

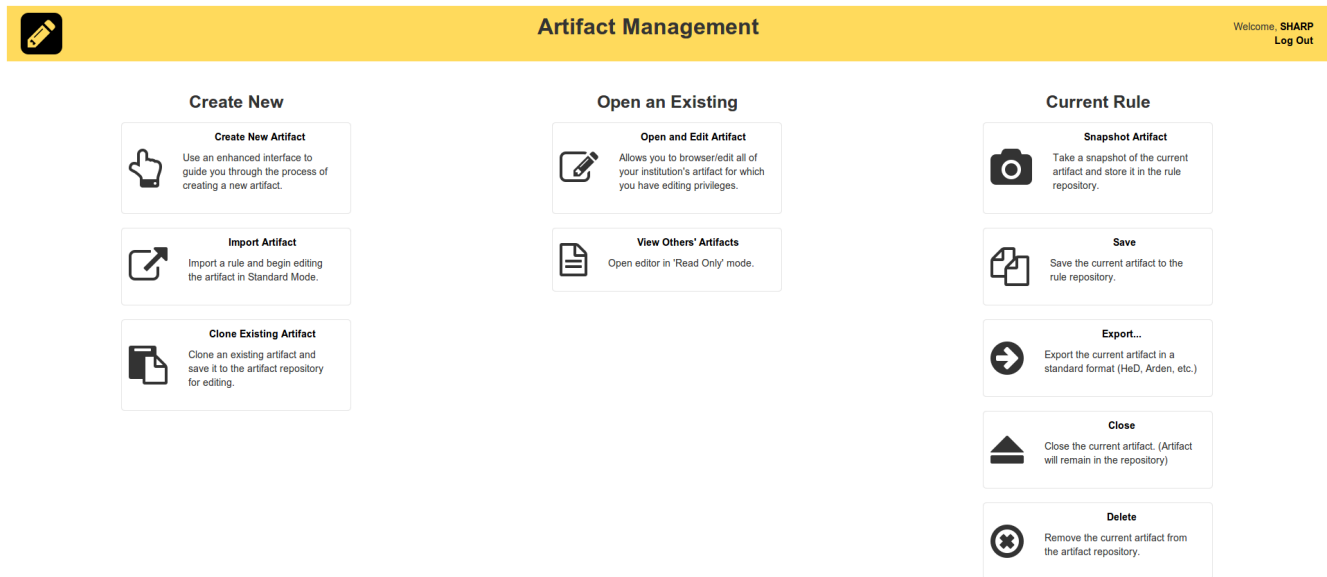
HeD Editor User Guide

The HeD editor is a web-based application that can be accessed from a common browser. It allows to create Clinical Decision Support (CDS) Knowledge Artifacts using a visual approach. The artifacts can be imported from Health eDecisions (HeD) XML documents, modified and eventually exported back to HeD or one of the other supported formats – currently OWL/RDF and HTML.

The editor's User Interface (UI) requires authentication. At the moment, the management of the credentials is delegated to the application server, so the editor can be accessed using any username/password pair that is accepted by the server. A dedicated authentication/authorization layer will be added at a later stage. Once the user has authenticated, the editor's proper interface becomes available.

The editor has a “master” view, which allows to control the artifact being edited at any given time, and several “slave” views which allow to author and edit the details of a KA. These views reflect the various components of an HeD document: metadata, triggers, conditions, actions and expressions. A final view is provided to summarize the artifact. This guide will present each of the views and describe the functionalities available from each.

Artifact Management



The main page focuses on the selection of the KA to edit. Artifacts can be loaded and exported into the HeD format, but internally are represented using the Web Ontology Language (OWL). Each artifact is a defined within an ontology that contains all and only the assertions about the artifact itself, its structure and its content. These ontologies import the HeD model ontologies and their dependencies (see the Developer's guide for more details). Artifacts are exposed using their title, but internally they are indexed using a versioned identifier, which allows to distinguish between different artifacts (with different IDs) and different versions of the same artifact (same ID, but different version). Notice that changing the title has no effect on the artifact's ID or version, since the title is not part of the ID. Currently, the editor does not support multiple documents, so only one artifact can be edited at a time.

In detail, the main page offers the following functionalities:


- **Create New Artifact:** a “blank” KA will be created. If an artifact is currently being edited, it will be closed without saving, unless the user cancels the request. The new artifact will have a random, unique ID and version 1.0.
- **Import Artifact :** loads and opens a KA from an HeD/XML document. A popup file selector allows to choose a file from the local file system and upload it to the editor. The import process generates an OWL model that describes the content of the original artifact. As with Create, any currently open artifact will be closed. The artifact's ID will be imported from the HeD document, if present, otherwise one will be assigned by the system as in the case of new artifacts.
- **Clone Existing Artifact :** creates a copy of the currently open artifact. The new artifact is independent from the original one, i.e. it has a different ID and a separate version history, but is otherwise equivalent in content.
- **Open Artifact :** loads and opens a KA from the internal repository. As usual, this operation may override any open artifact. Notice that the internal repository stores the KA using OWL/RDF rather than HeD/XML, so this operation will load OWL directly rather than importing it from

XML. The editor will show the list of artifacts available in the repository in a popup window. At the moment, the list of artifacts is flat: future releases will support the categorization of the artifacts and the ability to query (semantically) the repository for artifacts matching some user-defined criteria.








- View Artifact : opens a KA in read-only mode. This functionality is currently not implemented.
- Snapshot Artifact : creates a new version of the currently open artifact and persists it. Unlike “clone”, this artifact has the same ID as its source
- Save Artifact : persists a copy of the currently open artifact. ID and version are not changed during the process
- Export : allows to download a copy of the current artifact, serialized in one of the supported formats (HeD, OWL/RDF, HTML). As more formats become available, it will be possible to perform the conversion from here.
- Close : closes the current artifact. No authoring operation will be possible until another artifact is created/opened/imported.
- Delete : closes the current artifact and removes it from the repository. Once confirmed, the operation can not be undone.

When a KA is opened/loaded/imported, the editor will move automatically to the authoring section. It is always possible to come back to the main page by clicking on the “pencil” icon on the top left of the screen.

Metadata

**Hemoglobin A1c reminder for patients with poorly-controlled diabetes : Metadata**Welcome, SHARP
Log Out



MetadataSelect TriggerDefine LogicChoose ActionReviewTechnical View

ContributorsCoveragePublishersRights & UsageRelated ResourcesSupporting EvidenceDocumentation



Metadata:


Artifact Name: Hemoglobin A1c reminder for patients withLanguage: EnglishArtifact Type: Rule

Describe this artifact in your own words. You should describe the consequences of the rule and the conditions necessary to cause those consequences.

SaveClear

Contributors:

Name:
Role:
Type: ☐ Person ☐ Company
 Add  Clear

Name	Role	Type	
[Aziz, Boxwala]	Author	Person	

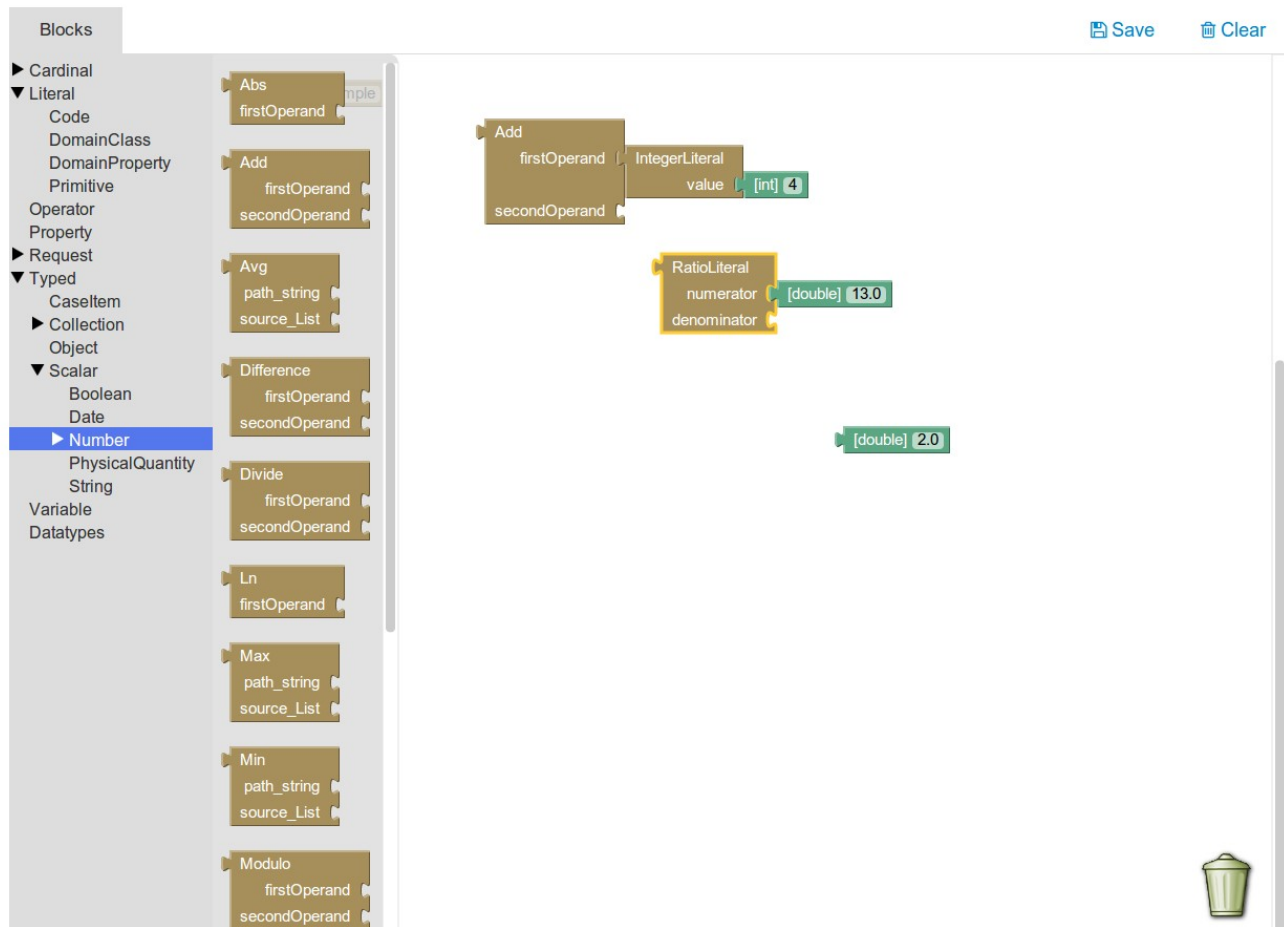
By default, the metadata section allows to define some basic information about an artifact (title, language, type and description). Additional sections can be shown/hidden on demand, clicking on the icons in the palette below the navigation bar. In Figure 2, for example, the contributor section has been expanded. The sections are modelled on the elements in the HeD schema. Their purpose is as follows. More details can be found in the HeD official specification.

- Contributors: information about individuals or organizations who authored/revised or generally participated in the creation of the artifact.
- Coverage: medical coded concepts describing the context where the artifact is supposed to be deployed.
- Publishers : individuals or organizations who participated in the distribution of the artifact
- Rights & Usage : information about the copyright on the artifact (if any) and its holders
- Related Resources: references to other artifacts or material somehow connected to the current artifact, with the notable exception of supporting evidence, which has a dedicated section
- Supporting Evidence : references to resources providing evidence to support the logic encoded in the artifact.
- Documentation: free form, rich text used to describe the artifact and provide additional information for users and consumers.

With the exception of documentation, the other sections follow the same pattern. A form is provided on the left part of the screen. The form's inputs are linked to the major elements in the corresponding HeD sections. Two local buttons (“Add” and “Clear”) are provided to create a new element or clear the one currently edited in the form. When “Add” is clicked, an element is added to the artifact and displayed on the right part of the screen. The display grid shows the main data elements: a full detailed view can be opened clicking on the “magnifying lens” icon. Currently the view is read-only: elements can not be modified, but have to be deleted and recreated. An element can be deleted clicking on the “trash bin” icon on the corresponding row, placed to the right of the “lens” icon.

Blockly Authoring

Triggers, Conditions, Actions and Expressions are assembled using the jigsaw puzzle-like, visual library “Blockly”.



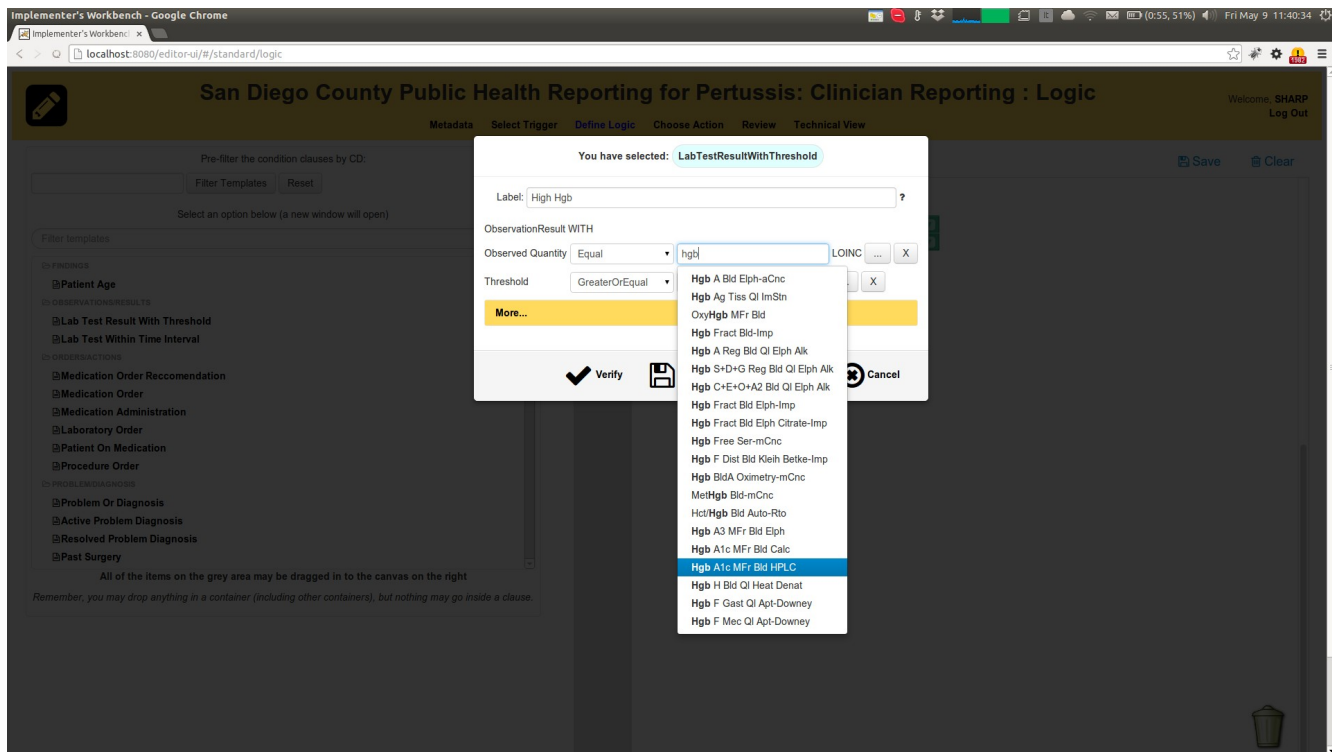
Blockly allows to compose expressions by assembling pieces on a canvas. A palette on the left allows to explore the available block types. The palette and available blocks are generally different for each section, but the functioning is the same. Any block can be dragged and dropped on the canvas. Blocks are then connected to each other to form expressions. The blocks' inputs and outputs are typed, so not all combinations are valid and thus possible. Unfortunately the current version of the library does not allow to differentiate the shape of the connector, so the difference is not immediately visible. The type information is available as a popup tooltip attached to each block.

Blocks can be deleted or collapsed through a context menu, accessible by right-clicking on a block.

Dragging a block onto the trash bin in the lower right corner of the canvas will delete the blocks, too.

Notice that an expression is NOT added to the current artifact until the “Save” link on the top right of the screen is clicked. “Clear”, instead, will reset the canvas to its initial state.

Template-based Authoring



The editor provides a simplified way to author expressions. The “Technical view” tabs supports free form expressions, but they require at least some technical skill which may not always be proper of a subject matter expert. On the other hand, many CDS artifacts are built following common patterns, so the editor provides the notion of “template” (aka “primitive” or “clause”) to facilitate authoring. Templates are parametric, pre-configured expressions which only require to fill some placeholder slots before they can be evaluated concretely. For example, “Patient on MEDICATION” is a template that requires to specify an actual medication, but otherwise can be mapped (in HeD) to a condition involving a ClinicalRequest for, say, SubstanceAdministrationOrders and some temporal/existential filtering operations.

In the current version, a template is defined by a DomainClass and one or more constraints on that class' properties. A constraint involves an expression and a value, which can be a literal (typically entered by the user) or the result of an existing named expression.

Templates are provided for triggers, conditions and actions alike: their editing process is the same in all cases, the only difference is given by the context where the template instances (concrete named expressions) will be used. Templates are placed in panels on the left part of the screen and can be searched and filtered. When the user has identified the desired template, a double click will open a popup window where the template parameters can be specified.

First of all, the user has to provide a name that will be assigned to the generated expression. That name will also be used to reference the expression, typically in variable expression blocks. The name is supposed to be a unique identifier, so names should not be repeated.

Second, for each field constrained by the template, the user will have to provide the required details.

The operations can be selected from a dropdown list. The editor performs some type inference checks to restrict the set of applicable operations.

Values are modelled internally using HeD literal expressions, which in turn involve one or more primitive datatype “elements”. For example, a Physical Quantity (PQ) requires a value (a decimal number) and a unit (a string). By default, the editor exposes one element only: in order to edit the literal in full, it is necessary to click on the “...” button on the right of each constraint. There, it will also be possible to switch from literal to expression values. Exposing only one element is not a limitation in general, since many primitives only have one element in the first place; moreover, templates allow to define default values for the individual elements (see the Templating Guide), so that the editor will pre-assign those values as the template is being instantiated.

Constraints on a field can further be *optional* or *multiple*. Optional constraints are not shown by default: they can be displayed clicking on the “More...” button, which will expand the window and show the additional constraints. Multiple constraints can be repeated: a multiple constraint will have a “+” button on the right which allows to duplicate the row and enter a second, alternative constraint on the same field.

Finally, templates may come with one or more integrity constraints which will allow to distinguish valid attempts to create an expression from invalid ones. For example, forgetting to specify the value of a non-optional field constraint will result in a validation error. Similarly, a template may require a numeric value to be, e.g., greater than 0: whenever a negative number is entered, an error message will be returned.

Validation can be performed at any time clicking on the “Verify” button in the popup window. Clicking on “Ok”, instead, will validate AND instantiate the template. As a consequence of the instantiation process, one or more named expressions will be generated, based on the semantics of the template. The expressions will then be available to build trigger, conditions, actions or other complex expressions.

CTS II Terminology Server Integration

The editor has the ability to connect to a CTS-II compliant server at runtime, in order to resolve medical terms and vocabularies. The functionality is integrated in a few different places (including the metadata section, where the clinical applicability of an artifact can be defined using coded concepts). The most important scenario is, arguably, the creation of an expression from a template. Many properties are of type “CD”, so they can be restricted to have a specific coded concept, or to belong to a value set. The editor supports both scenarios, as follows:

- Individual codes: when the user starts typing, the editor will make a background call to the CTS-II server, trying to resolve terms whose readable name (“designation”) matches what the user is typing. As the server returns, a list of suggestions will appear, allowing the user to pick one of the codes.
 - In general, the string will be resolved against any CD (“entity”) stored in the server. In many cases, however, the editor will have some context information that will limit the search to a particular code system. This is a typical scenario with templates, which usually restrict a CD property to hold values from a specific code system or value set
 - The editor supports the distinction between valuesets and code systems, so it will make the appropriate calls to the CTS-II server in a way that is transparent to the user. So, for example, a template may bind the admissible values of an attribute to the RXNORM code system: in this case, only concepts defined in RXNORM will be returned as suggestions. If the template, on the other hand, required that the code belongs to “anticoagulant drugs” (and a corresponding value set had been defined in the CTS-II server), the codes included in that set would be the candidates for the match.
- Entire Value Sets: sometimes, an entire value set may be used in conditions. For example, consider the difference between “substanceCode *equal to* [code for aspirin]” as opposed to “substanceCode *in* [antithrombotic drugs value set]”. The editor allows to look up and pick entire value sets as well as individual codes. In particular, when working with templates, choosing between “Equals” and “In” allows to make the distinction.

The image displays two side-by-side screenshots of a software interface for defining a template condition. Both screenshots show a 'PatientOnMedication' label and a 'SubstanceAdministrationEvent WITH' section. The left screenshot shows the 'Medication' field set to 'Equal' with a dropdown menu open showing 'Aspirin 325 MG [Ecp]'. The right screenshot shows the 'Medication' field set to 'InValueSet' with a dropdown menu open showing 'Antithrombotic drugs'. Both interfaces include a 'More...' button and a footer with 'Verify', 'Confirm and Save', and 'Cancel' options.

Triggers

The screenshot shows a web application titled "Hemoglobin A1c reminder for patients with poorly-controlled diabetes : Triggers". The interface includes a top navigation bar with tabs: Metadata, Select Trigger (active), Define Logic, Choose Action, Review, and Technical View. A sidebar on the left lists event types under three categories: ACT (User Logs In, User Logs Out, User Accesses Patient Record), TIMED (Specific Time Event, Recurring Time Event), and UPDATE (Patient In Pre-admission Status, Patient Admitted, Patient In Pre-discharge Status, Patient Discharged, Outpatient Visit, Lab Test Result Available, Procedure Result Available). The main workspace displays a hierarchical tree of trigger components: Triggers (red) contains Temporal (orange) and Typed (brown). The Temporal block is expanded, showing PeriodLiteral (isFlexible: true, count: 1), alignment, frequency, period, PhysicalQuantityLiteral (value_double: 1.0, unit: day), and phase. The Typed block is expanded, showing TimestampIntervalLiteral (low_dateTime: 5-15-2014, lowClosed, high_dateTime, highClosed). The Typed block is currently set to (Patient). The interface also includes a "Save" button, a "Clear" button, and a trash icon at the bottom right.

Rule triggers can be of two different types. “Typed” triggers require an expression that defines and selects the domain-specific elements, such as (new) Lab Tests or Encounters. “Timed” events are defined in terms of HeD PeriodLiterals: a period may establish an interval of validity, a temporal offset within the interval, a frequency and a maximum number of repetitions. Timed events are used to schedule evaluations of the artifact, while typed events are used to force the evaluation as a reaction to an external occurrence. Multiple triggers can be specified for an artifact: they will be considered alternatives. Complex events such as sequences or conjunctions of other events are not supported explicitly, but can be defined as custom named expressions and added as typed events.

Conditions

The screenshot shows a web-based logic editor titled "Antithrombotic Rule Full : Logic". The interface includes a top navigation bar with tabs for "Metadata", "Select Trigger", "Define Logic" (active), "Choose Action", "Review", and "Technical View". On the right, a user is logged in as "SHARP". The main workspace is divided into three sections: a left sidebar for "Filter templates", a central "Blocks" area, and a right canvas for building logic rules. The "Filter templates" sidebar lists categories like FINDINGS, OBSERVATIONS/RESULTS, ORDERS/ACTIONS, and PROBLEM/DIAGNOSIS, with specific items like "Patient Age" and "Lab Test Result With Threshold". The "Blocks" area contains a "Conditions" block with a dropdown menu showing "All", "One+", "Not", and "Age gt 18". The canvas shows a logic rule being constructed: "All must be true" followed by "Not Low Platelet", "One+ must be true" followed by "CABG", "IVD", and "AML", and "Not On AntiTh Rx". A trash icon is visible in the bottom right corner of the canvas.

The editor allows to define conditions (or “applicable scenarios”) in a restricted, but simplified way.

- Boolean (true-or-false) variable expressions can be used as simple conditions.
- Object variable expressions (e.g. the result of a clinical request) can be used as conditions in combination with the “Exists” block, which maps to the IsNotEmpty HeD expressions
- The “All”, “One”, “At least one”, “None” connectives can be used to aggregate nested conditions.
- The “Not” operator can be used to negate a conditions

If more sophisticated conditions are desired, they have to be created as named expressions and then referenced using a variable condition.

Actions

The screenshot shows a web-based editor for defining actions. The title bar reads "Antithrombotic Rule Full: Actions". Below the title bar is a navigation menu with tabs: "Metadata", "Select Trigger", "Define Logic", "Choose Action" (selected), "Review", and "Technical View". On the right side of the title bar, it says "Welcome, SHARP" and "Log Out".

The main workspace is divided into three sections:

- Left Panel:** A search bar at the top. Below it is a list of categories and their associated actions:
 - COMMUNICATIONS
 - Deliverable Message
 - DOCUMENTATION
 - Record Justification
 - ORDERS/ACTIONS
 - Medication Order Recommendation
 - Medication Order
 - Medication Administration
 - Laboratory Order
 - Procedure Order
- Bottom Left Panel:** A section titled "Conditional actions can be further specified:" with a "Filter templates" search bar. Below it is a list of categories and their associated actions:
 - FINDINGS
 - PatientAge
 - OBSERVATIONS/RESULTS
 - LabTestResultWithThreshold
 - LabTestWithinTimeInterval
 - ORDERS/ACTIONS
 - MedicationOrderRecommendation
- Right Panel:** A "Blocks" area showing a visual representation of the action logic. It consists of a vertical stack of blocks:
 - Action:** Perform all
 - Title: AntiThrombotic for IVD
 - Condition:
 - Create:**
 - Title: Create Pt Msg
 - Condition:
 - Action Sentence: PTMsg
 - Create:**
 - Title: Create Alert
 - Condition:
 - Action Sentence: Alert
 - Perform:** exactly one
 - Title: Treatment and Doc options
 - Condition:
 - Collect:**
 - Title: Record Exception
 - Condition:
 - Action Sentence: Record Exception
 - Create:**
 - Title: Order AntiTh
 - Condition:
 - Action Sentence: AntiTh
 - Create:**
 - Title: Propose AntiTh
 - Condition: Not Low Platelet
 - Action Sentence: Propose AntiTh


The Editor supports HeD actions in a way similar to Triggers and Conditions. It partially constrains the way actions can be specified, but at the same time it tries to provide a simplified way. Actions can be specified in two ways:

- Action Groups, which contain other Actions
- Atomic Actions

Action Groups allow to define an execution behavior (“all”, “one”, “none”, ...), as per HeD specification. Atomic actions have a type (“Create”, “Update”, “Remove”, ...) and a sentence, which is required to be a variable expression.

All actions have a descriptive title and, optionally, a local condition. Local conditions can be defined similarly to global conditions (see the dedicated section), using boolean variable expressions and logical connectives.

Overview / Preview



Antithrombotic Rule Full : Review


Welcome, **SHARP**
Log Out

Metadata Select Trigger Define Logic Choose Action **Review** Technical View

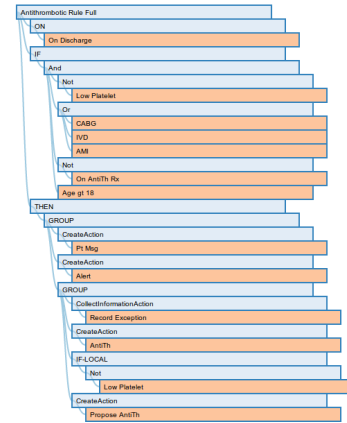
Use the navigation bar above to make any needed changes, then save/submit the rule using the buttons below

Artifact Name:

Status:

 Save

Preview output:



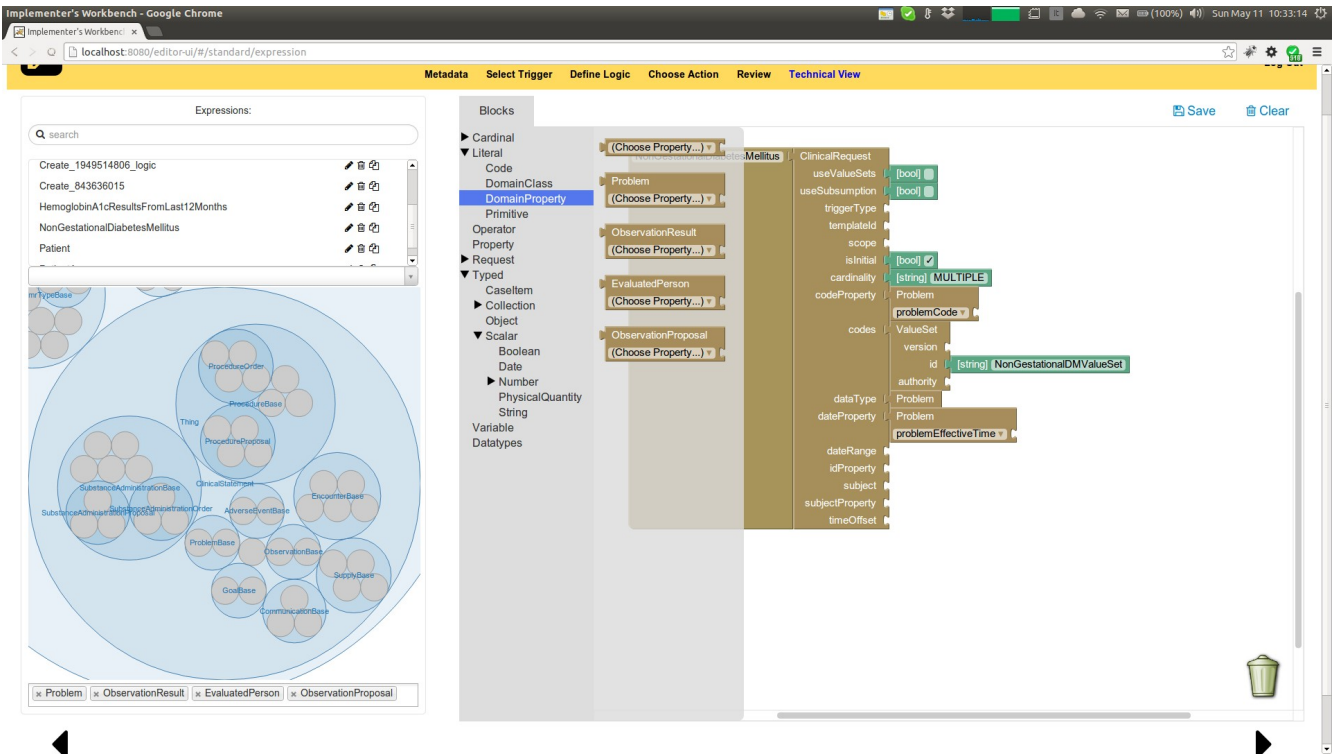
```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<knowledgeDocument xmlns="urn:hl7-org:knowledgeartifact:r1" xmlns:dt="urn:hl7-org:cdsdt:r2" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <metadata>
    <identifiers>
      <identifier root="asu.bmi.edu_427878269" version="1.0"/>
    </identifiers>
    <dataModels/>
    <title value="Antithrombotic Rule Full"/>
    <description value="This diagram represents a clinical rule based on NQF 0068 - PQRS 204: Ischemic Vascular Disease (IVD): Use of Aspirin or Another Antithrombotic"/>
    <relatedResources>
      <relatedResource>
        <relationship value="AdaptedFrom"/>
        <resource>
          <title value="NQF 0068"/>
          <location value="http://www.qualityforum.org/WorkArea/linkit.aspx?LinkIdentifier=id&itemId=66208"/>
          <description value="The original quality measure"/>
        </resource>
      </relatedResource>
    </relatedResources>
    <applicability>
      <coverage xsi:type="Coverage">
        <focus value="ClinicalFocus"/>
        <description value="Acute myocardial infarction"/>
        <value code="57054005" codeSystem="SNOMED">
          <dt:displayName value="Acute myocardial infarction"/>
        </value>
      </coverage>
    </applicability>
    <status value="Draft"/>
    <eventHistory/>
    <contributions>
      <contribution>
        <contributor xsi:type="Person">
          <contact>
            <name use="C">
              <dt:part type="GIV" value="Davide"/>
            </name>
          </contact>
        </contributor>
        <role value="Author"/>
      </contribution>
    </contributions>
  </metadata>
  <rule>
    <on>
      <onDischarge/>
    </on>
    <if>
      <and>
        <not>
          <lowPlatelet/>
        </not>
        <or>
          <cabg/>
          <ihd/>
          <ami/>
        </or>
        <not>
          <onAntiThRx/>
        </not>
        <ageGt18/>
      </and>
    </if>
    <then>
      <group>
        <createAction/>
        <inReg/>
        <createAction/>
        <alert/>
      </group>
      <group>
        <collectInformationAction/>
        <recordException/>
        <createAction/>
      </group>
      <ifLocal>
        <not>
          <lowPlatelet/>
        </not>
        <createAction/>
        <proposeAntiTh/>
      </ifLocal>
    </then>
  </rule>
</knowledgeDocument>
```

The review tab serves two main purposes.

- First, it provides a compact summary of an artifact's structure, following the “on.. if.. do..” paradigm and pointing to the named expressions involved.
- Second, it provides a preview of the resulting artifact in any of the supported output formats – currently HeD, OWL/RDF and HTML.

The user can also set the artifact's status from here and save the change.

Domain Specific Models



Expressions may involve domain classes and properties, such as Problems and Encounters and their respective attributes. The editor assists the users so that they don't have to remember and type the names manually. The expression tab (aka “Technical View”) provides a domain exploration tab which shows the taxonomy of domain classes in a tree-based display. The tree can be navigated, zooming in and out from the more generic to the more specific classes. When the user clicks on a class name, the class is added to the set of “active” classes actually used in the artifact. In alternative, a domain class can be searched by name using the combo-box just above the domain explorer.

The selection process is necessary since, in general, a domain may be large. Restricting the classes improves the efficiency of the editor in enforcing the validity of the expressions and helps the user focus on the necessary concepts.

Active classes will then appear in the Blockly palette, under Literal/DomainClass and Literal/DomainProperty. The latter blocks allow to choose attributes specific to a particular class, rather than the class itself.

Appendix - Expression operator taxonomy

The Blockly palette in the “Technical View” tab allows to pick HeD operators and compose named expressions. The operators are organized along different dimensions, as follows:

- Cardinal: Operations are grouped by number of operands: unary, binary, ternary, nary (variable number), nullary (zero operands). Misc operations have a fixed number of operands/attributes, but with specific names and semantics. So, for example, an “Add” is considered a binary operator since it requires two numeric expressions as operands, but a “DateAdd” is a Misc operator since it explicitly requires a start *date*, a date *granularity* and a *number of periods*.
- Literal: Literal expressions usually have no operands, but return values. They are further divided in CodeLiteral expressions – expressions that return CDs, Domain Class literals, Domain Property Literals (see the dedicated section) and Primitive Literals, which expose the HeD literal expressions.
- Operator: An alphabetically sorted list of the HeD operators, as per the original model.
- Property: Operators that allow to get or set properties of an object expression.
- Request: Operators that model the HeD concept of (Clinical)Request
- Typed: Operations are grouped by their return type, including object-returning, collection-returning and scalar-returning expressions. Dates, Strings, Numbers, etc.. are considered scalar values for this purpose
- Variable: References to other named expressions (must be already defined)
- Datatypes: primitive datatypes (int, string, bool, ...) used to specify values for literal expressions and operator attributes.