrated in Figure 71), where the value slot can only be a number, instead of an instance; and
- allow a statement expressing that the goals associated with a guideline are being reached (Goal_Criterion as explained in Subsection III.3.1 and illustrated in Figure 36).

Figure 71 - An instance of Numeric_Term_Criterion. It is a simplified version of General_Comparison_Criterion.

III.4.2. PAL-based Expression Language

A guideline author can use the templates described so far to write most common decision criteria with little training. The criteria that can be written in this way, however, are not sufficiently expressive. Take the expression, “Presence of an authorized medication that is contraindicated by some medical condition”. It requires that, for each authorized medication, the Guideline Interpreter finds its contraindications from the ATHENA Knowledge Base and checks to see if there is any patient-data instance that suggests the presence of these contraindications.

Instead of trying to resolve such complex criteria procedurally, the Protégé Axiom Language (PAL)—a constraint language—is used to write them. The PAL constraint language implements a subset of first-order predicate logic written in Knowledge Interchange Format (KIF) syntax [5](Genesereth, 1991). It makes full use of Protégé’s frame-based knowledge model. For example, variables can range over instances of Protégé classes (e.g., in Figure 72, the variable ?current_med ranges over instances of Medication class). Attributes of classes (e.g.,

Absolute_Contraindications in Figure 72) can be used as a binary predicate to check that a constant or the value of a variable (e.g., ?contraindication) is a value of the slot. An attribute (e.g., domain_term in Figure 72) can also be used as a function of one argument that returns the value of the slot for an instance. It is not expected that domain experts untrained in logic will formulate these complex logical criteria. Full documentation of the PAL constraint language is available at the URL: <http://protege.stanford.edu/plugins/paltabs/pal-documentation/index.htm>.

Figure 72 - Example of a PAL expression

```
(defrange ?current_med :FRAME Medication)
(defrange ?problem :FRAME Note_Entry)
. . .
(exists ?current_med
  (and (patient_id ?current_med $patient_id)
    (exists ?med_class
      (and (subclass-of
        (drug_name ?current_med) ?med_class)
        (exists ?contraindication
          (and (Absolute_Contraindications
            ?med_class ?contraindication)
            (exists ?problem
              (and (patient_id ?problem $patient_id)
                (subclass-of
                  (domain_term ?problem)
                  ?contraindication)))))))))
```

The PAL language was originally designed for writing constraints on concepts and relationships in a Protégé knowledge base. In adapting it for writing decision criteria that involve patient data, it is necessary to introduce a variable (\$patient_id in Figure 72) that stands for the patient ID that, at run time, the Guideline Interpreter will use to replace the variable before evaluating a PAL criterion. Thus, an instance of PAL_Criterion (Figure 73), aside from a label, has two parts: (1) the case_variable slot (shown as Case Variable) that tells the Guideline Interpreter which string in the PAL expression stands for the patient ID, and (2) the slot that holds the PAL expression itself (such as the one in Figure 72).

The PAL constraint language has a query form that, instead of evaluating to *true* or *false*, returns instances that satisfy constraints written in PAL. Instances of PAL_Query, which has exactly the same format as PAL_Criterion, contain a PAL query (instead of a PAL constraint). PAL query is used in a number of places. For example, PAL query can be used to find all current medications, such that each of them is contraindicated by a problem a patient is experiencing. Furthermore, in Substitute_Activity, the drug to be deleted is specified using a PAL query. PAL query is also used to obtain the names of drugs whose doses need to be increased.

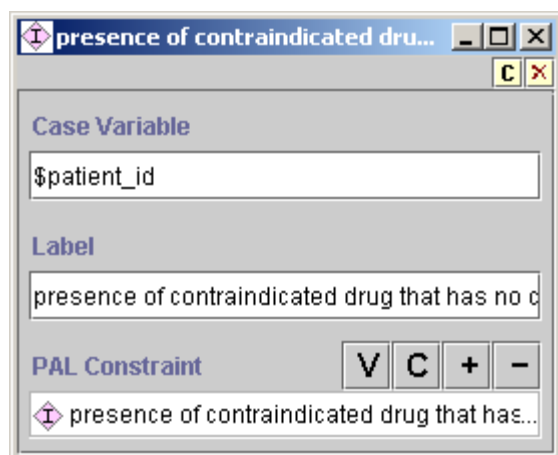


Figure 73 - An instance of PAL_Criterion

III.4.3. Summary

Criteria languages in the EON Guideline Model are all informed by the same patient data model and serve complementary requirements. The template-based language provides form-based templates that domain experts can easily use to write the common decision criteria. The logic-based PAL language adds additional expressiveness for writing specific types of complex criteria. Both the template-based and the PAL languages make use of the taxonomic hierarchies in the medical concept model to infer generalization/specialization relationships and to make abstractions based on definitions embedded in the hierarchies (e.g., the definition of *hyonatremia*).

A common evaluation-result interface unifies the usage of the template-based and PAL criteria. For criteria written in each language, the guideline execution engine invokes the appropriate criteria-evaluation engine. Each criteria-evaluation engine returns the result consisting of the criterion being evaluated, the truth value of the criterion as applied to available patient data, and annotations that can be used for explanation purposes.

IV. Guideline Interpreter's Use of the Knowledge Base

This subsection considers how the Guideline Interpreter generates a guideline-based advisory in response to requests from client programs. It will refer to the PCAServer (Protocol Compliance Adviser Server), which is a computer program that provides a collection of methods for client programs (such as ATHENA Client) to request and obtain advisories computed by the Guideline Interpreter.

IV.1. Overview

As described in Subsection **Error! Reference source not found.**, the PCAServer is implemented as a CORBA server. It has two external interfaces: the PCAServer interface and the PCASession interface. The PCAServer process is started on a central server machine with an initialization file that sets up a number of configuration parameters (see Table 1). A client program uses the PCAServer interface to open a client-specific PCASession. A client program can be the batch program that invokes the PCAServer to compute and store precomputed advisories, or it can be the ATHENA Client that runs on a clinical workstation. A client program interacts with the PCASession to obtain appropriate guideline recommendations (see Figure 74). The ATHENA Client, for example, may request updated advisories, precomputed advisories, or the computation of advisories when there are no precomputed advisories. All copies of the PCASession share the same guideline knowledge base loaded in the PCAServer, but otherwise are independent of each other. Thus, each copy of PCASession handles requests from a running ATHENA Client and keeps data for different patients separate from each other.

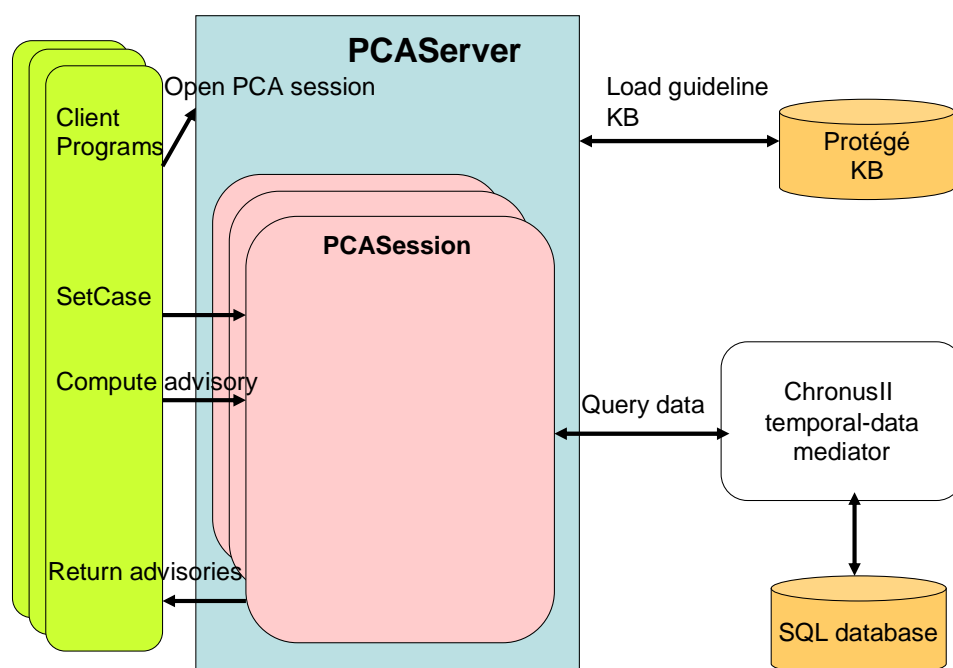


Figure 74 - The PCAServer and PCASession. Each ATHENA client calls the PCAServer in order to open its own PCASession, and interacts with the PCASession to get advisories. PCASessions share the same guideline knowledge base loaded in the PCAServer, but manage patient data specific to the requested cases.

Parameter Name	Possible Parameter Values	Comment
DATABASE	String (e.g., ATHENEONPA)	ODBC data source of

		patient data
SERVER_LOGFILE	Path to a directory (e.g., \\myPC\C\ATHENADSS\log\PA\)	Directory to which server log files should be written
SERVERKB	Path to Protégé project (e.g., \\myPC\C\ATHENADSS\domain_model\ATHENA.pprj)	The knowledge base to load
PCAINIFILE	Path to ini file (e.g., \\myPC\C\ATHENADSS\ini\ATHENA.ini	An argument in opening PCASession used to initialize ChronusII
PCAOUTPUTDIR	Path to a directory (e.g., \\myPC\C\ATHENADSS\output\PA\)	Directory from/to which precomputed advisories can be read/written

Table 1 - PCAServer initialization parameters

The PCASession interface has a number of operations that can be used by a client program such as ATHENA Client or the batch program (Table 2). For example, the setGuideline operation selects the guideline that should be used for the session, and setCase selects the patient for whom the guideline-based advisory should be generated. Other operations compute advisories, load precomputed advisories, send data updates to the server, and request updated advisories. A printData operation returns an HTML version of the data that are used to compute the advisories.

Operation Name	Parameters	Returned Value	Operation
setGuideline	guideline_name	None	Select the guideline to apply.
setCase	Patient id, session time	None	Load data from database, and set the time for which advisories are generated (default to current time).
computeAdvisories	None	Set of guideline service records	Given selected guideline and selected patient, compute advisories.
computeAndStoreAdvisories	Storage directory	Set of guideline service records	Precompute advisories.
loadPrecomputedAdvisories	Patient id, session time, storage directory	None	Load precomputed advisories from storage directory (session time not used).
containAdvisories	None	True/False	Check whether precomputed advisories are successfully

			loaded.
getAdvisories	None	Set of guideline service records	Get (usually precomputed) advisories.
resetAdvisories	None	None	Delete current advisories.
updateData	Patient data	None	Send a collection of patient data.
updateAdvisories	None	Set of guideline service records	Update advisories after data updates.
printData	None	String	Return a listing of patient data in HTML form.
setDummyCase	None	None	Set up a dummy patient case in order to generate advisories based purely on data entered through a user-interface.

Table 2 - Operations available in the PCASession interface

PCASession provides operations, such as computing and loading precomputed advisories, which are custom-tailored to the requirements of the ATHENA DSS. The EON guideline execution architecture allows such adaptations by providing a set of generic services that the PCASession can use to implement the application-specific functionalities. These generic services include, for example, evaluating eligibility criteria, testing guideline goals, deriving abstractions about patient states, and traversing a clinical algorithm to evaluate evidence-based preferences for alternatives in management decisions. The generic services, based on the capabilities of the EON Guideline Model, are defined by a separate interface that is implemented by a program referred to as the core EON guideline execution engine (CEGEE). It would go beyond the scope of the ATHENA manual to describe the CEGEE.

The Guideline Interpreter converts patient data into:

1. categorical conditions or interventions (e.g., atrial fibrillation and ACE inhibitor) whose presence or absence can be tested using `Presence_Criterion` (see Subsection III.4.1),
2. qualitative data (e.g., proteinuria with values of 1+, 2+, etc.) that have a term and a nominal value, and that are used in `General_Comparison_Criterion`,
3. numeric data that have a term and a numeric value (e.g., that can be tested using `Numeric_Term_Criterion` or `General_Comparison_Criterion`),
4. demographic information (e.g., sex) that can have a nominal value (e.g., male) or a numeric value (e.g., 56 years old), and
5. allergy data indicating a substance (often drugs) to which the patient is allergic.

The following subsections give a sense of the guideline advisories and their relationship to the ATHENA Knowledge Base. They describe the structure of the advisories and how the PCASession's `computeAdvisories` operation generates guideline recommendations for the ATHENA Client.

IV.2. Structure of Guideline Advisories

The PCASession delivers to the client program guideline advisories consisting of a set of Guideline Service Records. Each Guideline Service Record is composed of:

- *an assumption* – in the form of a goal criterion and its assumed status (*true*, *false*, or *none* when no assumption is made);
- *a collection of conclusions* – in the form of parameter/value/justification, about the patient (e.g., eligibility status and patient characteristics, such as JNC risk group classification);
- *a collection of chosen scenarios* – e.g., patient in the scenario of receiving two anti-hypertensive medications;
- *a collection of guideline goals* – e.g., target BP=135/85 because of Diabetes Mellitus;
- *a collection of activities* – evaluated in terms of their indications, contraindications, interactions, and side effects (e.g., evaluation of adding the ACE inhibitor drug class);
- *details of the actions associated with a decision point* – e.g., onscreen messages, evaluating drugs to add, and dose increases; and
- *a collection of decision points* – each of which containing a ranked list of alternative actions. The decision points and their associated actions are based on the choice and branch nodes in the clinical algorithm and consultation guidelines. The result of evaluating (strict) rule-in and (strict) rule-out criteria determine the ranking of the alternatives. (For example, for a patient taking one antihypertensive agent, the alternatives include intensifying treatment, staying with the same drug regimen, and substituting another drug.)

When there is no blood pressure in the records, the advisories will be two sets of the Guideline Service Records, one for the assumption that the target blood pressure is satisfied and one for the assumption that it is not.

Guideline_Service_Records are the recommendations that the Guideline Interpreter delivers to client programs, which present the recommendations in format appropriate for the user interacting with the client programs. ATHENA Client, for example, presents the recommendations in a graphical user interface described in Subsection **Error! Reference source not found.** The batch program that precomputes the advisories (Subsection **Error! Reference source not found.**), on the other hand, generates an HTML document that shows the recommendations in textual format. The HTML view the recommendations is designed to facilitate the debugging of the output of the Guideline Interpreter.

The following examples illustrate the structure of a Guideline Service Record. Following each example, in boxed figures, is the HTML output based on the recommendation structure described in the example.

EXAMPLE of a Guideline Service Record – Part 1

Shown first are how the assumption, conclusions, chosen scenarios, and goals are represented, for a sample patient, in the Guideline Service Record.

Assumption: none¹⁸
Conclusions: Eligible for JNC-VI Hypertension Guideline: true
Risk Group: C
Scenario Choice: on one anti-hypertensive drug
Guideline Goals: goals: SBP < 130 and DBP < 85
justification: presence of diabetes, heart failure or renal
insufficiency
Goal state: failed evaluation record of goal criteria

¹⁸ Assumption is *none* because, in this example, blood pressure data exist in ATHENA and, therefore, no assumption is made about whether the guideline goal is achieved.

Patient classification:

- JNC-VI Hypertension Guideline:true[because *Eligibility criteria* evaluate to **true**(*presence of diagnosis of hypertension && absence of renovascular disease && no diagnosis of pregnancy && Creatinine(1.3/2002-8-16) && Absence of Secondary Hypertension && absence of spinal cord injury && absence of narcolepsy && Not taking cyclosporine && Not taking spironolactone && Not taking minoxidil && absence of renovascular disease && absence of IHSS && Absence of Ascites && Not off guideline && Age(62.0/1999-12-23) && Antihypertensive_Agents && not taking tacrolimus && absence of transplant recipient*)]
- Risk Group C (presence of TOD/CCD or DM):[because *presence of TOD/CCD or DM* evaluate to **true**(*presence of TOD/CCD*)]

Scenario choice: on one anti-hypertensive drug

Goal: SBP < 130 and DBP < 85(presence of diabetes, heart failure or renal insufficiency)

Reached goal? failed(Treatment Systolic BP(142/DB Systolic BP))

Figure 75 HTML view of the parts of a Guideline_Service_Record that contains the PCAServer's conclusions about a patient (Patient classification), the scenario and goal chosen for the patient, and whether the patient reached the goal.

EXAMPLE of a Guideline Service Record – Part 2

The following shows a sample of three guideline decision points. The first one (*on medication: things to check*) is derived from the consultation guideline *on medication*. Each decision point has a set of associated actions (e.g., *Warning pregnancy female<50yrs* and *women >60 and menopause*). Each action has a preference (e.g., *ruled_out* or *preferred*) that is computed from the condition associated with the action. The result of computing the condition is detailed in the justification part (e.g., *not female less than 50 years*). The action_specification portion of the action contains details of the action (e.g., specific onscreen messages or increasing dose).

1. on medication: things to check

- * Warning pregnancy female<50yrs
preference: ruled_out
justification: not female less than 50 years
action_specification - (detailed on-screen message)
- * women >60 and menopause
preference: preferred
justification: rule in criterion *female >=60* evaluate to true
action_specification - (detailed on-screen message)...

2. one-drug-therapy-choices

- * Blood pressure not adequately controlled; intensify drug treatment
preference: preferred
justification: BP not controlled
action_specification - (detailed conditional on-screen message)

- * continue with one-drug regimen
preference: ruled_out
justification: BP not controlled...
 - * on one drug, consider substitution
preference: ruled_out
justification: BP not controlled...
3. step up choices
- * Increasing dose
preference: preferred
justification: drug dose not at maximum
action specification - (Evaluate Increased Activity Intensity to evaluate possibility of increasing dose)
 - * evaluate new drug to prescribe
preference: preferred
justification: always true
action specification - Evaluate Start Activity to evaluate various drug classes...

Only actions that are preferred are shown on the HTML output. Accordingly, for the examples shown above, the text in Figure 76 will appear on the HTML output.

Action Choices

- **assumption women >60 and menopause preferred**(rule in criterion *female >=60* evaluate to **true** because *Sex(Female) && Age(62.0/1999-12-23))*) We assume women older than 60 years are postmenopausal.
- **... (other messages)**
- **Blood pressure not adequately controlled; intensify drug treatment preferred**(strict rule-in condition *BP not adequately controlled based on most recent BP* evaluate to **true** because *BP not adequately controlled based on most recent BPTreatment_Systolic_BP(142/DB_Systolic_BP))*) Consider INTENSIFYING drug treatment: BP ELEVATED based on most recent available BP.
- **Increasing dose preferred**(strict rule-in condition *dose level not at maximum and if taking diuretics, not hypokalemia* evaluate to **true** because *thiazide diuretics not being used && there exists non-contraindicated drug not at maximum dose*))
- **evaluate new drug to prescribe preferred**(strict rule-in condition *true* evaluate to **true**))

Figure 76 HTML view of the parts of a Guideline_Service_Record that contains the PCAServer's conclusions about actions to be chosen and messages associated with the

EXAMPLE of Guideline Service Record – Part 3

There are four varieties of activity evaluation: add, delete, substitute, and change attribute. Following are brief examples of change attribute and add evaluations. The structure of delete evaluation is similar to that of add, and substitute evaluation is a combination of add and delete evaluations.

Change attribute evaluation

```
name: lisinopril
attribute_name: daily_dose
change_direction: up
messages: none
side_effects: none
```

Add evaluation

```
activity_to_start: Cardioselective Beta Blocker
relative_indications: Coronary_Artery_Disease(412., 414.9)
relative_contraindications: Obstructive_Pulmonary_Disease(492.8, 496.)
messages:
- name: add beta blocker, obstructive pulmonary disease
- text: Cardiovascular benefits of beta blocker therapy may outweigh the
      increased risk of bronchospasm in this patient
- action_spec_class: On_Screen_Message...
```

The corresponding HTML text is shown in Figure 77, below.

If changing attribute, consider:

- lisinopril(change drug_daily_dose: up)

Cardioselective Beta Blocker (atenolol)(based on ATHENAINSTANCE_00046)

- Relative indications: Coronary_Artery_Disease(412., 414.9)
- Relative contraindications: Obstructive_Pulmonary_Disease(492.8, 496.)
- add beta blocker, obstructive pulmonary disease(Cardiovascular benefits of beta blocker therapy may outweigh the increased risk of bronchospasm in this patient.)

Figure 77 The HTML output showing the drug recommendation portion of the Guideline_Service_Record

IV.3. Generation of Guideline Advisories

This subsection describes algorithms implemented by the `computeAdvisories` operation of `PCASession`.

The client program uses the `setGuideline` and `setCase` operations to indicate which guideline to apply to the patient and which patient case to load. The algorithm implemented in `computeAdvisories` involves the following steps (also see Figure 78):

1. Invoke the core EON guideline execution engine (CEGEE) to determine whether the patient satisfies the eligibility criteria of the guideline. If the result is *unknown* because of lack of sufficient data or is *true*, guideline advisories will be generated for the patient case.
2. Check to see if goals of the guideline have been met.
3. If the result of goal evaluation is *unknown*, `computeAdvisories` is programmed to make alternative assumptions. It will generate advisories assuming the goal has been met, and advisories assuming the goal has not been met.¹⁹ If the result of the goal evaluation is either *true* or *false*, then no assumption is made.
4. For each set of advisories, the `computeAdvisories` operation first asks the core EON guideline execution engine to evaluate a set of patient characteristics (e.g., hypertension risk groups). These patient characteristics are abstractions of the patient state that are computed even if they are not required for computing the rest of the guideline advisories.
5. To compute the rest of the advisory, the core EON guideline execution engine traverses the clinical algorithm of the guideline. It determines the starting point of traversal by evaluating the scenarios of the clinical algorithm (e.g., patient taking one anti-hypertensive agent). The CEGEE evaluates the preconditions of all scenarios, and presents the results for the calling program (`computeAdvisories` in this case) to make a selection.²⁰
6. The `computeAdvisories` method was programmed to make automatic selections based on the results of precondition evaluations. It selects the first scenario whose precondition evaluates to true.²¹ Starting from the initial scenario, the core EON guideline execution engine traverses the clinical algorithm. For each decision point that the CEGEE reaches in its traversal, it creates a record of guideline decision structure—as shown in Subsection IV.2.—and fills in the various slots of that record. For each action choice alternative at a decision point, it evaluates the associated rule-in and rule-out criteria. The result of these evaluations is stored in a justification record, and a preference such as

¹⁹ If a guideline can set more than one goal for a patient, the implementers of ATHENA DSS will have to decide what assumptions to make and reprogram the `PCASession` code. The core EON guideline execution engine can accept any number of and any combination of assumptions.

²⁰ The core EON guideline execution engine can be adapted to different usages. Instead of having preprogrammed responses to the choices presented by the execution engine, for example, another system may involve human users making selections.

²¹ Thus, it is important that all starting scenarios be mutually exclusive.

preferred, *neutral*, or *rule-out* is computed for each choice. The justification record refers to the evaluated guideline criteria, the patient data used, and assumptions, if any, made in evaluating the criteria. The *actions* slot associated with each action choice (see Subsection III.3.2) defines the set of actions to be performed if the choice is selected (see item #7, below).

7. The computeAdvisories operation is programmed to make automatic selections based on the preferences determined through the rule-in and rule-out criteria. (An action choice is selected only if its preference rating is preferred, which occurs only if a strict rule-out criterion evaluates to *false* or *unknown*, and a strict rule-in criterion evaluates to *true*.)
8. For each choice that is selected by computeAdvisories, the CEGEE continues the traversal of the clinical algorithm, presenting additional choices to the computeAdvisories program if there are more decision points in the algorithm.
9. After the core EON guideline execution engine finishes traversing the clinical algorithm and evaluating the action choices selected by the computeAdvisories operation, computeAdvisories returns advisories to its calling program in the form of completed Guideline Service Record structures (see Subsection IV.2).

The computeAdvisories operation of PCASession is the basic mechanism for generating an ATHENA advisory. The computeAndStoreAdvisories operation is implemented by calling the computeAdvisories operation. It then stores the advisories and the data used to compute the advisories as a Java serialization file. The updateAdvisories operation deletes previously computed results and calls computeAdvisories to generate new advisories. The loadPrecomputedAdvisories operation simply loads the precomputed advisory file, restores the Guideline Service Record structures, and returns them to the client.

In Figure 78, each highlighted rectangle represents the invocation of a service provided by the core EON guideline execution engine. The continueComputeAdvisory circle represents a sub-algorithm that may be invoked with assumptions of the guideline goal either being satisfied or not (rectangles labeled “3.” in Figure 78).

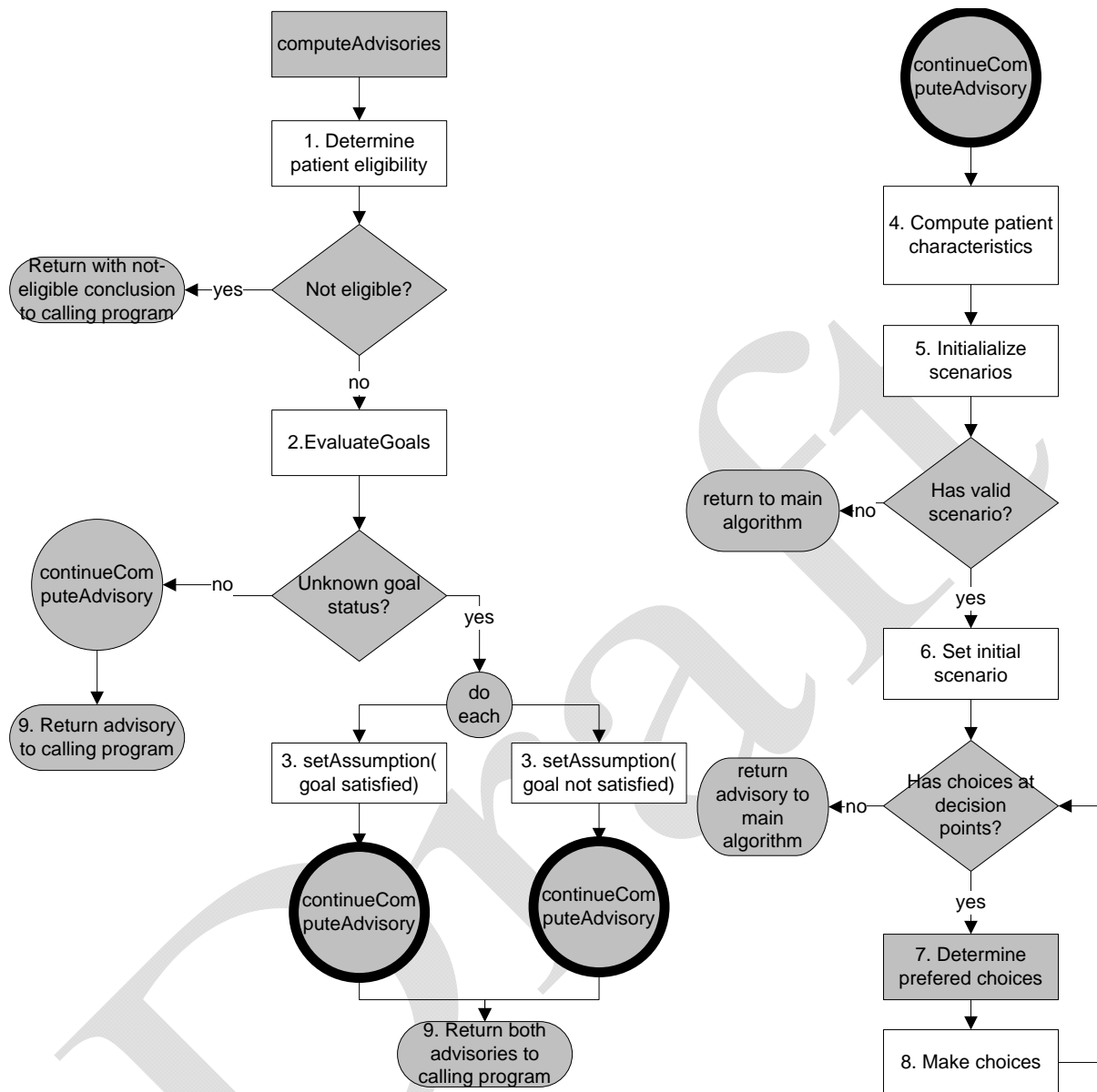


Figure 78 - The computational algorithm used by the computeAdvisories operation

- [1] Coleman WR, Martins SB, Tu SW, Goldstein MK, *Athena dss guide*. 2006, VA Palo Alto Healthcare System: Palo Alto.
- [2] Musen MA, Scalable software architectures for decision support. *Methods of Information in Medicine* 1999. 38: 229-238.
- [3] Gennari JH, Musen MA, Fergerson RW, Grosso WE, Crubezy M, Eriksson H, Noy N, Tu SW, The evolution of protégé: An environment for knowledge-based systems development. *Int J Hum Comput Stud* 2003. 58(1): 89-123.

- [4] Noy NF, Fergerson RW, Musen MA. The knowledge model of protege-2000: Combining interoperability and flexibility. 2th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2000). 2000; pp.
- [5] Genesereth MR. Knowledge interchange format. Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning. 1991; pp. 238-249.

Draft