MDHT Modeling Style Guide for CDA

This style guide describes how UML is used to model CDA templates and implementation guides. This style is used within the OHT Model-Driven Health Tools (MDHT), but is applicable to any UML tools that are compliant with the OMG standard UML 2.2 or later. Example illustrations are shown using the MDHT table editor, but the modeling style could be applied to UML models created in other UML editing tools. We only address support within MDHT tools in this guide.

Contents

General Modeling Principles

The core principle is that models are created that align with the general UML 2 specifications and modeling semantics. Whenever possible, UML constructs are used to represent CDA template structure and semantics, and only as a last step are extended using UML profile stereotypes, which are also applied as recommended by the UML 2 specifications. This requires some reinterpretation of current CDA implementation guides that are written in structured English prose. The purpose of this guide is to describe how CDA template rules are mapped to UML.

The MDHT design is guided by three primary requirements that determine choices for the UML modeling style. When conflicting requirements occur, they are resolved in the priority shown here.

1. Generate Java libraries for validating CDA instances
2. Generate Java libraries used for CDA application development
3. Publish implementation guides

The modeling style is also influenced by choices that enable automated model-to-model transformation and code generation using the Eclipse Modeling Framework (EMF). However, all efforts are made to create models at the Platform Independent Model (PIM) level as defined by the Object Management Group (OMG) architecture.

Each modeling construct includes a list of CDA profile stereotypes that define additional metadata required for complete CDA template specification. These metadata are not captured by the base UML construct, so it is extended using stereotype properties. The stereotypes and properties are defined in a separate profile specification.

Model Structure

Project

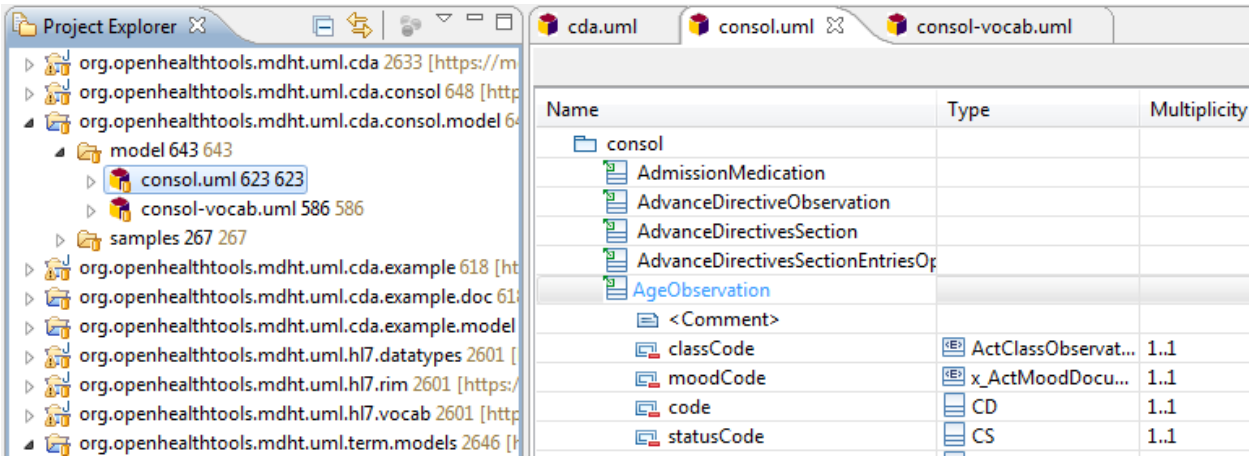
- All UML models are contained within Eclipse projects in your workspace. The details of project structure are outside of the scope of this UML style guide, but the general recommendation is that one Eclipse project contains one model package. This model package generates code for one Java package containing classes from the UML package. One or more related UML models containing code system or value set definitions may be contained in the same project, where those terminologies are used for constraints in this project's CDA templates.

Model or Package

- A "model" is represented using a UML Package and contains a collection of classes.
- A package may contain UML Package Import relationships that define which other packages are required for dependencies. For example, the IHE package imports the CCD package, which imports the CDA package. Package imports are optional at this time, but good practice for documenting dependencies.

- A package may contain UML Element Import relationships that include an individual class from another package, instead of importing the entire package contents. At this time, Element Import is only used to support publishing specifications that contain a subset of one or more other packages.
- CDA profile stereotypes on Package:
    - «CodegenSupport»

In the example MDHT screenshot shown below, the workspace contains model projects for cda, consolidated CDA (consol), an example model project, and others. Each project contains one primary model. Shown here, the consol.model project contains consol.uml and consol-vocab.uml that are open in the MDHT table editor.



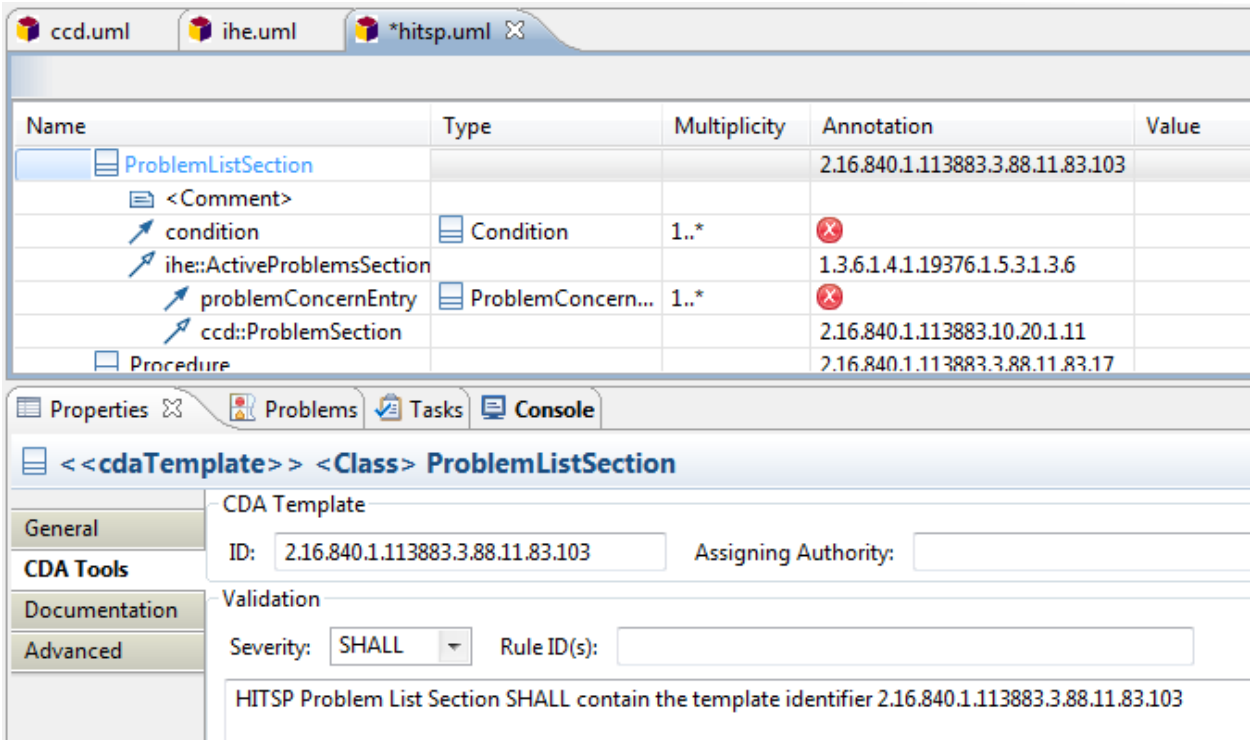
Template Classes

- Each CDA template is represented using one UML Class.
- Each CDA template has exactly one template ID assigned, which serves to identify use of this template in a CDA document instance.
- The template ID is defined in the class metadata, using a UML stereotype. All model relationships and constraints refer to the template class names, not their IDs. A template ID should **never** be used when specifying constraints or template relationships.
- A “conforms to” rule, also called “implies”, is represented using a UML Generalization, i.e. a template class has another template or CDA type as its superclass.
    - Direct conformance: immediate parent
    - Indirect conformance: parent of a parent
- Each CDA template class must conform to exactly one CDA base class, either as direct superclass or indirectly via a parent template. For example, a template conforms to CDA Section or Observation.
- All conformance rules from all superclasses are inherited into each template.
    - A template class may redefine (or, replace) conformance rules inherited from superclass templates. Any class attribute having the same name as an attribute from a parent template redefines that inherited rule.

    - TODO: clarify representation for redefining template associations other than simple attributes.
- A template class may have multiple immediate parent classes, often called “multiple inheritance”, such that the template conforms to two or more independent templates, plus their superclass templates. Although multiple inheritance is possible, it should be used with extreme caution and the details about when and how to use it are beyond the scope of this guide.
- CDA profile stereotypes on Class:
    - «CDATemplate»

In the example MDHT screenshot shown below, the HITSP C83 Problem List Section template conforms to the IHE Active Problems Section, which conforms to the CCD Problem Section. As shown, the template ID for this class is viewed and edited using a convenient properties tab, although it is saved in the UML standard format as a stereotype property. MDHT tools hide the UML internal structure from template designers.

These template conformance rules are modeled as UML generalization relationships, i.e. IHE ActiveProblemsSection is a superclass of HITSP ProblemListSection. These superclass relationships often refer to other models, from HITSP to IHE in this example. The base template is referred to using only its UML class, not its template ID (which is included within the target template class metadata). Also, ProblemListSection indirectly requires conformance with the CCD ProblemListSection, via the intermediate IHE class; there is no need for ProblemListSection to refer directly to CCD ProblemSection.



Categories of Template Classes

Most CDA templates fall into three categories, with a small number of examples in the fourth category. The published implementation guides use these four categories as primary sections of the guide.

- Document
  - Header constraints on subclasses CDA Clinical Document
- Section
  - Subtypes of CDA Section
- Clinical Statement
  - Subtypes include: Act, Observation, Encounter, Organizer, Supply, and others
- Other Classes
  - Classes within CDA header, e.g. Patient
  - Classes derived from ActRelationship, or Participation

CDA Header Constraints

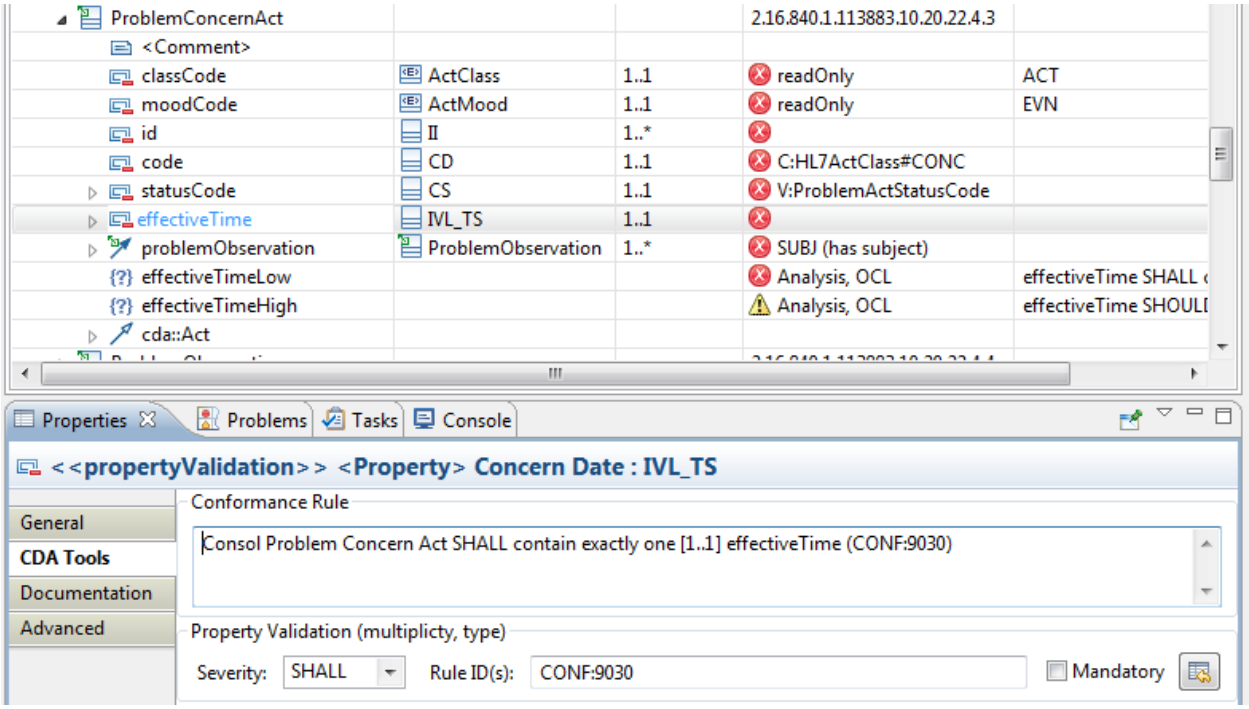
- Commonly called "header constraints," these are simply conformance rules on elements contained within a Clinical Document, other than children of `structuredBody` or `nonXMLBody`.
- In UML representation, header constraints on contained within a template class that is derived from Clinical Document, e.g. CCD Continuity Of Care Document class. Also, the Consolidated General Header Constraints class is a UML class that conforms to CDA Clinical Document. These general header constraints are often included via base class inheritance in other document types.
- Every Clinical Document template that includes header constraints must also have a template ID, such that document instances having that template ID conform to the included constraints.

Validation Characteristics

Each conformance rule has the following characteristics:

- Validation severity (a.k.a. "conformance verb"): SHALL, SHOULD, or MAY
- One or more Rule ID(s)
  - Multiple rule IDs are present only when a single modeled constraint implements more than one rule from the source text of a specification. These are used infrequently.
  - Enter multiple IDs in the MDHT editor by separating IDs with a comma.
- In MDHT properties view, the derived Conformance Rule message is displayed as read-only text
- CDA profile stereotypes:
  - «Validation»

In the example MDHT screenshot shown below, the `effectiveTime` attribute has conformance severity of SHALL and a rule ID of CONF:9030. The automatically derived conformance rule text message is shown in the properties view, using the current HL7 best practices for structured English formatting.



Modeling Conformance Rules in UML

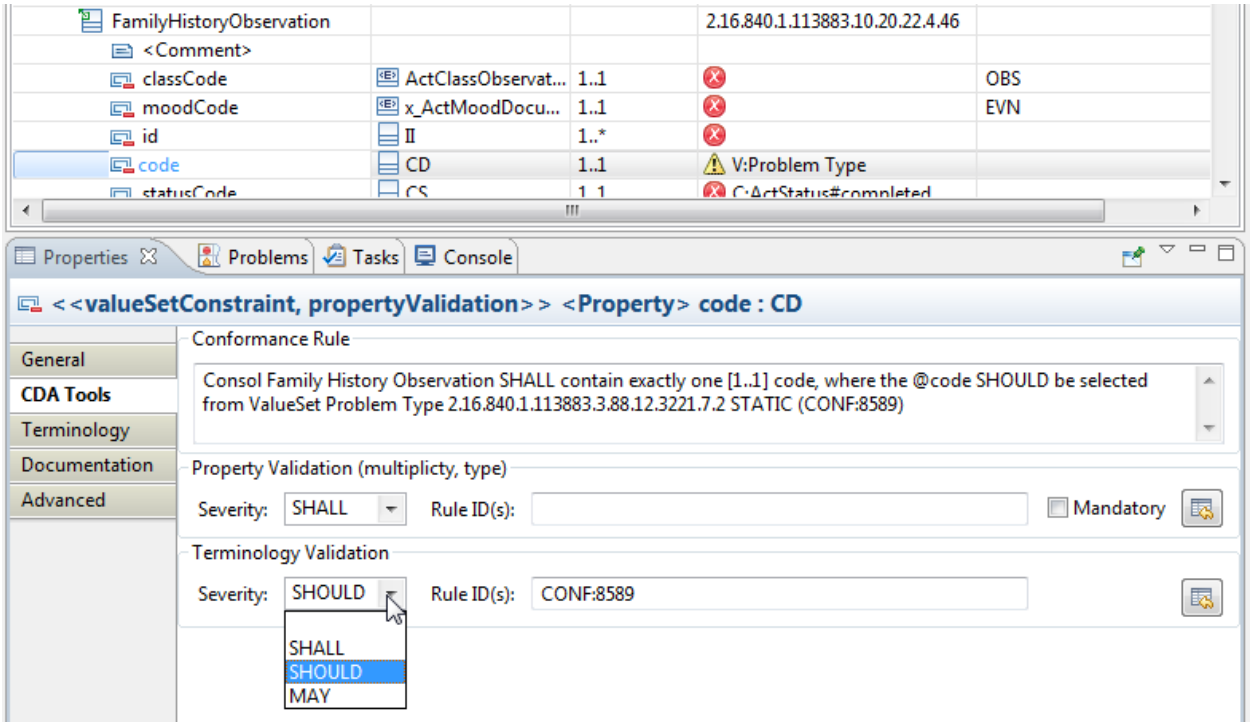
Attributes

In UML 2, an "attribute" is any property of a class whose type may be a primitive data type (such as String), or reference to another complex class. HL7 models use the term "attribute" to mean those class properties that have an HL7 data type as their type, such as ST, II, CS, CE, etc. When we refer to attribute constraints in this guide, we are referring to this more limited interpretation of HL7 attributes inherited from the base CDA (and ultimately RIM) classes.

- Attributes are restricted in CDA templates in the following ways:
  - Multiplicity (restricted from 0..* to 1..*, or from 0..1 to 1..1)
  - Data type (e.g., from ANY to CE)
  - Terminology constraints for coded attribute values
    - Code System (usually for a fixed code value)
    - Value Set
    - Concept Domain (for future use, not used in current CDA IGs)
- Validation characteristics are specified as described in a previous section
- Included for documentation
  - When a template class attribute or association does not add additional conformance rule constraints, it may be included in a template specification to add documentation about usage.
  - Documentation (as UML Comment) may be added to any attribute, with or without additional validation constraints

- CDA profile stereotypes on UML Property:
  - «PropertyValidation»
  - «CodeSystemConstraint»
  - «ValueSetConstraint»
  - «TextValue»

In the example MDHT screenshot shown below, the code attribute is required (by setting multiplicity to 1..1) and a value set constraint is specified by selecting from previously modeled value sets in the consol-vocab.uml model. The constraint severity is set to SHALL. The automatically derived structured English expression is shown in the properties view.



This example also shows a conformance rule that requires two separate validation settings on one attribute. In this conformance rule, the attribute SHALL be present, and if it is present in an instance, the terminology constraint SHOULD be enforced (i.e. a Warning when validating instances where the terminology rule is not satisfied). In general, the first Property Validation severity should be set only when the severity is different from the Terminology Validation, or when a terminology constraint is not present or applicable. The terminology constraint is specified separately using the Terminology tab. This option to specify Terminology Validation severity and rule ID is displayed in the CDA Tools tab only after a terminology constraint has been added.

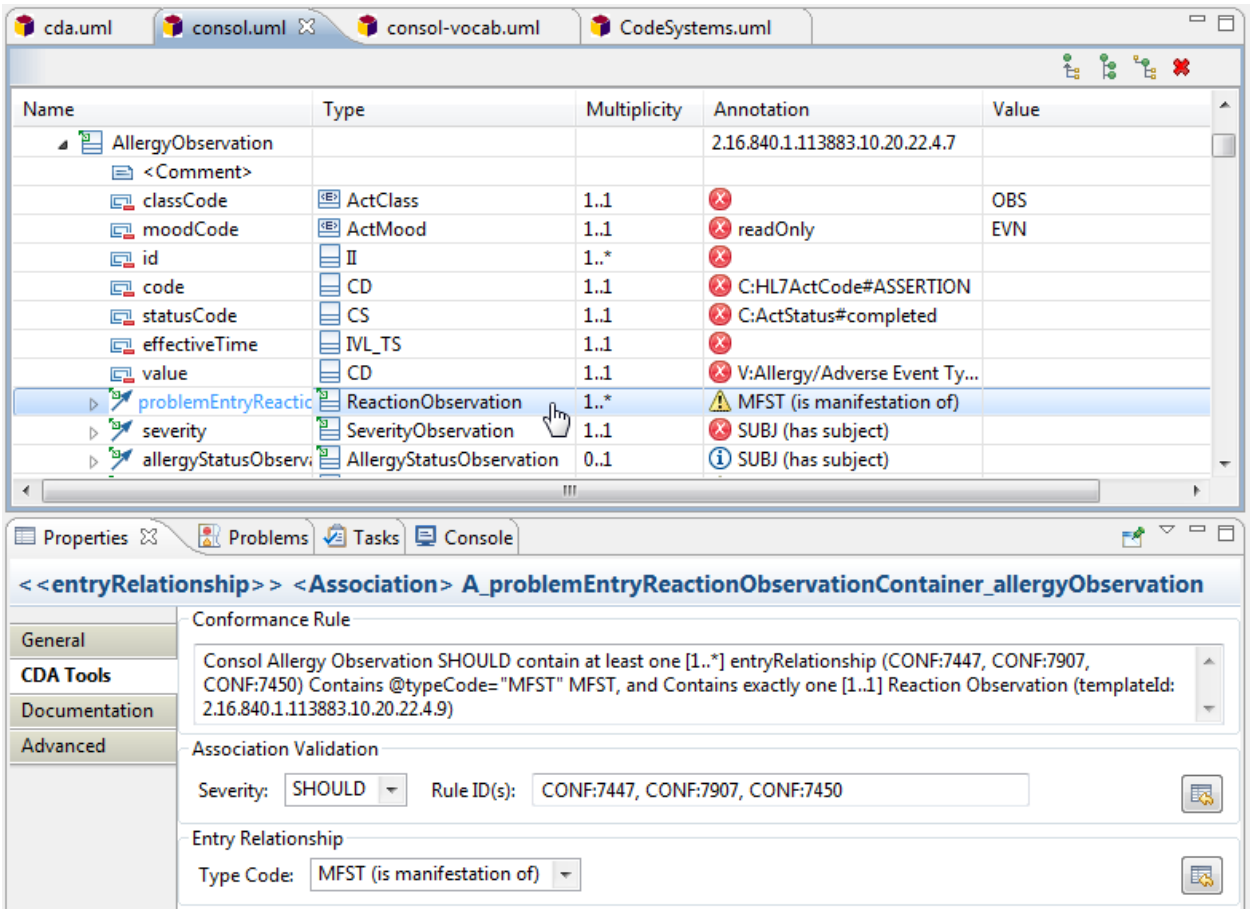
Associations to Included Templates

Associations are used in UML to define relationships between two classes, the navigable direction of that association, and optional metadata about the association.

- Associations between templates omit the intermediary CDA ActRelationship or Participation (they are re-inserted when Java code is generated for validation or development). A `typeCode` for the intermediary class may be assigned via a stereotype on the association.
- The following CDA association types are currently supported in transformation to Java and validation execution (other associations may be modeled, but will not be transformed into runtime code):
    - ClinicalDocument -> Section
    - Section -> Section
    - Section -> Entry
    - Entry -> Entry
- CDA profile stereotypes on UML Association:
    - «ActRelationship»
    - «Participation»

In the example MDHT screenshot shown below, the Allergy Observation clinical statement entry includes three associations to other entry templates, with severities set to SHOULD, SHALL, and MAY (indicated by the icons in the Annotation column of the table editor). The association to `ReactionObservation` has its `typeCode` fixed to MFST, its multiplicity set to 1..*, and association validation severity set to SHOULD. Thus, this template SHOULD contain one or more Reaction Observation templates.

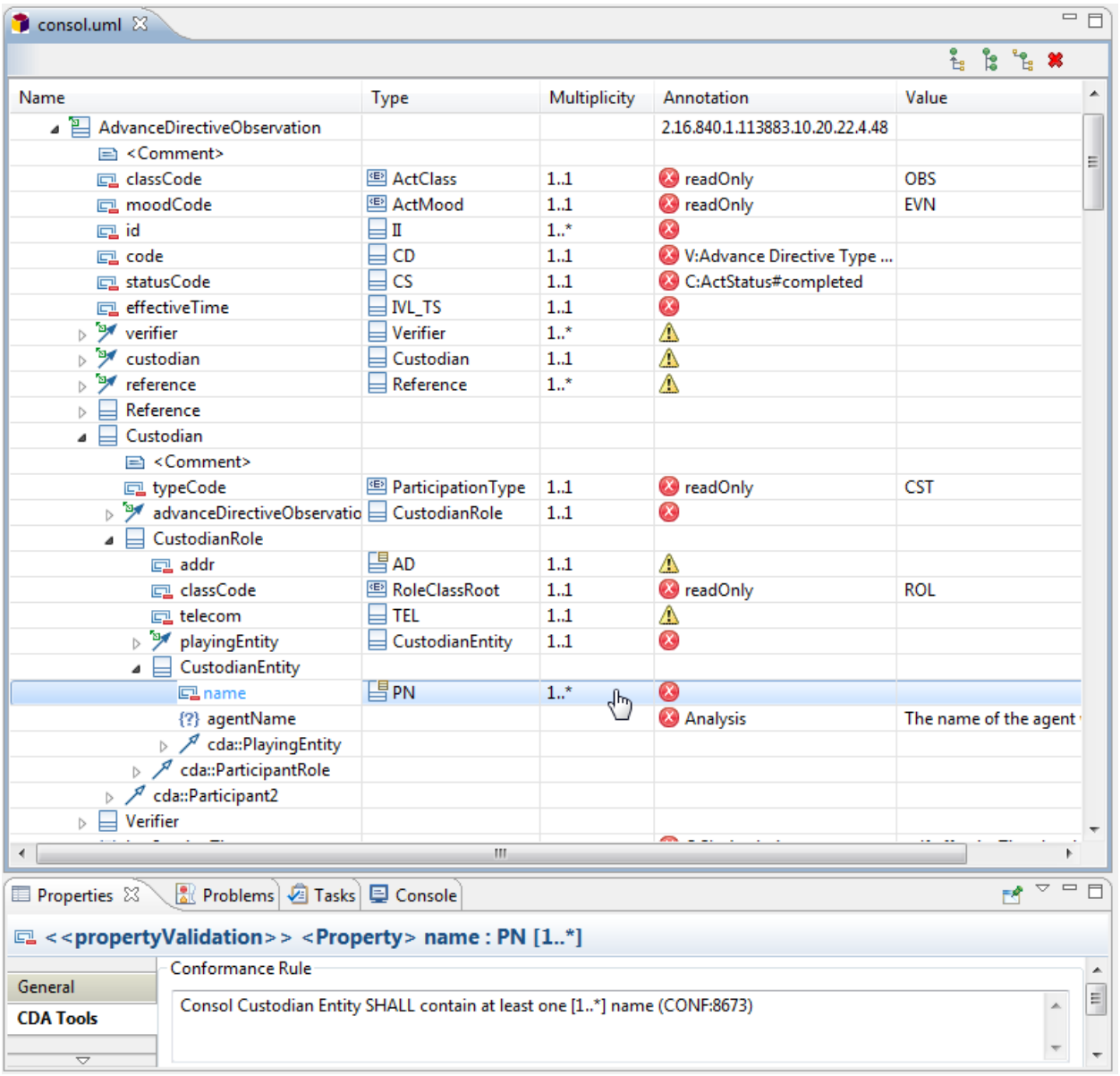
In UML, associations are not contained within a class, but navigable associations are shown this way in the MDHT table editor for simplicity and a more intuitive view. The Allergy Observation class actually contains a property whose type is the association's target class, and a separate Association object is owned by the same package containing this template class.



Associations to Untemplated Nested Classes

CDA conformance rules are sometimes specified on nested CDA elements within a template class, without defining the nested CDA class as a separately reusable template. We call these "untemplated classes". This style of conformance rule is supported in both the table editor and in transformation to validation of CDA instance documents. In many cases, this feature eliminates the need to write lengthy custom OCL expressions. The most common use for untemplated classes are for conformance rule contraints on child elements that are *not* CDA Clinical Statements. The example below illustrates constraints on a Participant, along with its role and entity elements.

The table editor for a template with nested classes appears as shown here:



Nested within the CDA Observation class are child elements for Custodian, CustodianRole, and CustodianEntity (and also Verifier and Reference, not expanded here). This template for Advanced Directive Observation includes constraints on these child CDA elements, without creating separate template definitions for the other classes. This screenshot shows that the CustodianEntity `name` is set to be required (multiplicity set to 1..*) with a SHALL validation severity.

These nested untemplated classes are translated into OCL for execution in Java-based validation, but those details are not required in this modeling style guide.

The modeling workflow process for nested classes could proceed as follows:

1. Create an association from AdvancDirectiveObservation to Participant2.
2. Change the association end name to "custodian".

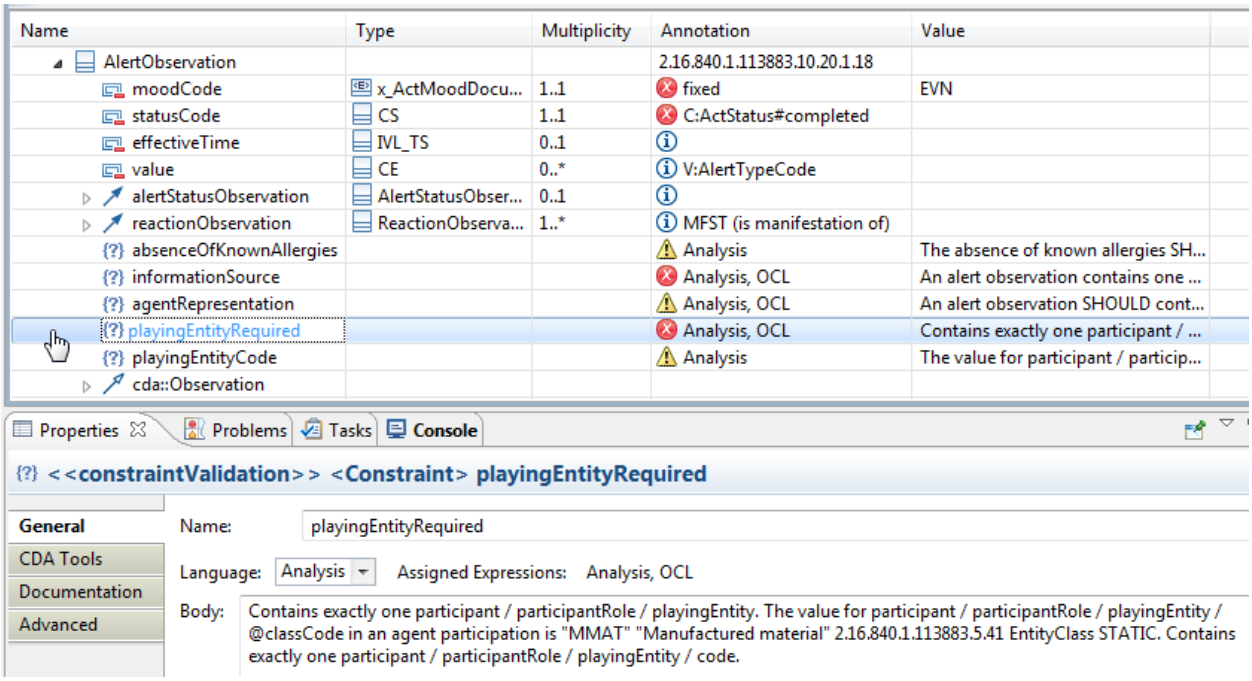
3. Right-click on the custodian association and select Add UML > Nested Class
4. Enter Custodian for nested class name
5. Select typeCode and participantRole as attributes to constrain
6. Add constraints on added attributes, as applicable
7. Repeat this process recursively for associations within the new nested class. In this example, create another nested class for the new participantRole association.

Other Constraints

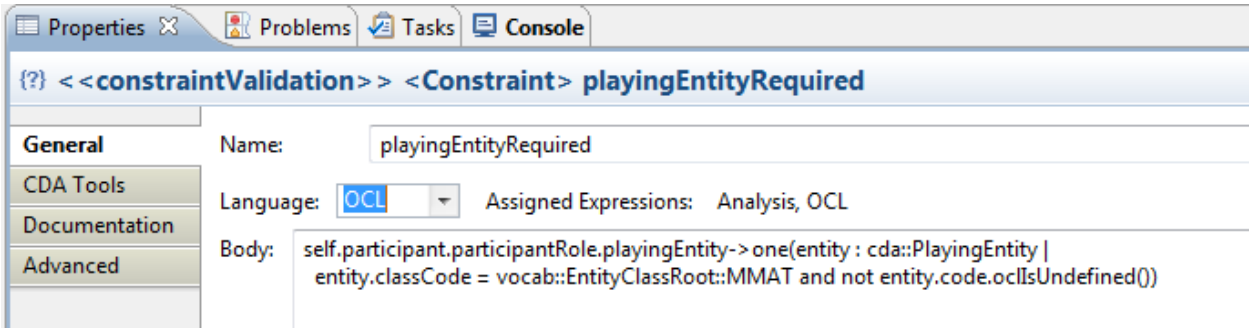
Whenever possible, template constraints are modeled using native UML structural definitions such as attributes with multiplicity restrictions or associations to required child templates. However, some constraints have a more complex expression that cannot be represented with simple structure. UML includes a Constraint metaclass for this purpose.

- A UML Constraint may be added to any UML model element to specify additional constraints that are not represented using other structural definitions. In most cases, a UML Constraint is added to a class, and that class becomes the root context for the constraint expression.
- A Constraint has the following properties:
    - Name
    - Language (one or more of : Analysis, OCL, or XPath)
    - Body (must include a valid expression for each specified language)
- Although not required, every Constraint should include an Analysis language expression as a structured English version of this constraint.
- A computable expression (in OCL or XPath) may not be possible, in which case only an Analysis language constraint is present. OCL language is preferred at this time and is the only language supported by MDHT automated transformation to runtime validation. Future support may be added for executing XPath expressions.
- The text of the analysis expression should be written independent of constraint severity. By doing so, a constraint's severity may be changed (e.g. from SHOULD to SHALL) without needing to rewrite the analysis language expression.
- OCL
    - templateId is never used in an OCL expression, only use model class names
    - Must use full qualified name for class references, e.g. ccd::ProblemObservation, or cda::Author, or datatypes::ST

In the example MDHT screenshot shown below, Alert Observation includes five Constraints. All have an Analysis language and three have an OCL language expression. A summary of what languages have been added is easily viewed in the Annotations column of the table editor. The detail of each constraint is contained in the Properties view, where the General tab contains the languages and expressions, and the CDA Tools tab contains extra stereotype extensions for validation severity and rule ID.



This constraint for `playingEntityRequired` is also defined in the OCL language and used for execution in Java runtime and validation.



Terminology

The subject of healthcare "terminology" is a vast topic. This document introduces a small subset of concepts necessary for understanding how to create CDA template models. For more information see references section, Core Principles and Properties of HL7 Version 3 Models.

- Code System: An ontology or similar system of concept definitions and codes, e.g. SNOMED CT, or LOINC.
- Value Set: A set of codes drawn from one or more code systems, either as a specific enumerated list or definition of a query on the code system ontology that will return a specific set of codes.
- Concept Domain: A named category of like concepts (a semantic type) that is specified for an attribute in a information model, whose data types are coded or potentially coded. Concept Domains exist to constrain the intent of the coded element while deferring the binding of the

element to a specific set of codes until later in the specification development process. Thus, Concept Domains are independent of any specific vocabulary or Code System.
    - **NOTE**: Concept Domains are not currently used in MDHT implementation, but support is planned for future releases.
- Concept Domain Binding: The association of a value set with a concept domain in a given context.

The UML models contain a minimal definition of terminology metadata for code systems and value sets. This is not intended to be a complete terminology representation, but is used to enable reference to code systems and value sets that are often defined in other tools or terminology services. These metadata also support definition of attribute constraints that *bind* to these terminologies and enable runtime validation of CDA document instances.

These terminology representations in UML are based on three sources:

- Information model included in the HL7 CTS-2 specification (see References)
- HITSP C80 specification metadata for code systems and value sets
- Terminology binding representation in the HL7 Model Interchange Format (MIF)

Terminology Definition

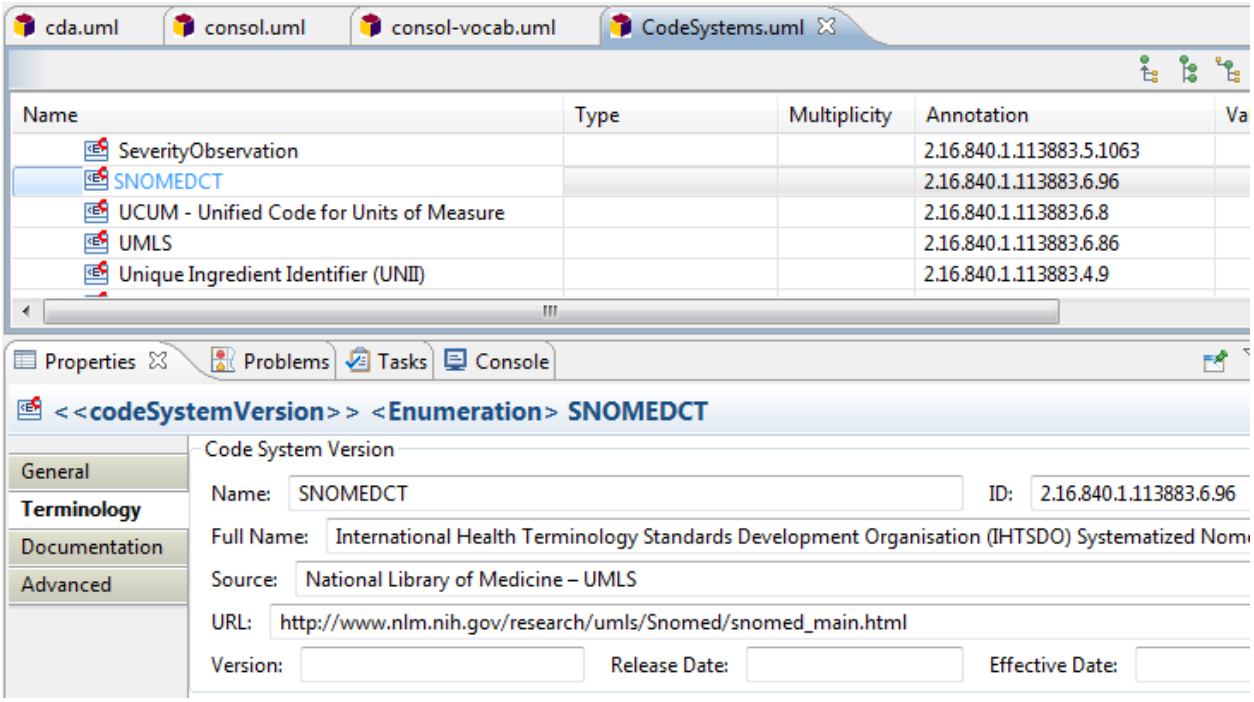
UML models for CDA implementation guides use two kinds of terminology definitions:

- Code System Version, modeled as UML Enumeration
- Value Set Version, modeled as UML Enumeration

Code System Version

A Code System Version does not contain a list of values in that code system, but only metadata about the code system, primarily its name and OID. It is recommended that all Code System Version definitions be included in a single shared model. MDHT tooling includes `CodeSystems.uml` within the `org.openhealthtools.mdht.uml.term.models` project.

In the example MDHT screenshot shown below, a Code System Version is defined for SNOMED CT and metadata are included that describe this code system. The metadata for a selected Code System Version may be browsed or edited in the Properties view. The most important properties of a code system definition are its **Name** and **ID**, which are used to validate CDA instances and publish implementation guides.

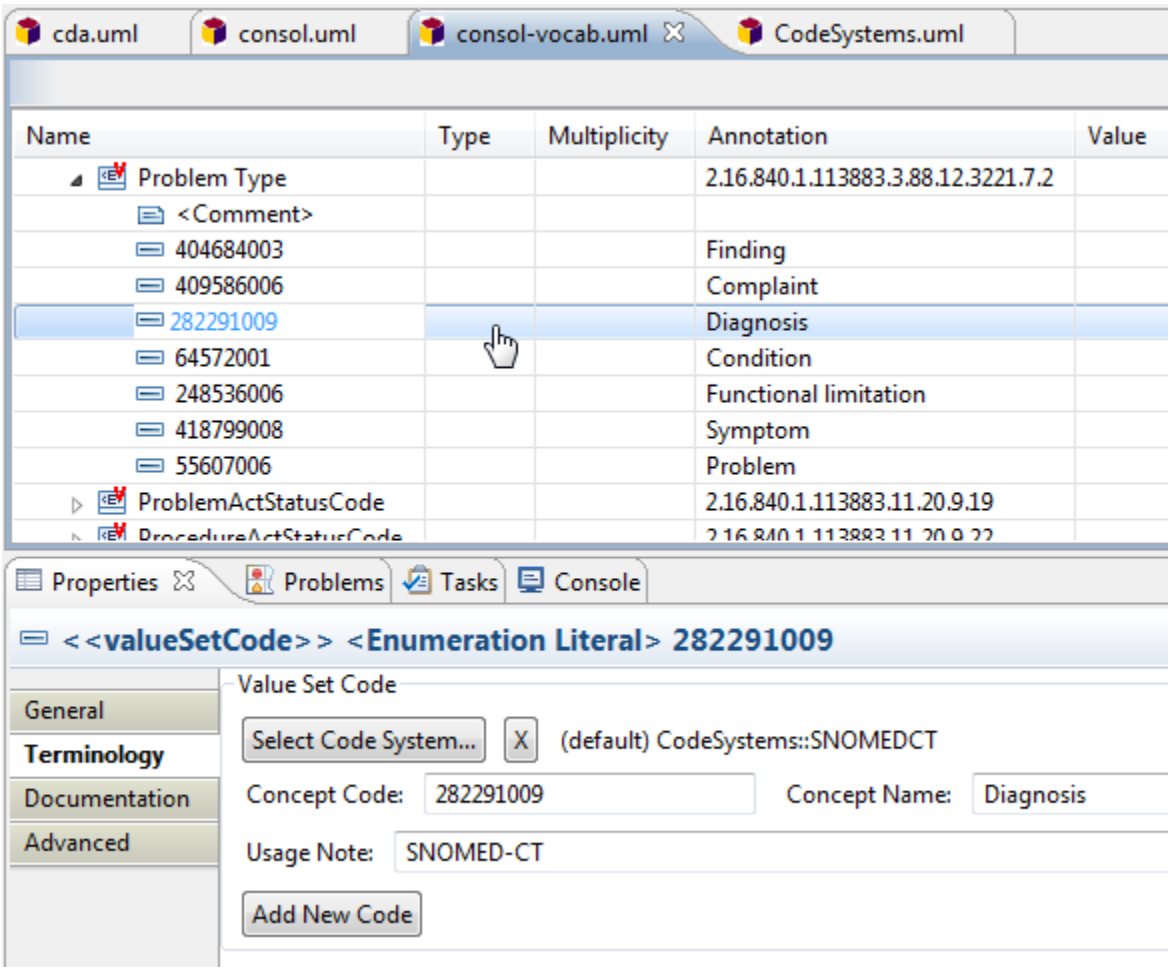


Value Set Version

A Value Set Version has either an "extensional" definition with an enumerated list of values or an "intensional" definition that does not contain an enumerated list. It is recommended that small extensional value sets (e.g. less than 20 values) be fully enumerated in their UML definition. For extensional value sets that do contain an enumerated list in UML, the generated Java runtime implementation will validate that CDA instances contain a code value from this list. A forthcoming MDHT enhancement will enable intensional and large extensional value sets to be used for validation via a CTS2 query to a terminology server.

Value Set Version definitions are usually contained in a model that is co-located in the same project as the domain model that uses those value sets. For example, consol-vocab.uml is located in the same project folder as consol.uml and it contains the value sets that are defined as part of the HL7 Consolidated CDA specification.

In the example MDHT screenshot shown below, a Value Set Version is defined for Problem Type and the metadata are included from the HL7 Consolidated CDA specification. The metadata for a selected Value Set Version may be browsed or edited in the Properties view. This value set has an "extensional" definition with an enumerated list of seven values. This value set is based on SNOMED CT, and the Value Set Definition in UML includes a reference to the corresponding Code System Definition in UML. It is possible for each code in a value set to be from a different code system, although this is rare in practice and all contained codes are assumed from the Value Set Definition's code system, unless overridden.

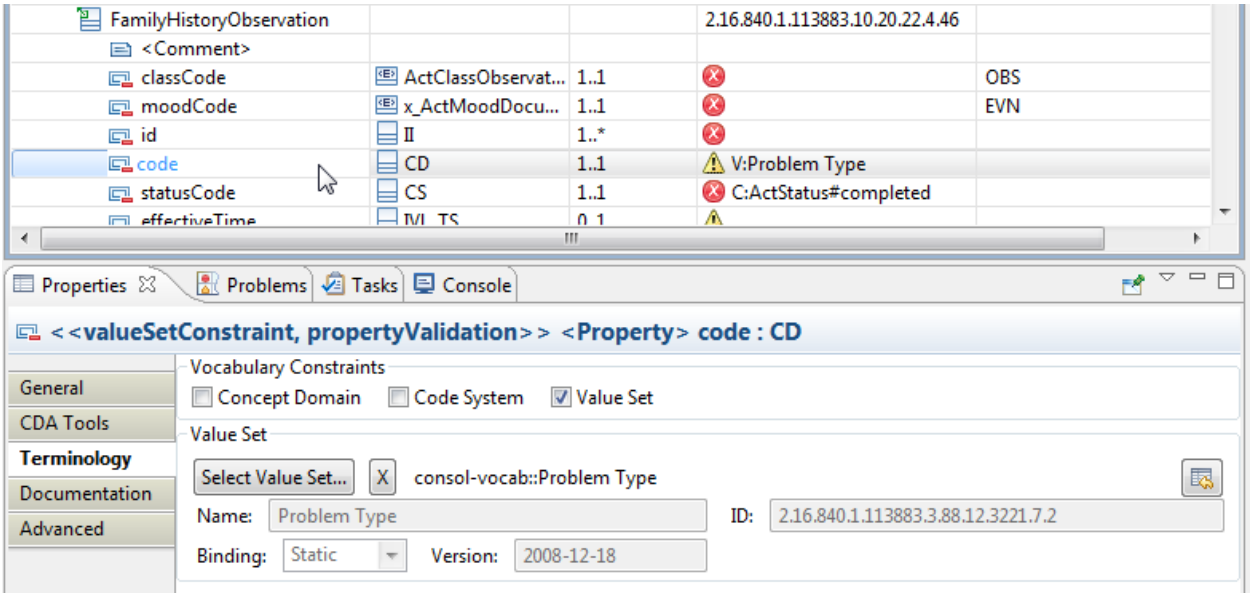


Terminology Bindings

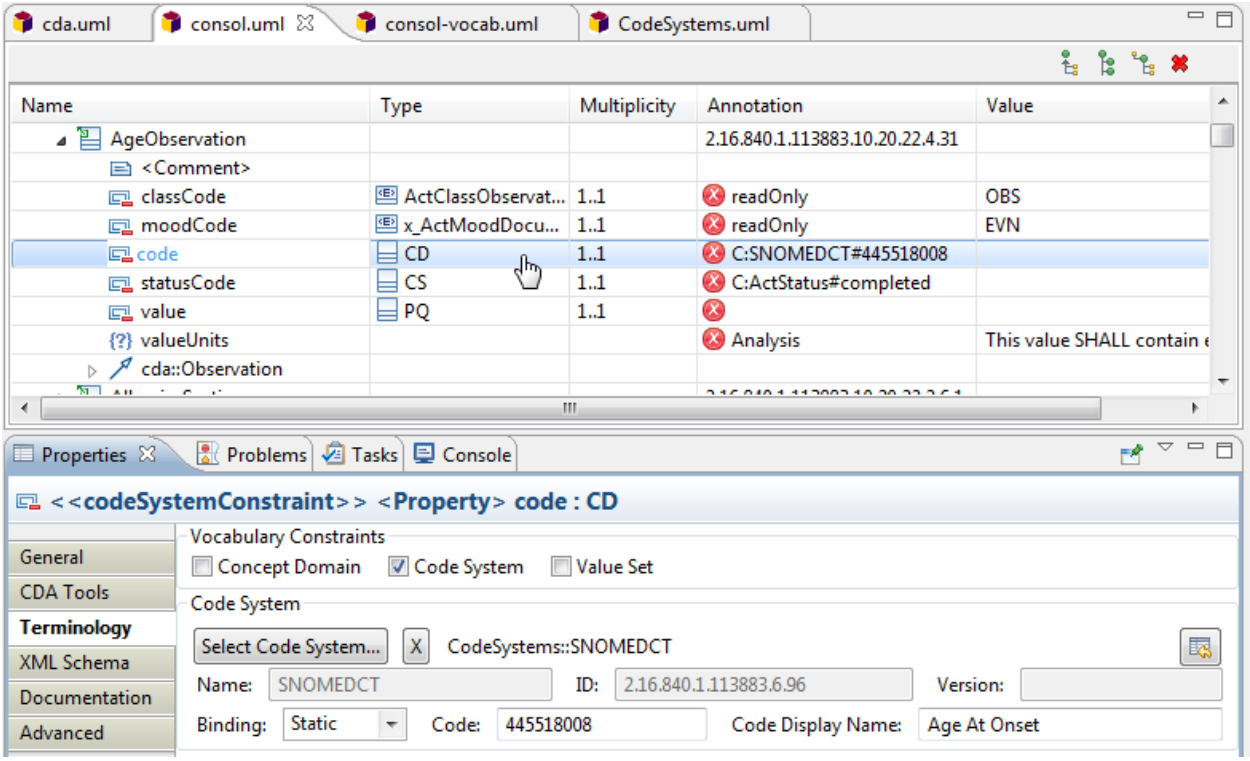
An information model, as represented by CDA templates, may include **bindings** to terminology that that specify the meaning or semantics of model elements, and/or constrain allowed value(s) of an attribute in a CDA document instance.

For CDA templates, any attribute with a coded value (it's type is derived from CD) may include a terminology binding. The MDHT editor will display a Terminology tab in the Properties view for coded attributes where you may add constraints. You should select one of the three binding types: Concept Domain, Code System, or Value Set. Select only ONE of these; in principle, more than one binding type could be specified but the current validation framework uses only one. Concept Domain is currently ignored for validation, but may be added for documentation.

In this example, the `code` attribute of `FamilyHistoryObservation` is constrained to require that instance values be from the `Problem Type` value set definition. The value set is selected from previously modeled (or imported) Value Set Version definitions (usually in a separate model in the same project). The metadata about that value set are displayed as read-only fields in the editor.



The second common constraint for CDA implementation guides is when an attribute is required to have a single code from a particular code system. In this example, the code SHALL be 445518008 from SNOMED CT, and the display name is "Age At Onset". The code system is selected from previously modeled Code System Version definitions (usually in a separate shared model). The metadata about that code system are displayed as read-only fields in the editor and the required values for code and display name are entered.

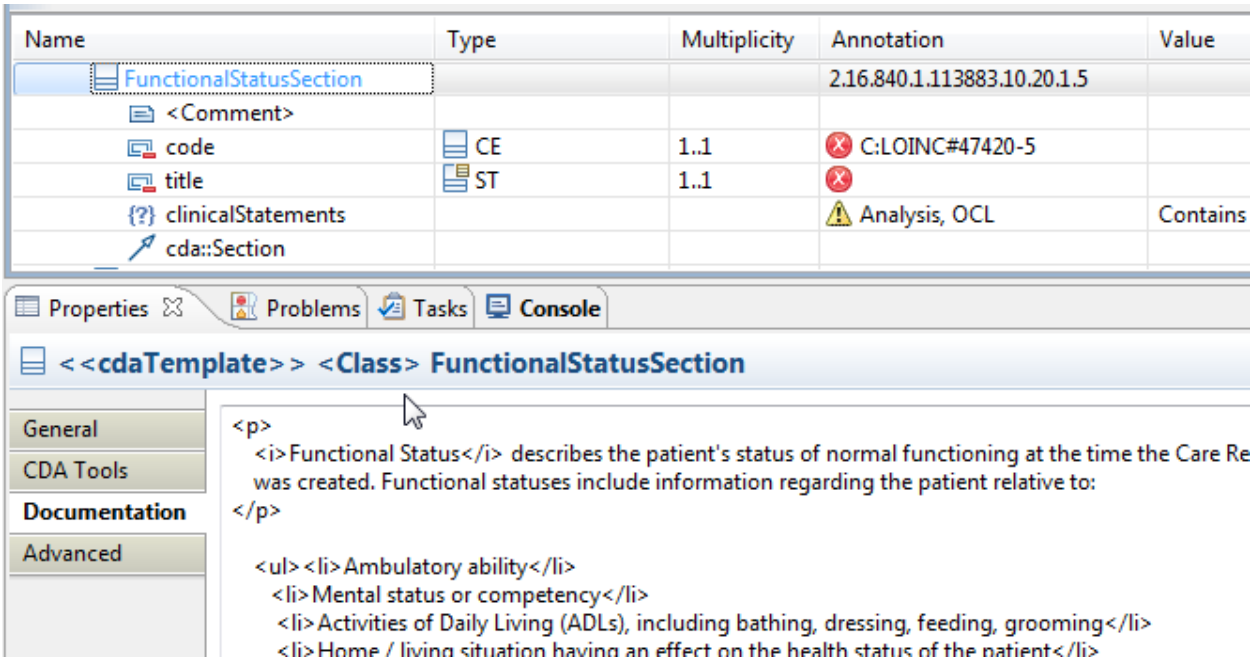


Documentation on Model Elements

A complete CDA implementation guide should include descriptive documentation for all template classes, and documentation on attributes, associations, or other constraints as necessary to explain how that content is expected to be used. All of this documentation is included in the generated implementation guide publication.

- UML includes a general Comment model element, and one or more Comments may be attached to any other model element. UML Comment is used to capture all documentation in these models.
- A comment's text may include limited use of presentation markup, however this is not required and should not be used unless it significantly improves presentation.
    - The most common markup is used to divide the text into multiple paragraphs. Wrap the text in xxx tags.
    - Emphasis with bold or italics: or
    - Unordered lists: xxx
    - Must be valid DITA content tags, but common tags like those listed here are the same as XHTML markup.
- If documentation includes XML style tags, e.g. `<assignedEntity>`, then the special characters must be escaped: `&lt;assignedEntity&gt;`
- Other XML character entities must be expanded with numerical equivalent
    - &, must use &
    - Etc.
- A CDA class attribute without constraints may be included strictly for the purpose of adding documentation about intended use. In this case, do not assign validation severity.

In the example MDHT screenshot shown below, documentation on a template class includes some basic presentation markup.



Glossary

TODO: complete this list and add glossary definitions

- Package
- Package Import
- Element Import
- Class
- Generalization
- Property
- Association
- Constraint
- Comment
- Enumeration
- Enumeration Literal
- Stereotype

References

- Core Principles and Properties of HL7 Version 3 Models
  - defines: Concept Domain, Code System, Value Set
  - http://www.hl7.org/v3ballot/html/infrastructure/coreprinciples/v3modelcoreprinciples.html
- Common Terminology Service 2 (CTS-2)
  - http://informatics.mayo.edu/cts2/index.php/Main_Page