MDHT CDA Tools User Guide

Contents

Introduction.......3
Standards Developers and Users.......4
Review CDA templates.......4
Open a model.......4
Using the MDHT UML Editor.......4
Create a new template.......7
Add template constraints.......10
Change attribute list.......10
Attribute constraints.......11
Association constraints.......15
Other general constraints.......17
Severity and message.......19
Publish an Implementation Guide.......20
Create a new Implementation Guide.......21
Standards Implementers.......23
Model-Driven Development Process Overview.......23
UML Profile for CDA.......23
CDA Template Stereotype.......24
Code Generation Support Stereotype.......24
Vocabulary Specification Stereotype.......24
Entry Stereotype.......24
Entry Relationship Stereotype.......25
Null Flavor Stereotype.......25
Text Value Stereotype.......25
Model-to-Model Transformation.......25
Invoke the Model-to-Model Transformation via Ant.......25
Generated OCL Constraints.......25
Runtime API.......26
Code Generation.......26
Producing an XML instance (serialization).......32
Consuming an XML instance (deserialization).......32
Validation.......33

Introduction

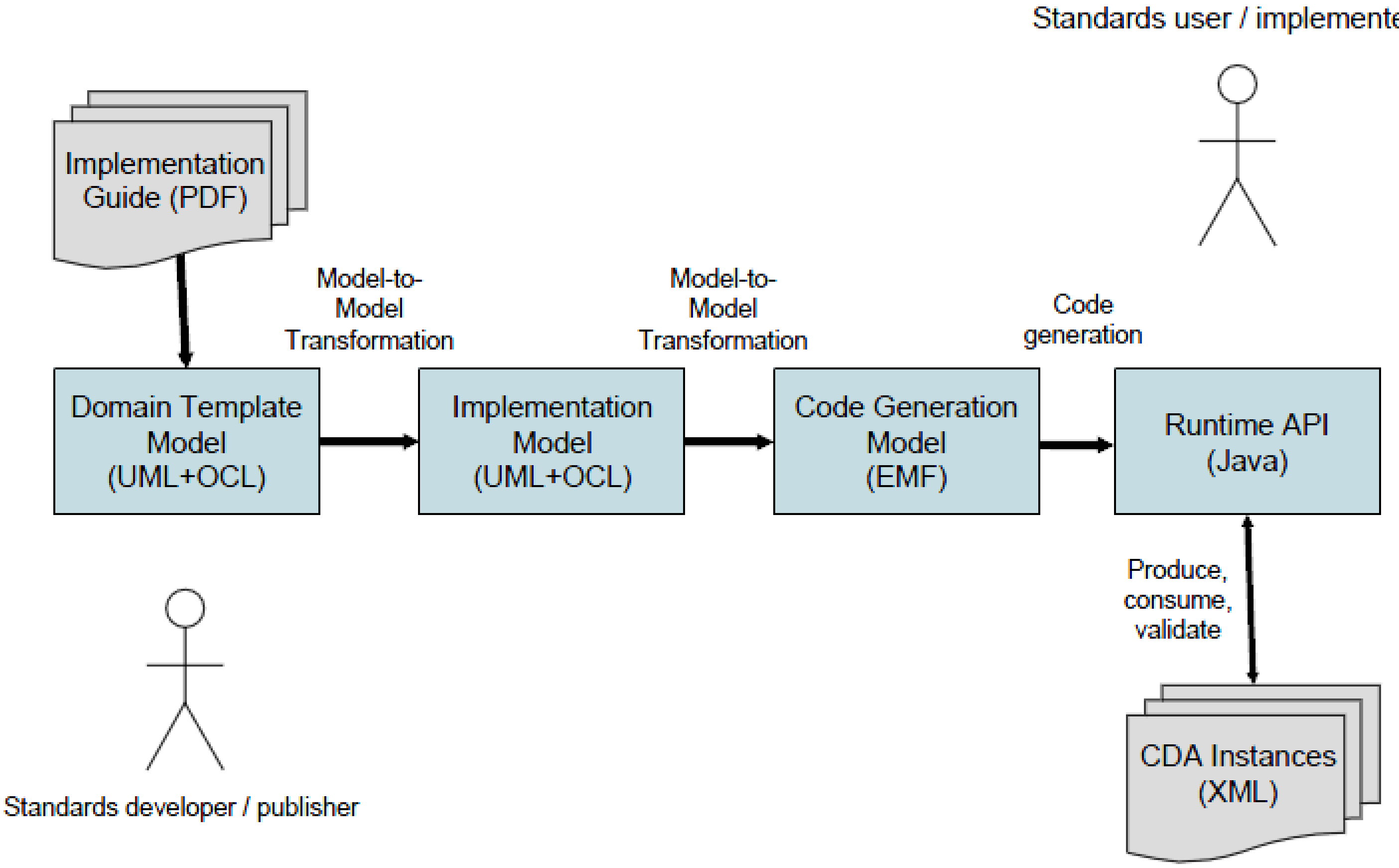
Overview of the Model-Driven Health Tools (MDHT) CDA Tools

CDA Tools is a component within the Model-Driven Health Tools (MDHT) project and provides UML-based tools for constraining HL7 Clinical Document Architecture (CDA) to create reusable templates and implementation guides. Tooling for consumers and implementers of CDA is produced by generating runtime components from models of implementation guides. Objectives of the CDA Tools are to:

- Accelerate and lower cost of adopting CDAr2 standard.
- Provide a model-driven framework for generating runtime API that supports:
  - Domain specific API (e.g. BloodPressureReading instead of Observation)
  - Construction of instances that conform to one or more templates
  - Consumption of XML instances that deserialize into appropriate template
- Support the validation of instances against constraints defined in model.

User Roles:

- Healthcare IT Standards Developer/Publisher
  - Create new models/templates
  - Combine and extend existing models
  - Publish Implementation Guides (IG), IHE Profiles, Data Dictionaries
- Healthcare IT Standards User/Implementer
  - Use generated runtime API in healthcare data exchange applications (e.g. EMR adapters to export/import CDA instances)
  - Minor modifications to existing models/templates


Standards user / implemente
Implementation Guide (PDF)
Model-to-Model Transformation
Model-to-Model Transformation
Code generation
Domain Template Model (UML+OCL)
Implementation Model (UML+OCL)
Code Generation Model (EMF)
Runtime API (Java)
Produce, consume, validate
CDA Instances (XML)
Standards developer / publisher

Standards Developers and Users

Introduction to roles and requirements of CDA standards development and use.

TODO: need to expand description of CDA template and IG development process.

Roles of the standards developer and user:

- Create new models/templates
- Combine and extend existing models
- Publish Implementation Guides (IG), IHE Profiles, Data Dictionaries

Review CDA templates

The MDHT UML Editor provides a simplified approach to browsing and editing UML models that is especially well suited to CDA template design.

The MDHT project provides an open source UML editing solution that may be used alone, or integrated into other complete UML modeling solutions. With only minor differences in the way a table editor is opened, the same solution described below is available within Rational Software Modeler (RSM) and Eclipse Papyrus. Other Eclipse-based UML tools also may be compatible, but have not been tested.

There are several key features in this MDHT UML Editor:

- A tree navigator view of UML content in the Project Explorer.
- A spreadsheet-like table where model content may be viewed and edited.
- Properties view tabs for easy access to most common UML model value.
- Template Constraint Dialog for adding attribute constraints.
- Model validation with error markers (limited scope in this milestone).

This editing solution only supports UML classes and enumerations, the typical content of a UML class diagram, but without the diagram. Many analysts and developers of CDA templates and message structures use Excel spreadsheets for analysis. This table editor provides similar functionality, with the significant benefit that the underlying content is saved as a valid UML 2.1 model. This model may be input directly into Java code generation or reporting tools that are based on UML. And the model may be enhanced with class diagrams using other full-featured and commercial UML modeling tools.

Open a model

Open a `.uml` model file using the Project Explorer view.

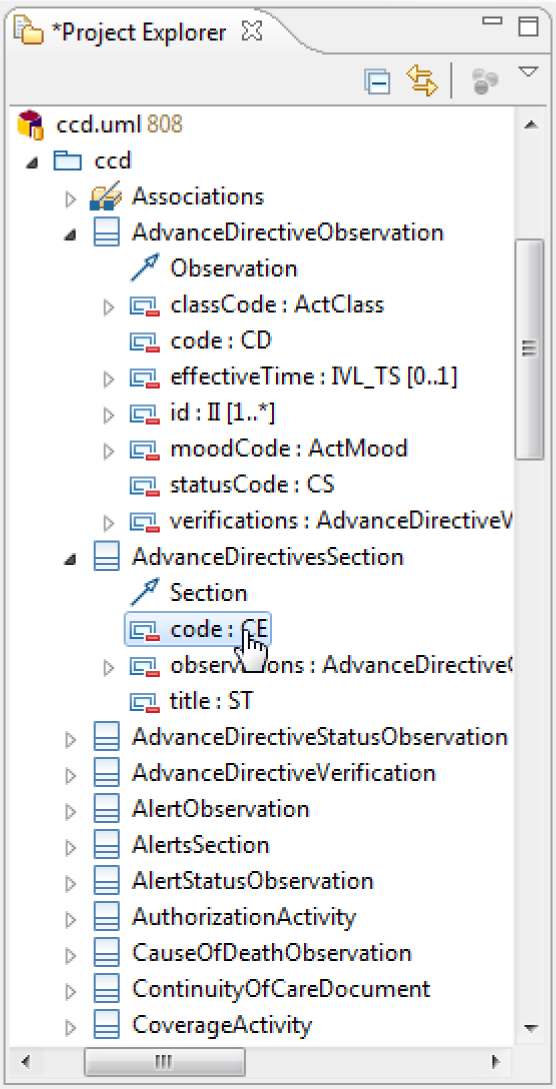
1. Assure that you are in the Resource Perspective, or some other perspective that has the Project Explorer view open. Note that the Java perspective Package Explorer view will **not** show the model tree navigator.
2. Open a UML model file (with extension `.uml`) by double-clicking that file in the Project Explorer, or right-clicking the file and choosing Open Model from the context menu.
3. Open the UML Table Editor by double-clicking on a `.uml` model file, or selecting **Open With** > **UML Table** from the context menu of a model element in the navigator tree. A *model element* is a package or class contained in the model; it is not the `.uml` file.

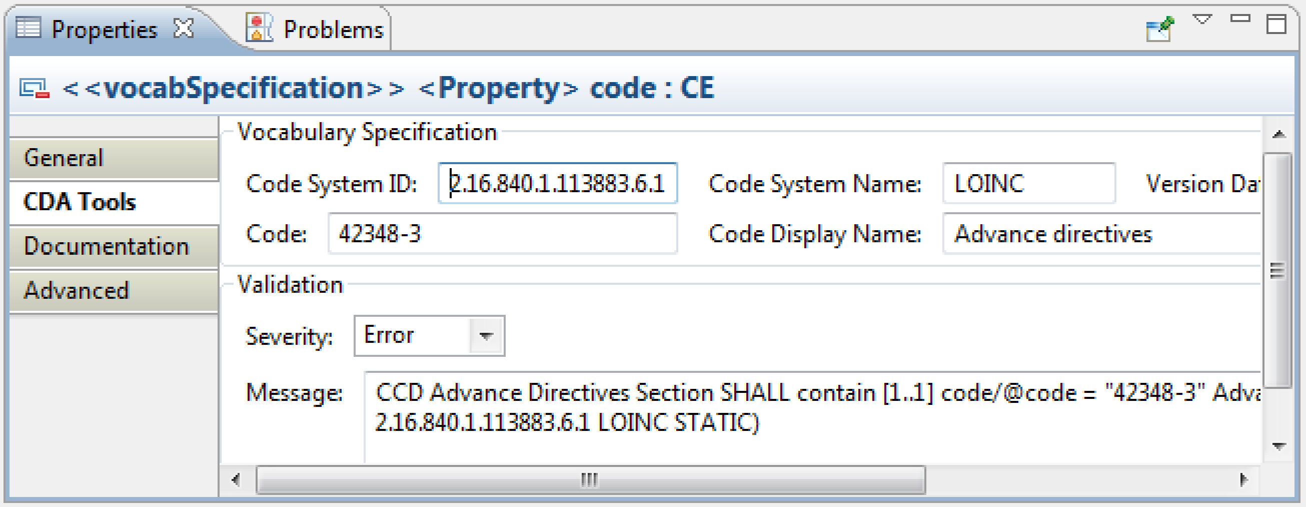
Using the MDHT UML Editor

Describes using the MDHT UML Editor to review the HL7 Continuity of Care (CCD) template model.

Let's start with an example of using the MDHT UML Editor to browse the HL7 Continuity of Care (CCD) template model. This is intended to be an *easy* UML editor, so not every aspect of UML is visible or editable, but we attempted to make all characteristics important to CDA template design easily accessible. We have more enhancements planned, but please let us know what would make it easier to use!

- Select a model element in either the Project Explorer navigator or in the UML Table Editor. The detailed properties of that element are shown in the Properties view. The properties are split into several tabs, one of which displays extended metadata that is specific to CDA template models.





<<vocabSpecification>> <Property> code : CE

Vocabulary Specification

Code System ID: 2.16.840.1.113883.6.1 Code System Name: LOINC Version Da

Code: 42348-3 Code Display Name: Advance directives

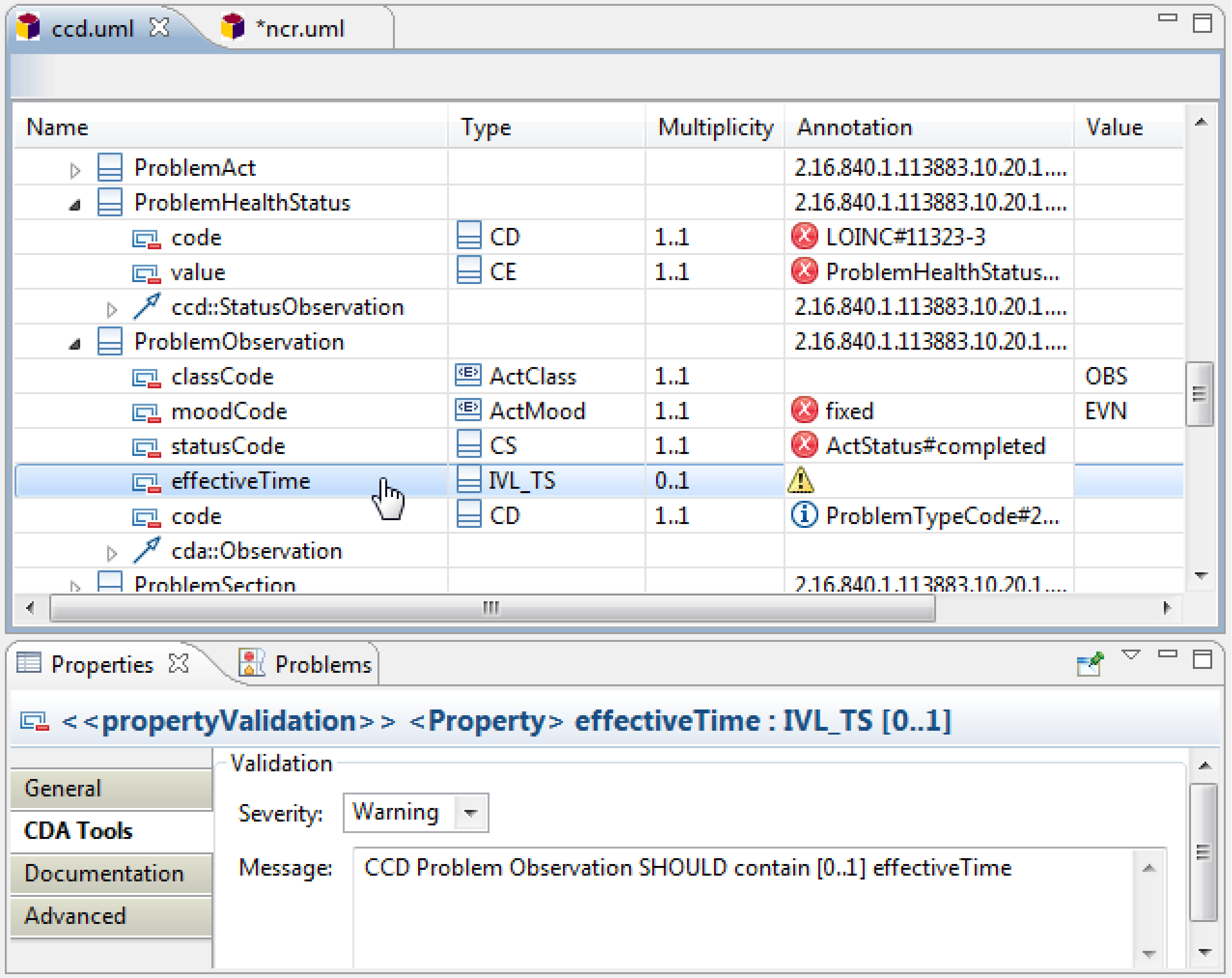
Validation

Severity: Error

Message: CCD Advance Directives Section SHALL contain [1..1] code/@code = "42348-3" Adva
2.16.840.1.113883.6.1 LOINC STATIC)

The Annotations table column shows a summary of HL7 metadata that is applied to a model element using UML stereotypes. Using the MDHT UML Editor, you don't need to know the details of how UML stereotypes are used. The CDA Tools properties tab displays entry fields for values that are appropriate for the selected model element.

In the example below, a code system constraint is assigned to the 'code' attribute, and a severity of Warning (a.k.a. SHOULD) is specified along with a message to be displayed when CDA documents are validated that don't satisfy this constraint. The severity level is shown as a graphical icon in the Annotation column.



Name	Type	Multiplicity	Annotation	Value
ProblemAct			2.16.840.1.113883.10.20.1....	
ProblemHealthStatus			2.16.840.1.113883.10.20.1....	
code	CD	1..1	⊗ LOINC#11323-3	
value	CE	1..1	⊗ ProblemHealthStatus...	
ccd::StatusObservation			2.16.840.1.113883.10.20.1....	
ProblemObservation			2.16.840.1.113883.10.20.1....	
classCode	ActClass	1..1		OBS
moodCode	ActMood	1..1	⊗ fixed	EVN
statusCode	CS	1..1	⊗ ActStatus#completed	
effectiveTime	IVL_TS	0..1	⚠	
code	CD	1..1	ⓘ ProblemTypeCode#2...	
cda::Observation				
ProblemSection			2.16.840.1.113883.10.20.1....	

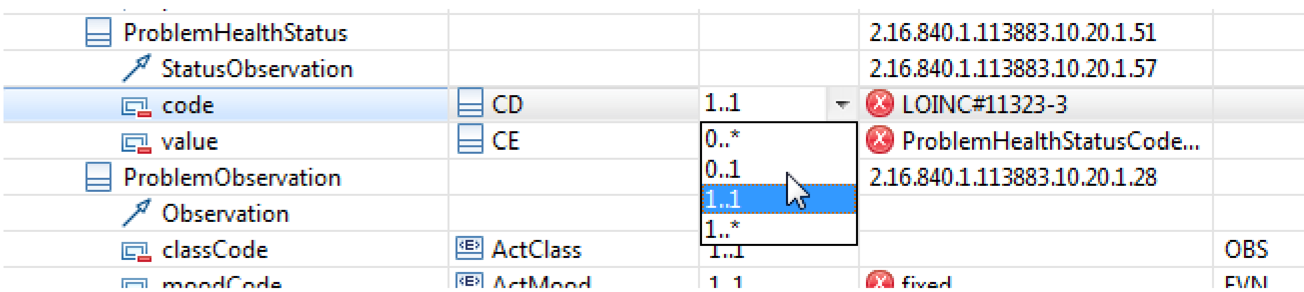
<<propertyValidation>> <Property> effectiveTime : IVL_TS [0..1]

Validation

Severity: Warning

Message: CCD Problem Observation SHOULD contain [0..1] effectiveTime

Many of the model element properties may be edited within the table cells. To activate editing (if allowable), either double-click on a cell or press the Enter key (NOTE: the table row must be selected/highlighted *before* editing a cell within that row). In the example shown below, a class attribute's multiplicity is edited using a pull-down list. Values shown in the Annotations column are a summary of more advanced HL7 metadata; these may be modified using the CDA Tools tab in the Properties view.



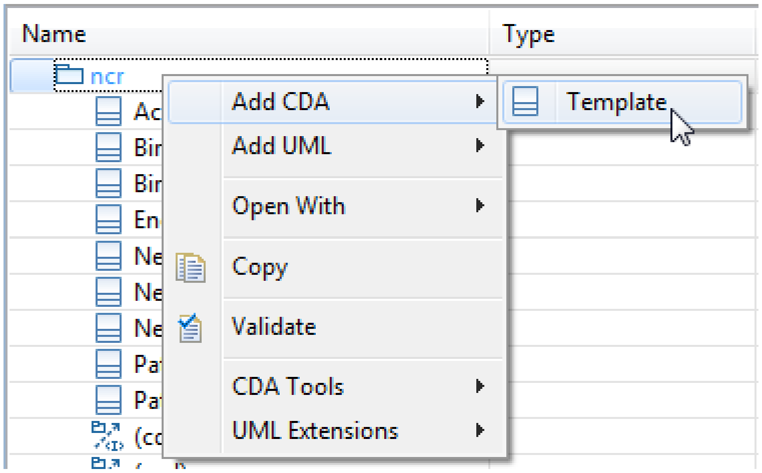
ProblemHealthStatus			2.16.840.1.113883.10.20.1.51	
StatusObservation			2.16.840.1.113883.10.20.1.57	
code	CD	1..1	⊗ LOINC#11323-3	
value	CE		⊗ ProblemHealthStatusCode...	
ProblemObservation			2.16.840.1.113883.10.20.1.28	
Observation				
classCode	ActClass	1..1		OBS
moodCode	ActMood	1..1	⊗ fixed	EVN

The data type of a class attribute may be modified in either the table cell or using the General tab in the Properties view. The attribute type is selected using a dialog that allows searching for any model element in one of the *currently* *open model(s)*. If several models are open simultaneously, then classes from all models are included in the dialog list. You can quickly narrow the selection by typing the first few characters of the class name you are searching for.

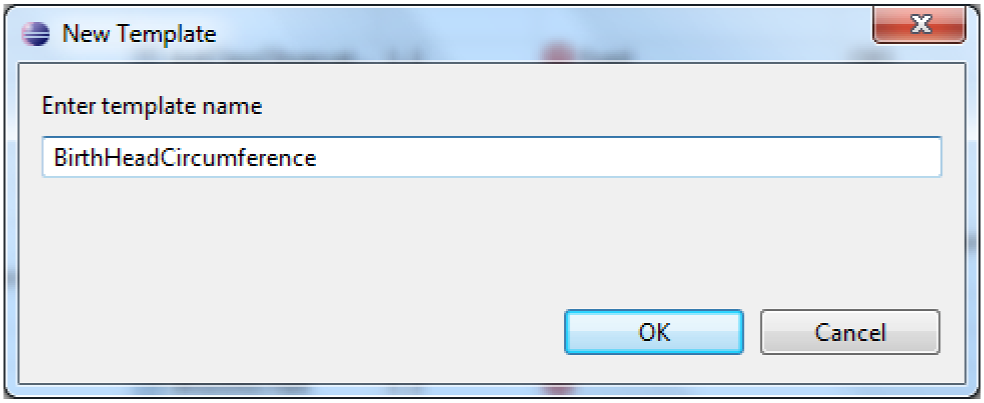
Create a new template

Create a new template class, choose its base type, and select initial set of constrained attributes.

1. Select the table row for the top model package.
2. To add a new template, right-click on the model Package (e.g. 'ncr' or 'hitsp'), select **Add CDA** > **Template**



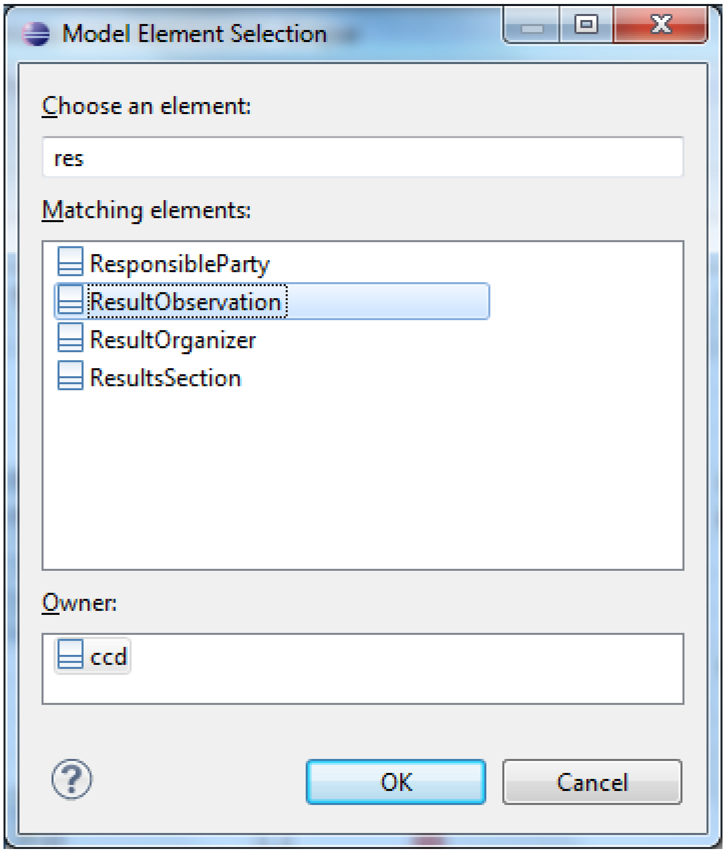
3. Enter the new template class name, using UpperCamelCase format.



Enter template name

BirthHeadCircumference

4. Select the base class that this template restricts, either a class from CDA or a parent template that this new template must conform to. You can type the first few characters of a class name and the Matching elements list will be limited to a few choices. Classes from all *currently open models* will be shown in this list. If necessary, open the model containing the base class you are searching for before invoking this dialog.



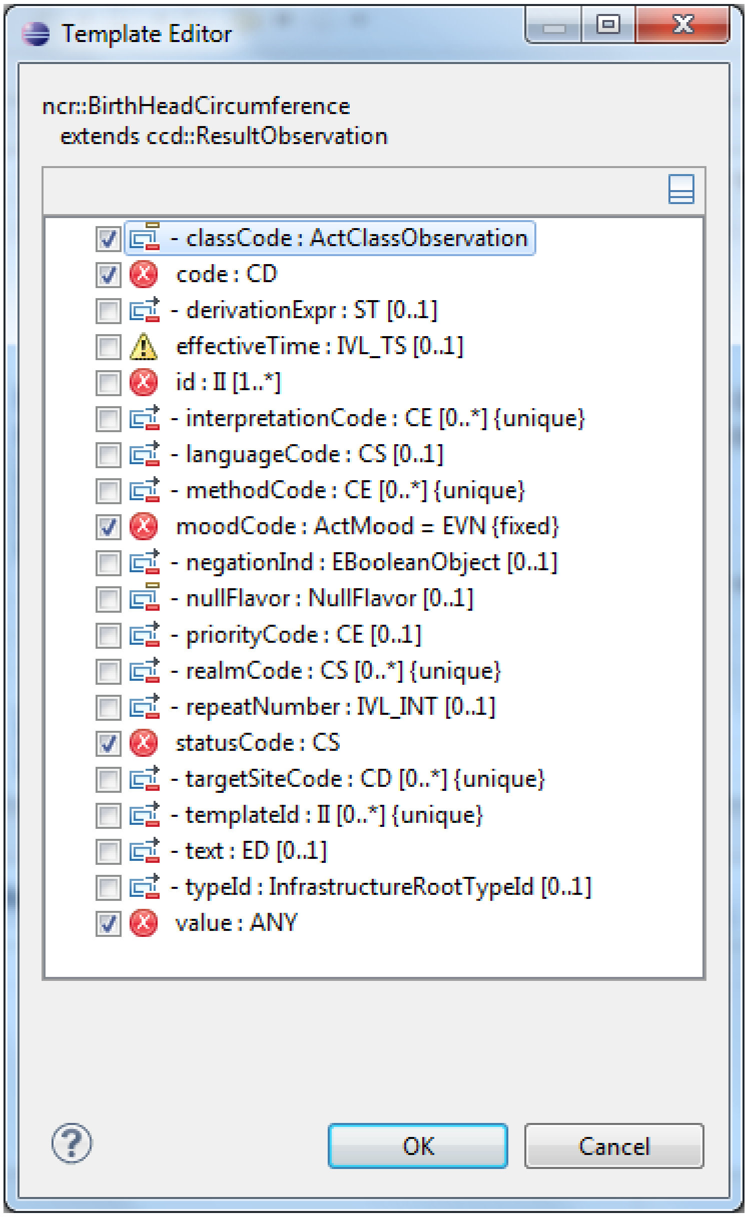
Choose an element: res

Matching elements:

- ResponsibleParty
- ResultObservation
- ResultOrganizer
- ResultsSection

Owner: ccd

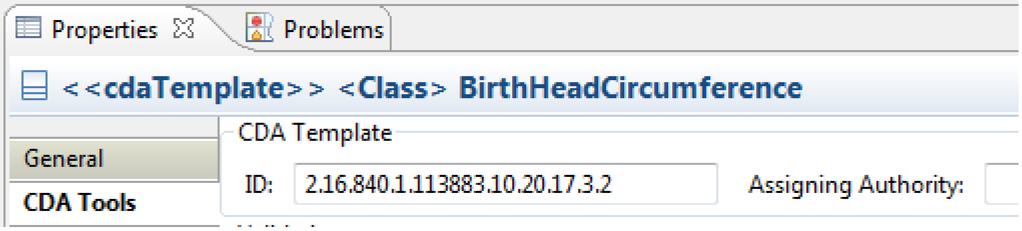
5. Add a check mark beside each inherited attribute that you want to constrain in this new template.



ncr::BirthHeadCircumference
extends ccd::ResultObservation

- ☑ - classCode : ActClassObservation
- ☑ code : CD
- □ - derivationExpr : ST [0..1]
- □ effectiveTime : IVL_TS [0..1]
- □ id : II [1..*]
- □ - interpretationCode : CE [0..*] {unique}
- □ - languageCode : CS [0..1]
- □ - methodCode : CE [0..*] {unique}
- ☑ moodCode : ActMood = EVN {fixed}
- □ - negationInd : EBooleanObject [0..1]
- □ - nullFlavor : NullFlavor [0..1]
- □ - priorityCode : CE [0..1]
- □ - realmCode : CS [0..*] {unique}
- □ - repeatNumber : IVL_INT [0..1]
- ☑ statusCode : CS
- □ - targetSiteCode : CD [0..*] {unique}
- □ - templateId : II [0..*] {unique}
- □ - text : ED [0..1]
- □ - typeId : InfrastructureRootTypeId [0..1]
- ☑ value : ANY

6. Enter the Template ID in the CDA Tools tab of the Properties view.

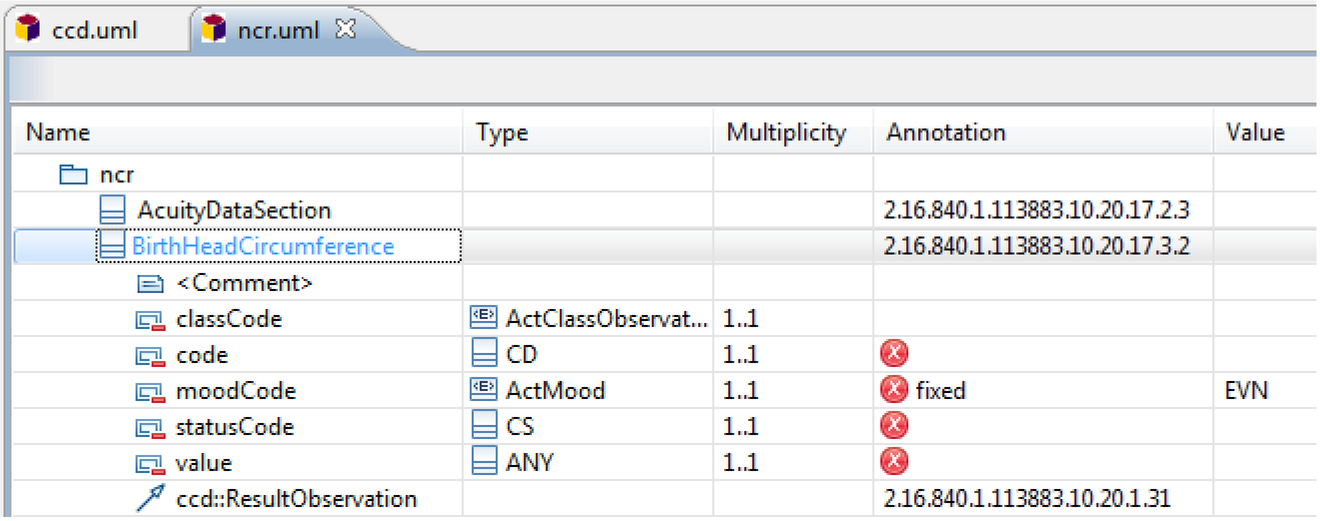


<<cdaTemplate>> <Class> BirthHeadCircumference

CDA Template

ID: 2.16.840.1.113883.10.20.17.3.2 Assigning Authority:

The following screen shows the new template class displayed in the table editor.



Name	Type	Multiplicity	Annotation	Value
ncr				
AcuityDataSection			2.16.840.1.113883.10.20.17.2.3	
BirthHeadCircumference			2.16.840.1.113883.10.20.17.3.2	
<Comment>				
classCode	ActClassObservat...	1..1		
code	CD	1..1	✖	
moodCode	ActMood	1..1	✖ fixed	EVN
statusCode	CS	1..1	✖	
value	ANY	1..1	✖	
ccd::ResultObservation			2.16.840.1.113883.10.20.1.31	

Add template constraints

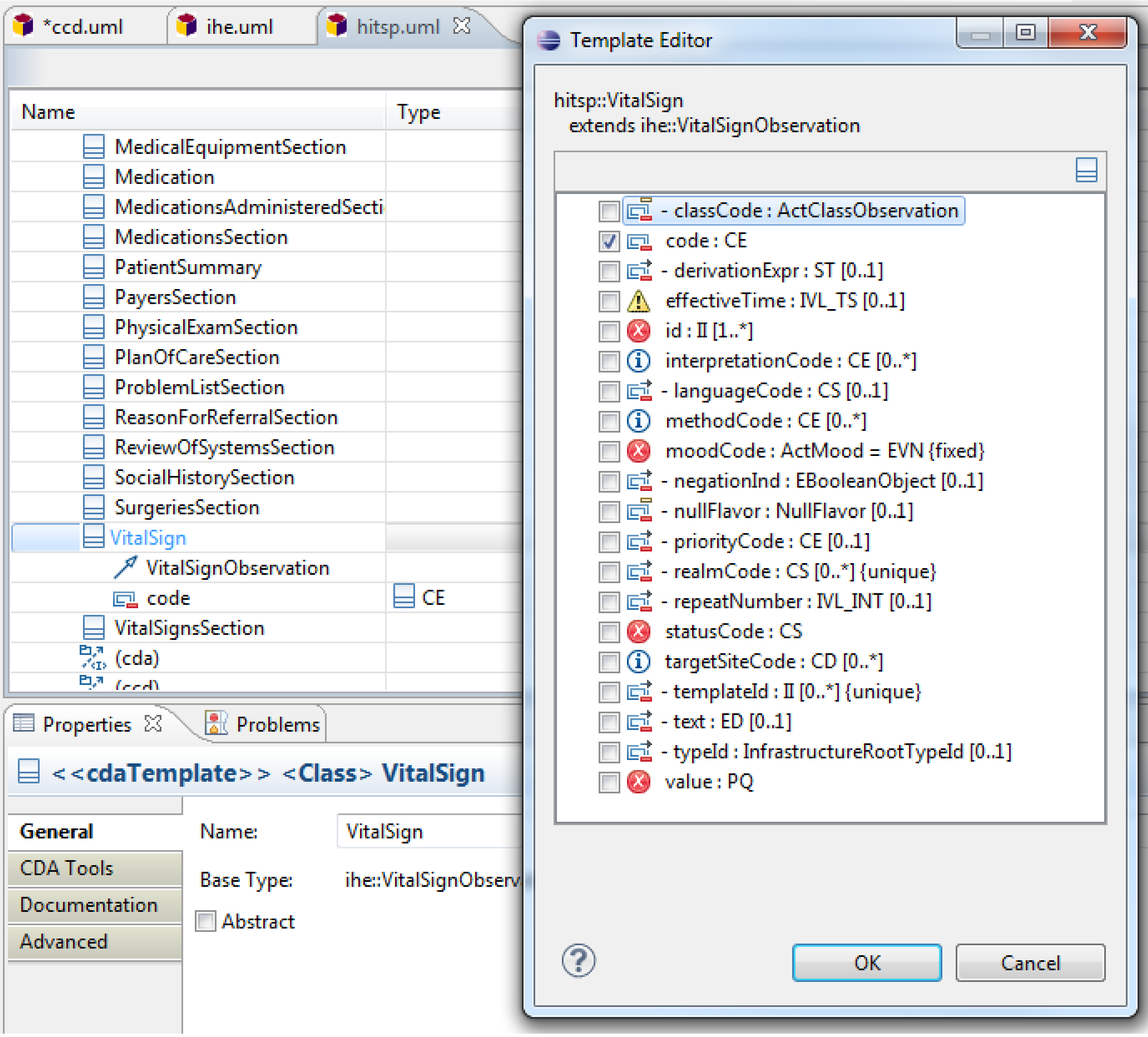
Describes all the kinds of constraints that may be added to a template class.

Change attribute list

Change the list of constrained attributes in an existing template.

To change the list of constrained attributes in an existing template, right-click on a template class, select **CDA Tools** > **Open Template Editor**

In the example below, a VitalSign template in the HITSP model specializes a VitalSignObservation template in the IHE model, which specializes ResultObservation in the CCD model, which specializes Observation in CDA. The HITSP template adds an attribute constraint for 'code', but other attributes have been constrained in the IHE and CCD templates. This dialog shows the icons for info/warning/error beside inherited constraints so that you can see the aggregate effect of all templates.



*ccd.uml | ihe.uml | hitsp.uml

Name	Type
MedicalEquipmentSection	
Medication	
MedicationsAdministeredSecti	
MedicationsSection	
PatientSummary	
PayersSection	
PhysicalExamSection	
PlanOfCareSection	
ProblemListSection	
ReasonForReferralSection	
ReviewOfSystemsSection	
SocialHistorySection	
SurgeriesSection	
VitalSign	
VitalSignObservation	
code	CE
VitalSignsSection	
(cda)	
(ccd)	

Properties | Problems

<<cdaTemplate>> <Class> VitalSign

General | Name: VitalSign
CDA Tools | Base Type: ihe::VitalSignObserv
Documentation | □ Abstract
Advanced

Template Editor

hitsp::VitalSign
extends ihe::VitalSignObservation

- □ - classCode : ActClassObservation
- ☑ code : CE
- □ - derivationExpr : ST [0..1]
- □ effectiveTime : IVL_TS [0..1]
- □ id : II [1..*]
- □ interpretationCode : CE [0..*]
- □ - languageCode : CS [0..1]
- □ methodCode : CE [0..*]
- □ moodCode : ActMood = EVN {fixed}
- □ - negationInd : EBooleanObject [0..1]
- □ - nullFlavor : NullFlavor [0..1]
- □ - priorityCode : CE [0..1]
- □ - realmCode : CS [0..*] {unique}
- □ - repeatNumber : IVL_INT [0..1]
- □ statusCode : CS
- □ targetSiteCode : CD [0..*]
- □ - templateId : II [0..*] {unique}
- □ - text : ED [0..1]
- □ - typeId : InfrastructureRootTypeId [0..1]
- □ value : PQ

Attribute constraints

Describes each kind of constraint that may be applied to template attributes.

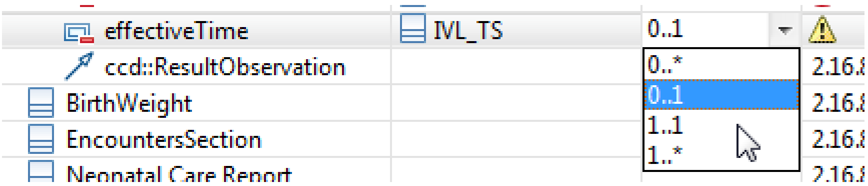
A template class contains attributes that restrict the content allowed by a parent template or CDA base type. A template attribute may define one or more constraints on the redefined attribute.

Multiplicity constraint

Constrain the multiplicity of a template attribute to be more restrictive than the same attribute in its parent class.

You can only constrain an attribute with a multiplicity that is more restrictive than its parent CDA class or template. For example, you can constraint `0..1` to be `1..1`, but you cannot change `1..1` to be `0..1`.

1. Select the table row for the attribute you want to constrain.
2. Select the Multiplicity table cell in that row using the mouse or by navigating with the keyboard arrow keys.
3. Either double-click in that cell or press Enter to show a list of multiplicity options, or type into the cell for other specialized constraints, e.g. `1..2`



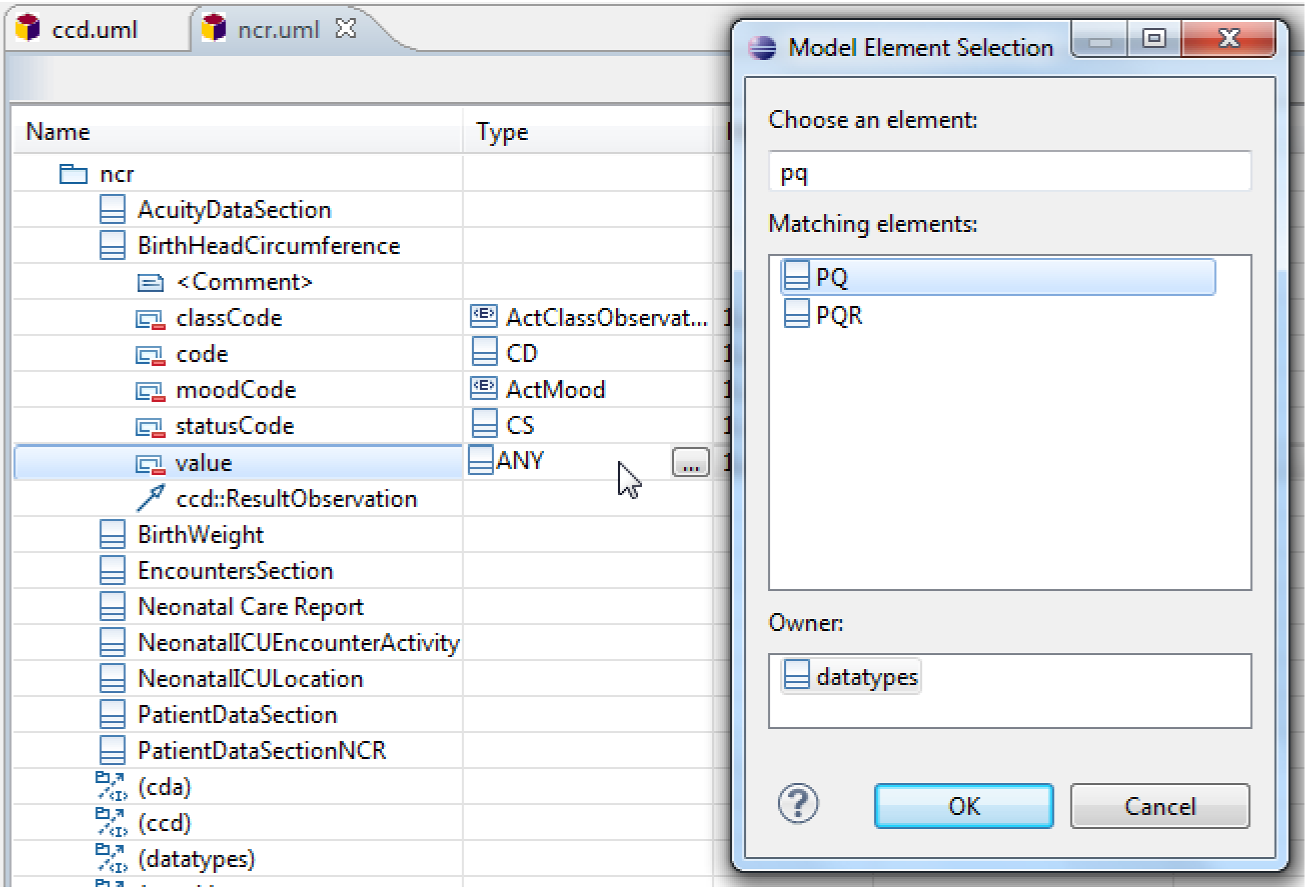
effectiveTime	IVL_TS	0..1 ▾	⚠
ccd::ResultObservation		0..*	2.16.8
BirthWeight		0..1	2.16.8
EncountersSection		1..1	2.16.8
Neonatal Care Report		1..*	2.16.8

Data type constraint

Constrain the data type of a template attribute to be more restrictive than the same attribute in its parent class.

1. Select the table row for the attribute you want to constrain.
2. Select the Type table cell in that row using the mouse or by navigating with the keyboard arrow keys.
3. Either double-click in that cell or press Enter to open a type selection dialog.

In this example, we are restricting the BirthHeadCircumference `value` data type from ANY to PQ. We expect to add more selection guidance in future milestones; for now, you are responsible for selecting an appropriate type restriction. In the future, model validation will also verify model integrity, including template constraints.



Name	Type
ncr	
AcuityDataSection	
BirthHeadCircumference	
<Comment>	
classCode	ActClassObservat...
code	CD
moodCode	ActMood
statusCode	CS
value	ANY ...
ccd::ResultObservation	
BirthWeight	
EncountersSection	
Neonatal Care Report	
NeonatalICUEncounterActivity	
NeonatalICULocation	
PatientDataSection	
PatientDataSectionNCR	
(cda)	
(ccd)	
(datatypes)	

Choose an element: pq

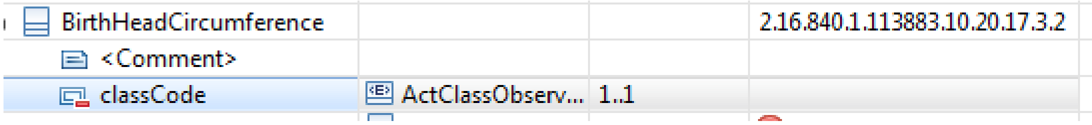
Matching elements:
PQ
PQR

Owner: datatypes

Fixed structural code value

Restrict a structural attribute (classCode, moodCode, typeCode) to a specified fixed code value.

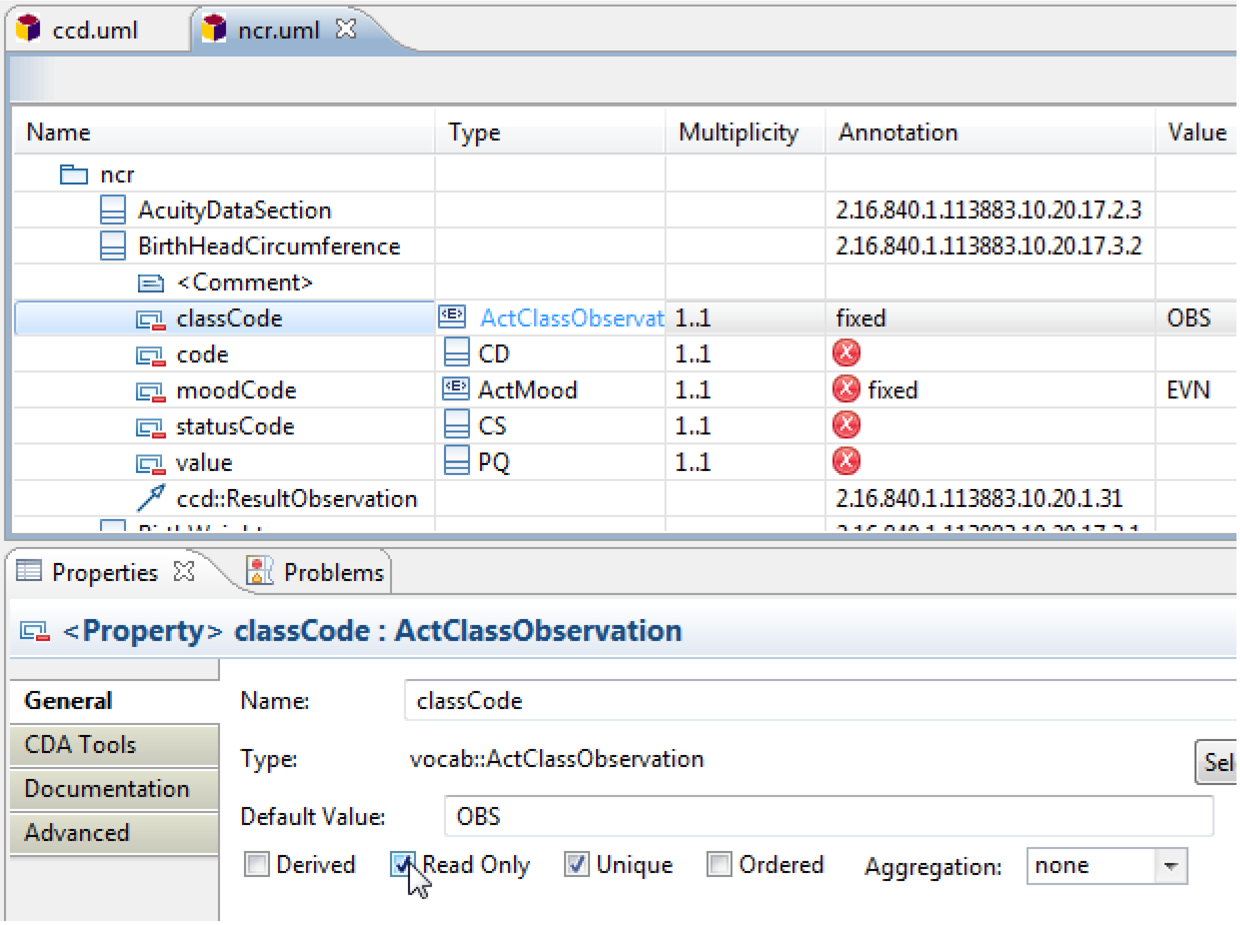
1. Select the table row for the attribute you want to constrain.



BirthHeadCircumference			2.16.840.1.113883.10.20.17.3.2
<Comment>			
classCode	ActClassObserv...	1..1	

2. In the Properties view General tab, enter a code in the **Default Value** field. For example, enter `OBS`.
3. Check the **Read Only** box.

The `classCode` attribute is now constrained to a fixed value of `OBS`.



Name	Type	Multiplicity	Annotation	Value
ncr				
AcuityDataSection			2.16.840.1.113883.10.20.17.2.3	
BirthHeadCircumference			2.16.840.1.113883.10.20.17.3.2	
<Comment>				
classCode	ActClassObservat	1..1	fixed	OBS
code	CD	1..1	⊗	
moodCode	ActMood	1..1	⊗ fixed	EVN
statusCode	CS	1..1	⊗	
value	PQ	1..1	⊗	
ccd::ResultObservation			2.16.840.1.113883.10.20.1.31	

<Property> classCode : ActClassObservation

Name: classCode

Type: vocab::ActClassObservation

Default Value: OBS

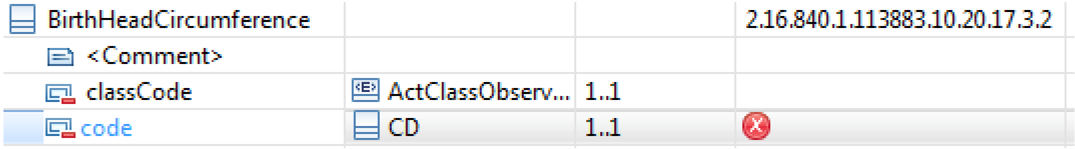
□ Derived ☑ Read Only ☑ Unique □ Ordered Aggregation: none

Vocabulary constraint

Specify the code system or value set ID and name, and optionally an individual code for coded attribute constraints.

The vocabulary constraint user interface fields are visible only when an attribute is selected with a data type that is a kind of `CD`.

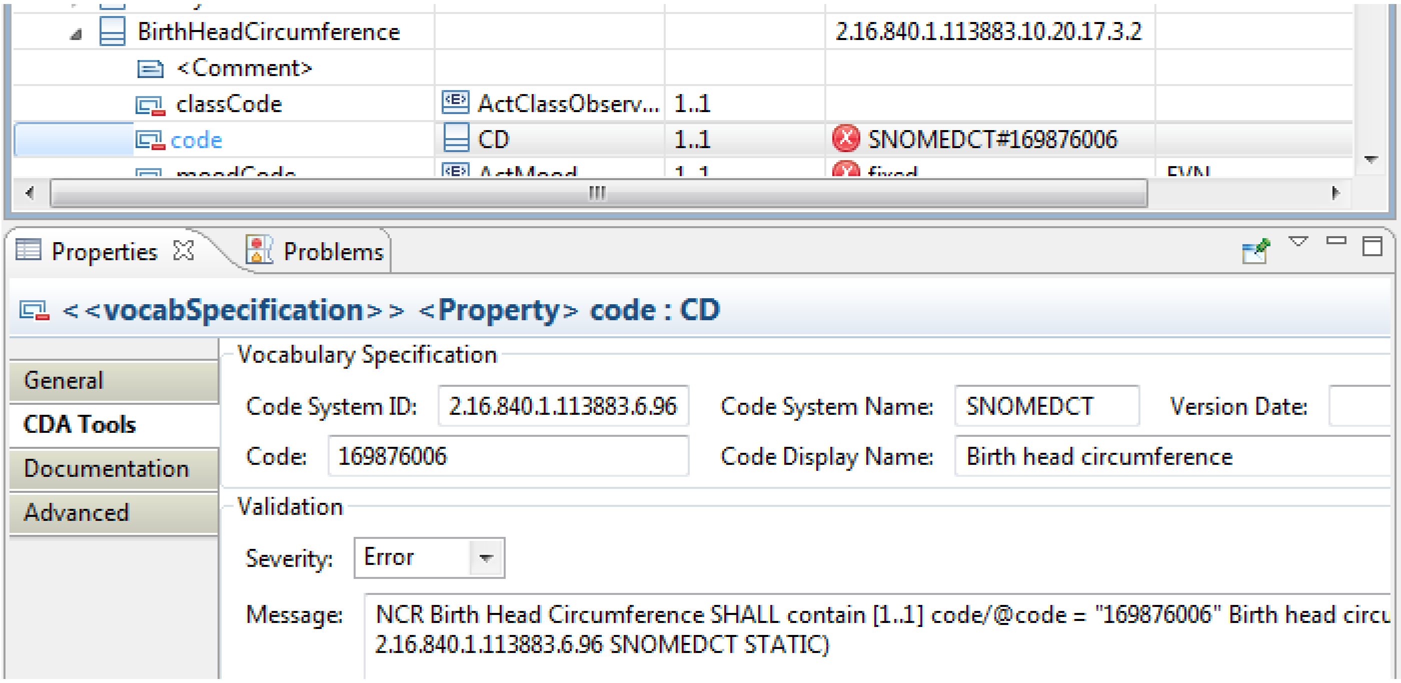
1. Select the table row for the attribute you want to constrain.



BirthHeadCircumference			2.16.840.1.113883.10.20.17.3.2
<Comment>			
classCode	ActClassObserv...	1..1	
code	CD	1..1	⊗

2. In the Properties view CDA Tools tab, enter the vocabulary constraints.

In this example, the `code` attribute is constrained to a specified code value from SNOMED CT. Notice that the table Annotation cell for the `code` attribute is automatically updated to show the vocabulary constraint that was entered in the Properties view. The icon proceeding the annotation indicates the constraint severity: Error, Warning, or Info.



BirthHeadCircumference			2.16.840.1.113883.10.20.17.3.2	
<Comment>				
classCode	ActClassObserv...	1..1		
code	CD	1..1	SNOMEDCT#169876006	
moodCode	ActMood	1..1	fixed	EVN

<<vocabSpecification>> <Property> code : CD

Vocabulary Specification

Code System ID: 2.16.840.1.113883.6.96 Code System Name: SNOMEDCT Version Date:

Code: 169876006 Code Display Name: Birth head circumference

Validation

Severity: Error

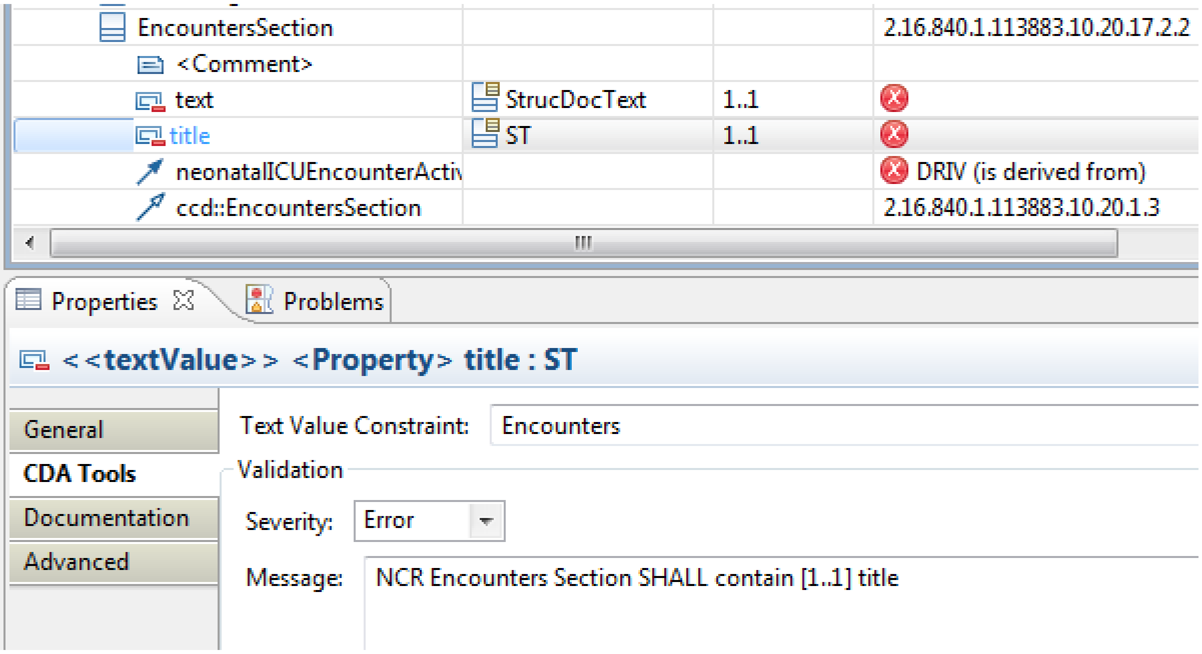
Message: NCR Birth Head Circumference SHALL contain [1..1] code/@code = "169876006" Birth head circu
2.16.840.1.113883.6.96 SNOMEDCT STATIC)

Text value constraint

Specify a text value constraint for attributes with a data type that is a kind of ED.

1. Select the table row for the attribute you want to constrain.
2. In the Properties view CDA Tools tab, enter the**Text Value Constraint**.

In this example, the EncountersSection title is constrained to the value `Encounters`.



EncountersSection			2.16.840.1.113883.10.20.17.2.2
<Comment>			
text	StrucDocText	1..1	
title	ST	1..1	
neonatalICUEncounterActiv			DRIV (is derived from)
ccd::EncountersSection			2.16.840.1.113883.10.20.1.3

<<textValue>> <Property> title : ST

Text Value Constraint: Encounters

Validation

Severity: Error

Message: NCR Encounters Section SHALL contain [1..1] title

Association constraints

Describes constraints on relationships between two template classes, e.g. that a Section template is required to contain a particular templated Observation.

Most CDA classes are derived from the RIM Act, so associations between templates (i.e. CDA classes that have been restricted via template specifications) must be connected using an ActRelationship. In our CDA modeling tools, the association is represented directly between two template classes and later implementation transformations will add the necessary ActRelationship intermediary.

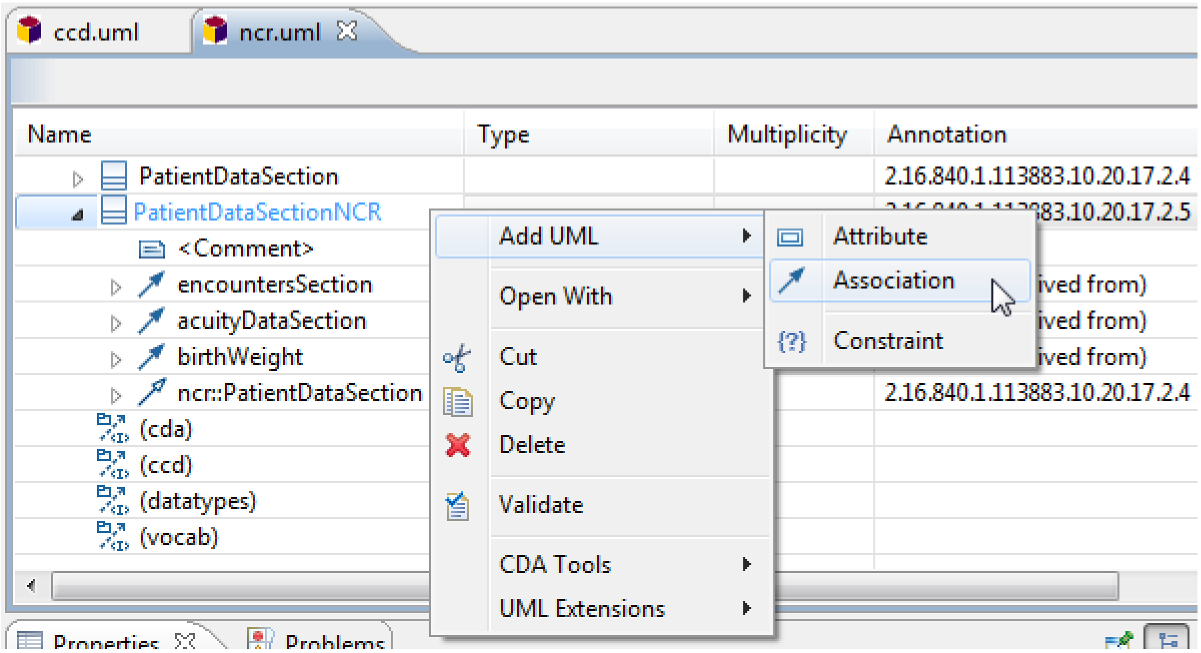
As part of defining the template association, a typeCode may be added as a restriction constraint, which will be used to generate the complete ActRelationship implementation. The CDA Tools do not yet support all types of act relationships in the CDA R2 base model; these are currently included:

- ClinicalDocument to Section (component)
- Section to ClinicalStatement (entry)
- ClinicalStatement to ClinicalStatement (actRelationship)

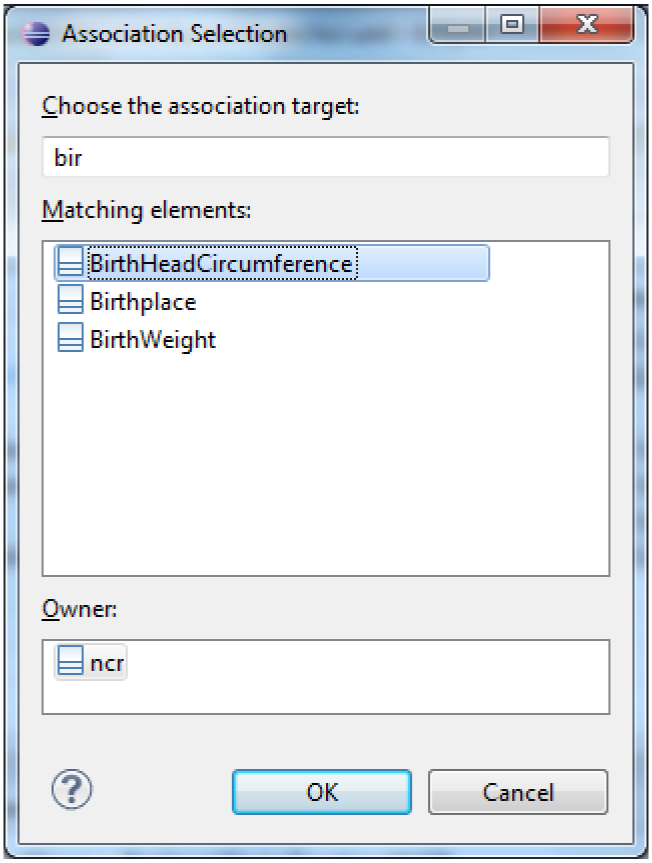
Create a new association

Create a new association constraint between two template classes.

1. Select the table row for the template class that is the source/origin end of the association.
2. To add a new association, right-click on the table row and select **Add UML** > **Association**



3. Select the association target class. Type the first few characters of the class name to filter the list. Classes are shown for all open models, which includes classes in other models such as CDA or CCD.



Association Selection

Choose the association target:

bir

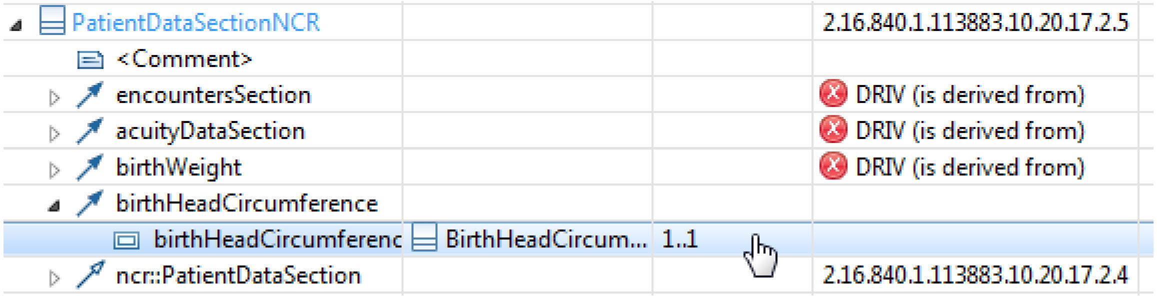
Matching elements:

- BirthHeadCircumference
- Birthplace
- BirthWeight

Owner:

ncr

4. Assign the target end multiplicity, if different from the default 1..1



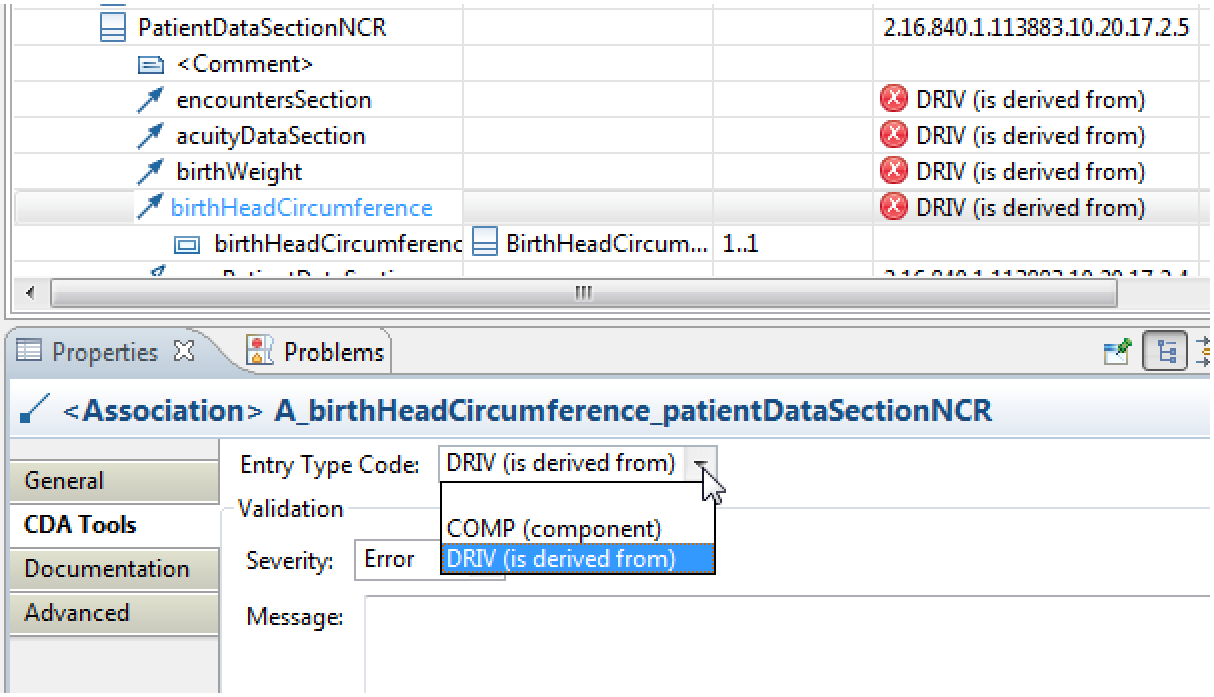
PatientDataSectionNCR			2.16.840.1.113883.10.20.17.2.5
<Comment>			
encountersSection			DRIV (is derived from)
acuityDataSection			DRIV (is derived from)
birthWeight			DRIV (is derived from)
birthHeadCircumference			
birthHeadCircumferenc	BirthHeadCircum...	1..1	
ncr::PatientDataSection			2.16.840.1.113883.10.20.17.2.4

Association type code constraint

Add a `typeCode` constraint for the relationship connecting two template classes.

1. Select the table row for an association.
2. In the Properties view CDA Tools tab, the **Entry Type Code** field will be visible if this tooling supports `typeCode` constraint for the source/target association you have selected. Select the desired type code from the pull-down list.

This example adds an association from PatientDataSectionNCR to BirthHeadCircumference, with multiplicity 1..1 and `typeCode` `DRIV`.



Entry Type Code: DRIV (is derived from)

Severity: Error

Other general constraints

Specify other constraints that cannot be expressed as basic attribute and association constraints.

Your objective should be to get all constraints in a formalized representation that is automatically generated into application code. The majority of template constraints may be expressed as UML attributes and associations, as described in other sections of this guide. However, a few constraints require more complex expressions. Our recommendation is to create a UML Constraint and to:

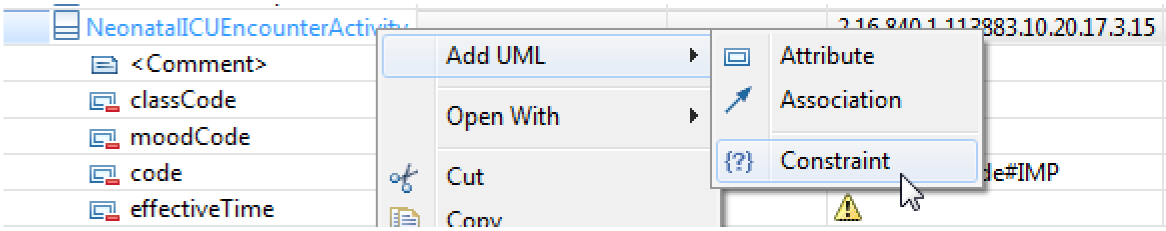
1. Add an *Analysis* language body that provides a human-readable description, and
2. Add an *OCL* language body that is a formal executable representation based on the CDA object model.

When an implementation guide is published from this model, the *Analysis* text is included in the document, whereas the *OCL* expression is used when generating Java code from the model.

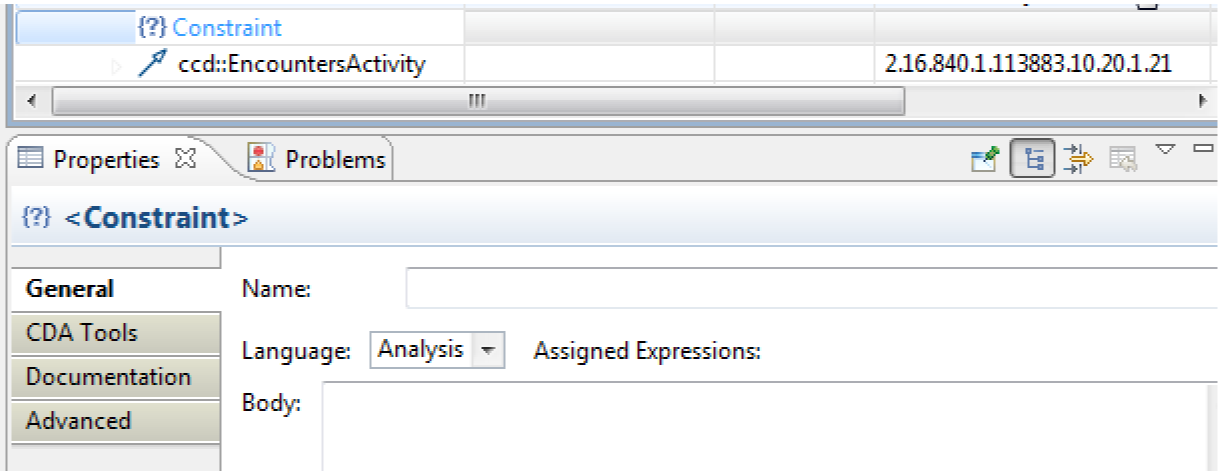
Analysis language constraint

Add informal constraint definition in human readable text.

1. Select the table row for the template class that is context for the new constraint.
2. To add a new constraint, right-click on the table row and select **Add UML** > **Constraint**



3. Select the table row with the new constraint (it has the general label 'Constraint'), as shown here. Enter the constraint name, select 'Analysis' from the Language pull-down list, and enter the constraint body.



{?} Constraint			
ccd::EncountersActivity			2.16.840.1.113883.10.20.1.21

{?} <Constraint>

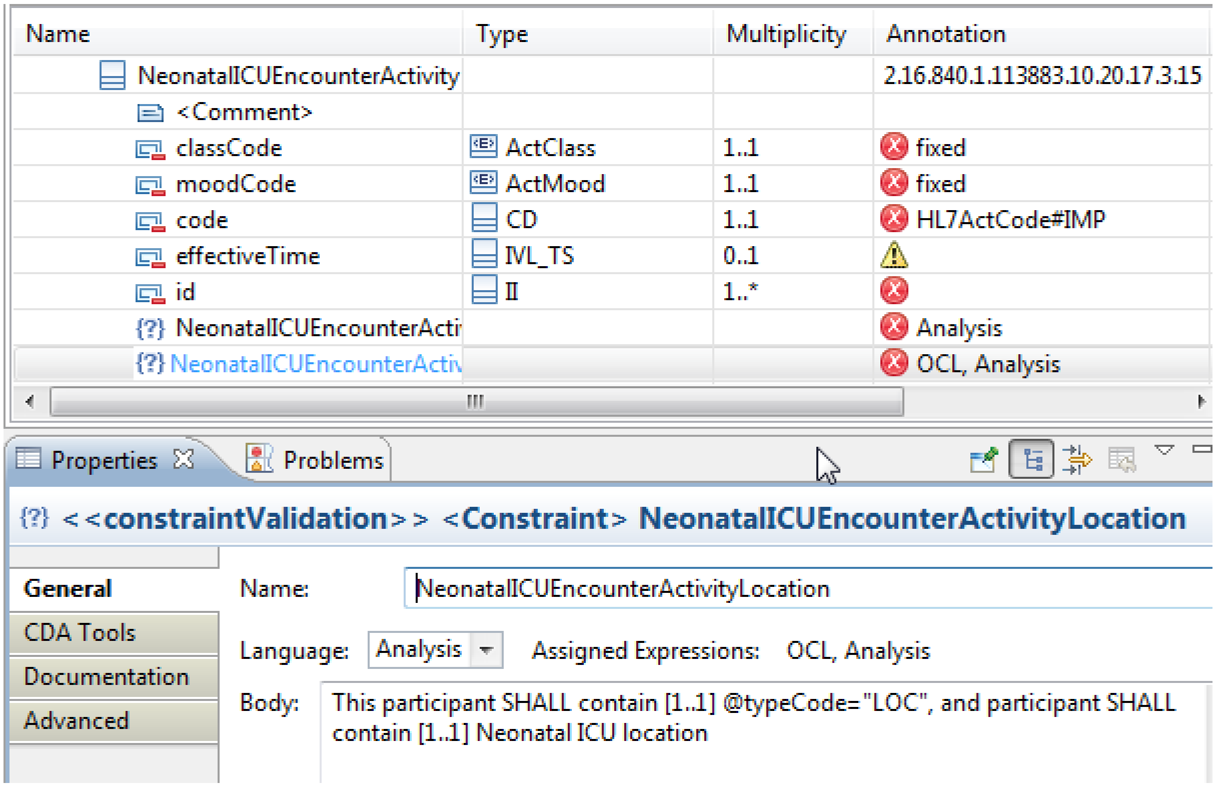
Name:

Language: Analysis

Assigned Expressions:

Body:

This is an example of a completed constraint within the Neonatal ICU Encounter Activity class.



Name	Type	Multiplicity	Annotation
NeonatalICUEncounterActivity			2.16.840.1.113883.10.20.17.3.15
<Comment>			
classCode	ActClass	1..1	⊗ fixed
moodCode	ActMood	1..1	⊗ fixed
code	CD	1..1	⊗ HL7ActCode#IMP
effectiveTime	IVL_TS	0..1	⚠
id	II	1..*	⊗
{?} NeonatalICUEncounterActi			⊗ Analysis
{?} NeonatalICUEncounterActiv			⊗ OCL, Analysis

{?} <<constraintValidation>> <Constraint> NeonatalICUEncounterActivityLocation

Name: NeonatalICUEncounterActivityLocation

Language: Analysis

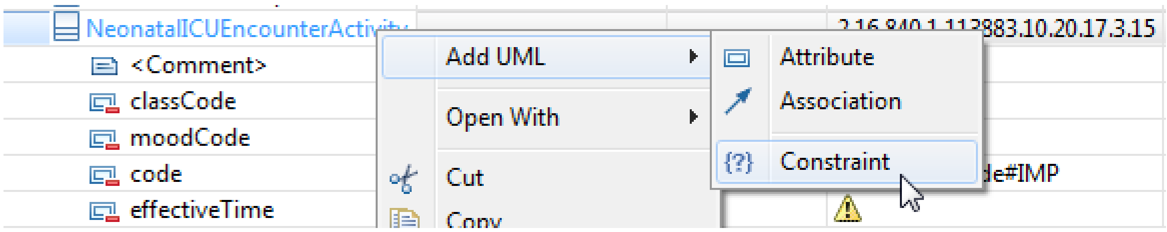
Assigned Expressions: OCL, Analysis

Body: This participant SHALL contain [1..1] @typeCode="LOC", and participant SHALL contain [1..1] Neonatal ICU location

OCL language constraint

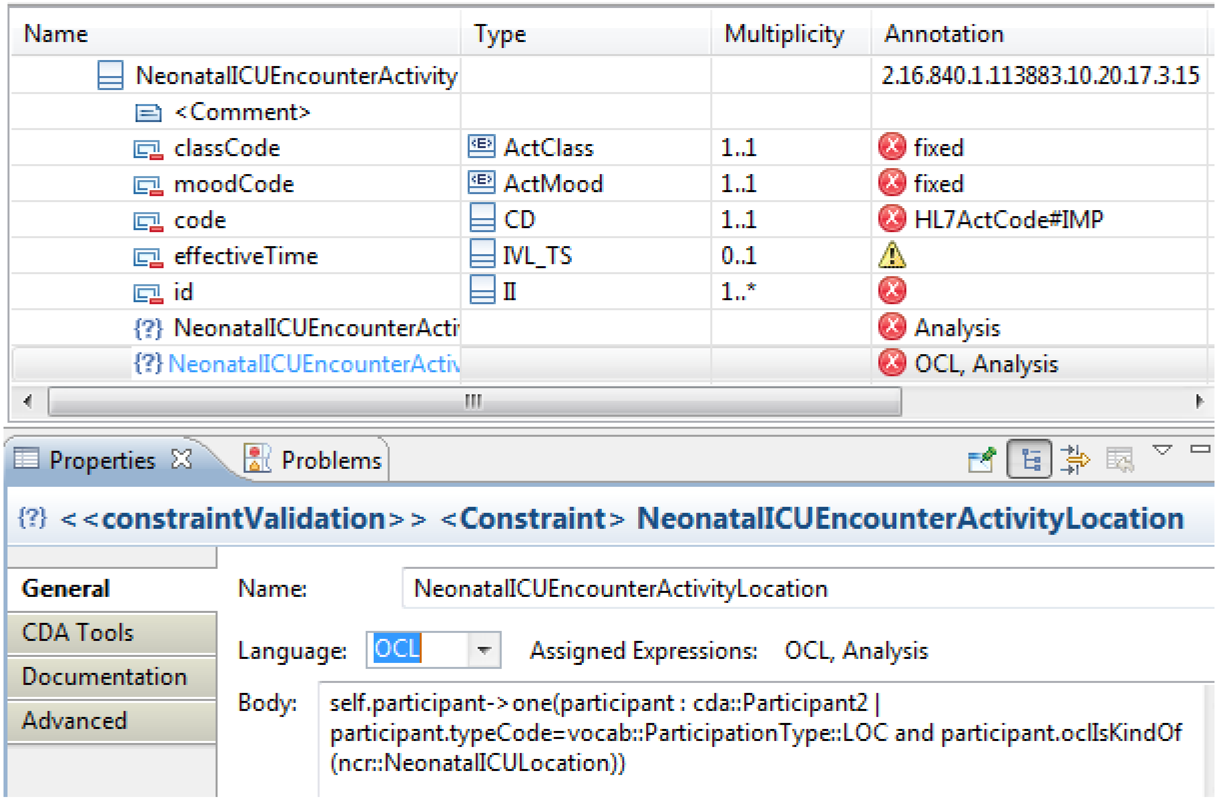
Add formal constraint definition in Object Constraint Language (OCL).

1. Select the table row for the template class that is context for the new constraint.
2. To add a new constraint, right-click on the table row and select **Add UML** > **Constraint**



3. Select the table row with the new constraint (it has the general label 'Constraint'), as shown here. Enter the constraint name, select 'OCL' from the Language pull-down list, and enter the constraint body.

This is an example of a completed constraint within the Neonatal ICU Encounter Activity class. Notice that this constraint has both Analysis and OCL languages assigned. You can switch between the two body values by changing the selection in the Language pull-down. Also notice that the Annotation table column displays which, if any, constraint languages have been entered.



Name	Type	Multiplicity	Annotation
NeonatalICUEncounterActivity			2.16.840.1.113883.10.20.17.3.15
<Comment>			
classCode	ActClass	1..1	☒ fixed
moodCode	ActMood	1..1	☒ fixed
code	CD	1..1	☒ HL7ActCode#IMP
effectiveTime	IVL_TS	0..1	⚠
id	II	1..*	☒
{?} NeonatalICUEncounterActi			☒ Analysis
{?} NeonatalICUEncounterActiv			☒ OCL, Analysis

{?} <<constraintValidation>> <Constraint> NeonatalICUEncounterActivityLocation

Name: NeonatalICUEncounterActivityLocation

Language: OCL Assigned Expressions: OCL, Analysis

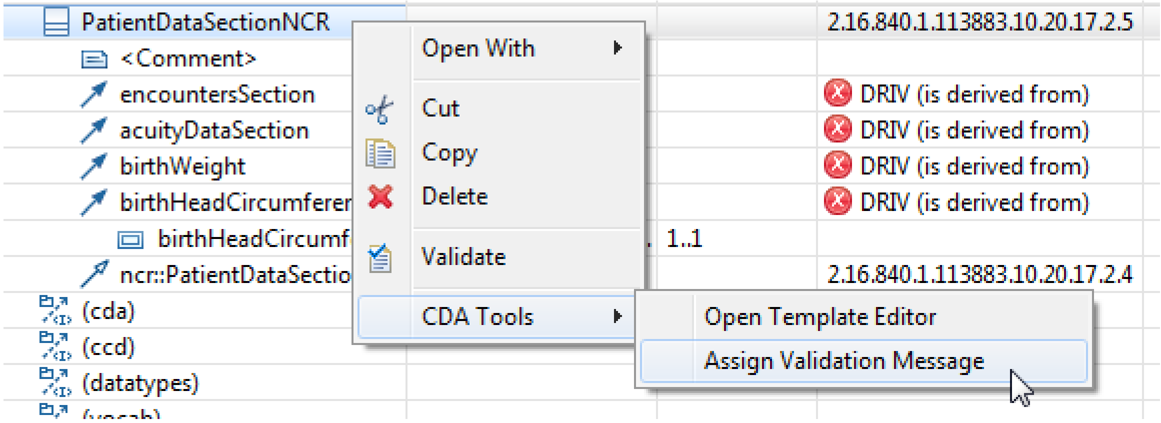
Body: self.participant->one(participant : cda::Participant2 | participant.typeCode=vocab::ParticipationType::LOC and participant.oclIsKindOf (ncr::NeonatalICULocation))

Severity and message

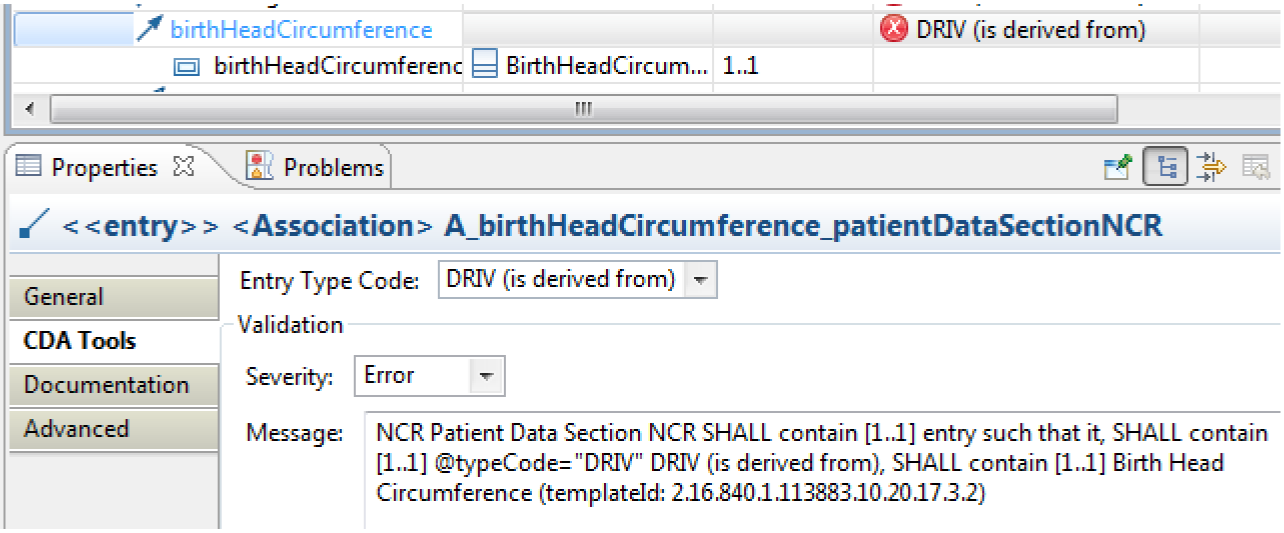
A template constraint may be assigned a severity level and a message string that describes the constraint.

Each template constraint may be assigned a severity level chosen from: Error, Warning, or Info. If not specified, Error is the default. These values correspond to typical requirement specification modes of SHALL, SHOULD, and MAY.

Each constraint also has a message that is displayed by generated Java code when CDA document instances are validated using these template definitions. Although a custom message can be entered for each constraint, it is easiest to calculate the message text from the model. The messages may be assigned as shown below.



After assigning calculated messages for this class, the association message looks like this:



<<entry>> <Association> A_birthHeadCircumference_patientDataSectionNCR

Entry Type Code: DRIV (is derived from)

Validation

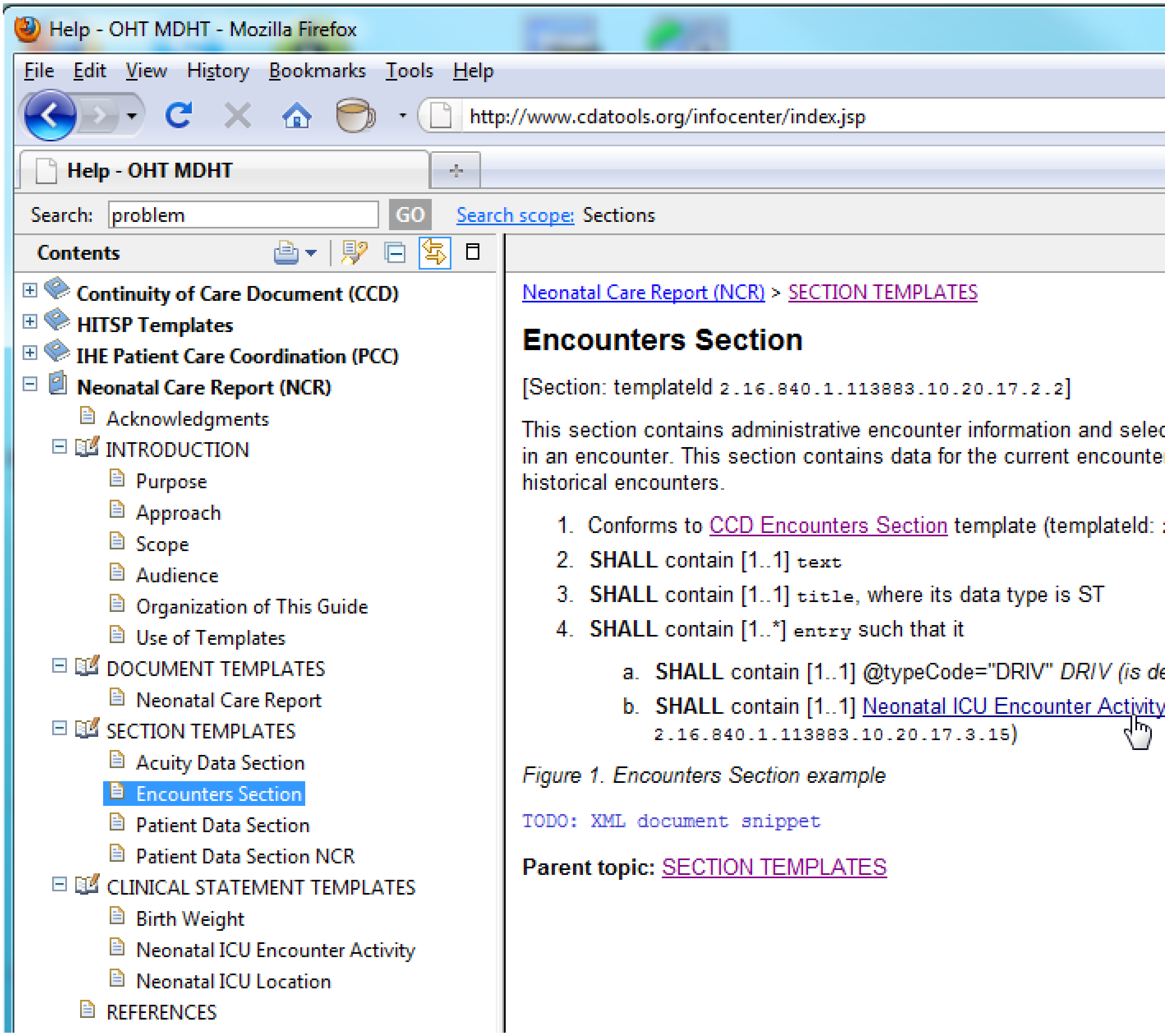
Severity: Error

Message: NCR Patient Data Section NCR SHALL contain [1..1] entry such that it, SHALL contain [1..1] @typeCode="DRIV" DRIV (is derived from), SHALL contain [1..1] Birth Head Circumference (templateId: 2.16.840.1.113883.10.20.17.3.2)

Publish an Implementation Guide

Publish an implementation guide document from a UML model containing CDA template definitions.

This is still work-in-progress, however excellent progress has been made using the DITA XML standard for technical documentation as a vehicle for publishing implementation guides. The current proof-of-concept tooling can publish guides in both PDF and the Eclipse Help format. The on-line help format is a promising new approach for publishing and sharing CDA template specifications. The following screen shot illustrates an example.


Help - OHT MDHT - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.cdatools.org/infocenter/index.jsp
Help - OHT MDHT
Search: problem
GO
Search scope: Sections
Contents

Continuity of Care Document (CCD)
HITSP Templates
IHE Patient Care Coordination (PCC)
Neonatal Care Report (NCR)
Acknowledgments
INTRODUCTION
Purpose
Approach
Scope
Audience
Organization of This Guide
Use of Templates
DOCUMENT TEMPLATES
Neonatal Care Report
SECTION TEMPLATES
Acuity Data Section
Encounters Section
Patient Data Section
Patient Data Section NCR
CLINICAL STATEMENT TEMPLATES
Birth Weight
Neonatal ICU Encounter Activity
Neonatal ICU Location
REFERENCES

Neonatal Care Report (NCR) > SECTION TEMPLATES

Encounters Section

[Section: templateId 2.16.840.1.113883.10.20.17.2.2]

This section contains administrative encounter information and selec in an encounter. This section contains data for the current encounte historical encounters.

1. Conforms to CCD Encounters Section template (templateId: :
2. SHALL contain [1..1] text
3. SHALL contain [1..1] title, where its data type is ST
4. SHALL contain [1..*] entry such that it
   a. SHALL contain [1..1] @typeCode="DRIV" *DRIV (is de*
   b. SHALL contain [1..1] Neonatal ICU Encounter Activity 2.16.840.1.113883.10.20.17.3.15)

Figure 1. Encounters Section example

TODO: XML document snippet

Parent topic: SECTION TEMPLATES

Create a new Implementation Guide

Create a project for a new implementation guide model.

NOTE: we will work on replacing this process with a Wizard in the next milestone.

The best way to create a new project is to copy the provided example project and run the `refactor.xml` Ant script to rename it. This can be accomplished by following these steps.

1. Download and disconnect the example project from the MDHT SVN repository
   a) Check out `cda/example/org.openhealthtools.mdht.uml.cda.example`. Please check out a new unmodified project, don't start with an example you have changed.
   b) Right-click on this new project and select **Team** > **Disconnect**, and select the option: 'Also delete SVN meta-information from file system'

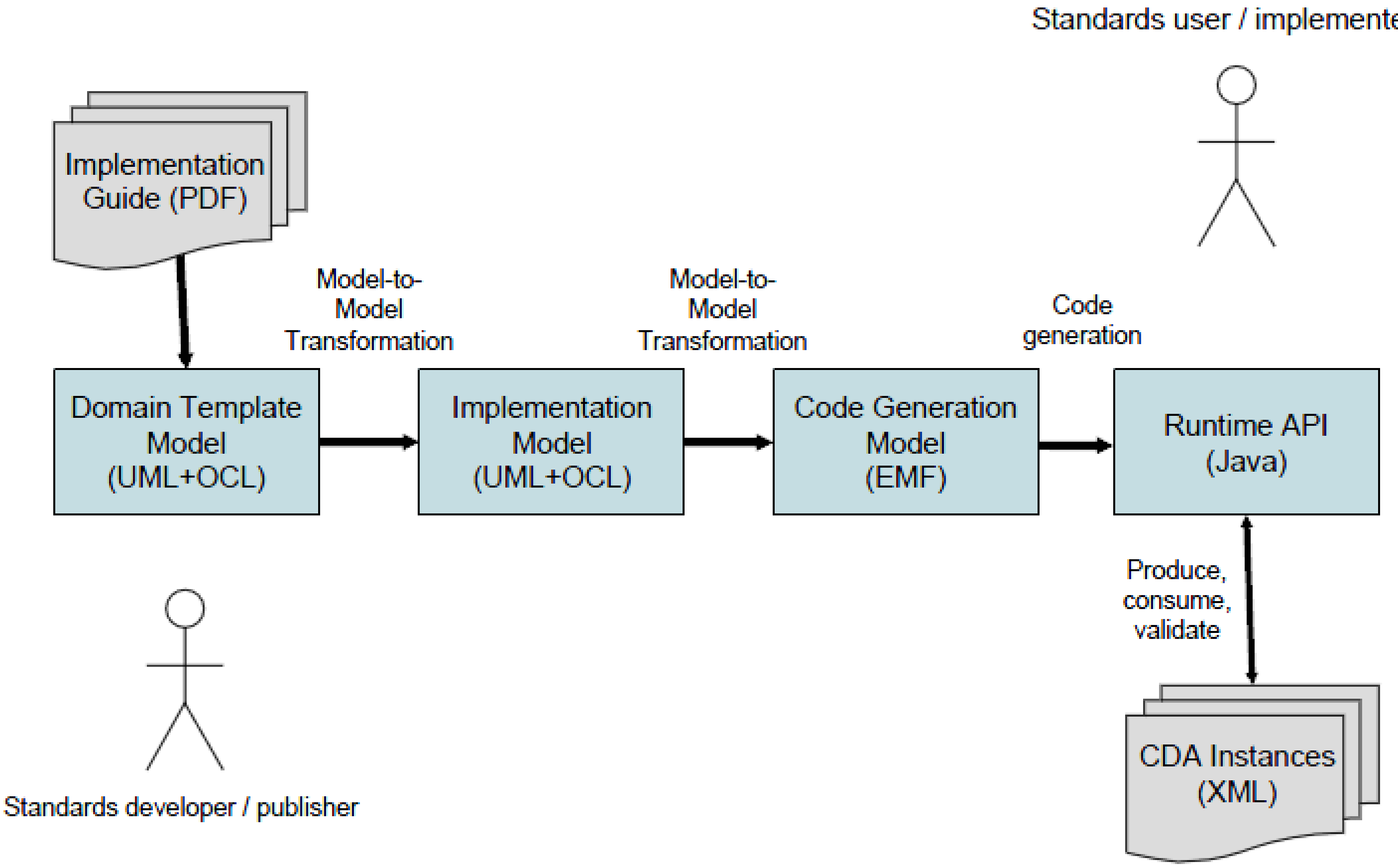
2. Rename the project. Right-click on the new project and select **Refactor** > **Rename**. For example, enter the new project name 'org.openhealthtools.mdht.uml.cda.ncr'.
3. Edit and run the `refactor.xml` script.
    a) Open the refactor.xml script in the Ant or text editor.
    b) Change the property values for the following properties: basePackage (e.g. org.openhealthtools.mdht.uml.cda), packageName (e.g. ncr), prefix (NCR), and nsURI (e.g. http://www.openhealthtools.org/mdht/uml/cda/ncr).
    c) Run the `refactor.xml` script as an Ant build. Make sure to run in the same JRE as the workpace.

Standards Implementers

The main role of the standards implementer is to generate a runtime API from the template model and write applications that leverage the generated code (e.g. EMR adapters). These applications may consiste of CDA content producers, CDA content consumers, and CDA content validators.

Model-Driven Development Process Overview

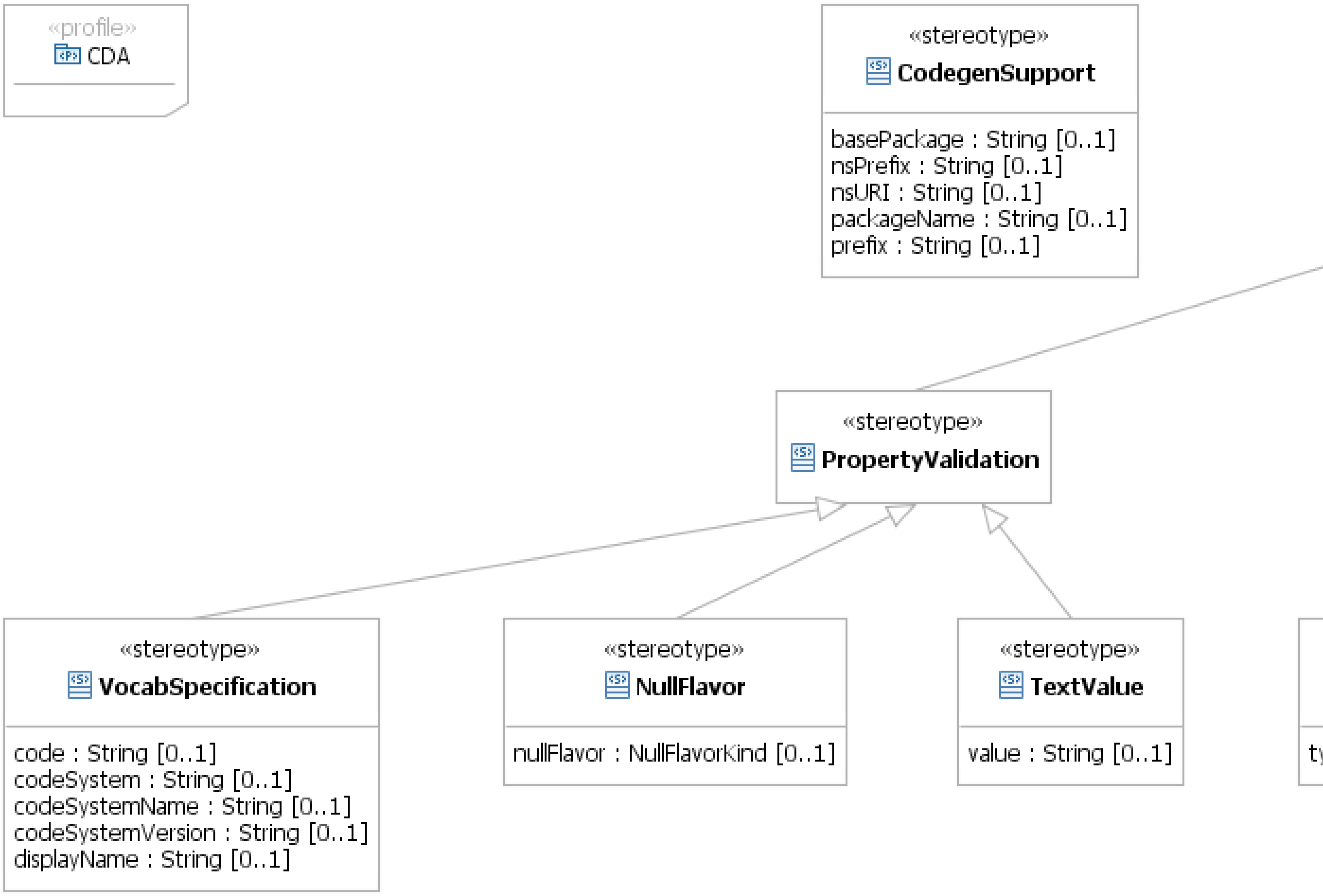
A model-driven development process is used to produce source code from UML models.


Standards user / implemente
Implementation Guide (PDF)
Model-to-Model Transformation
Model-to-Model Transformation
Code generation
Domain Template Model (UML+OCL)
Implementation Model (UML+OCL)
Code Generation Model (EMF)
Runtime API (Java)
Produce, consume, validate
CDA Instances (XML)
Standards developer / publisher

The standards developer is responsible for creating the *domain template model*. This model is input to the model-to-model transformation process. The template model is ultimately converted into an *implementation model*. The implementation model has been restructured and prepared for code generation. We use the Eclipse Modeling Framework (EMF) to generate Java source code. Once the implementation model has been created, it is imported into EMF. The final step of the process is to produce source code from the EMF model.

UML Profile for CDA

A UML profile is a generic extension mechanism supported by most UML modeling tools used to refine or customize a UML model. Profiles are applied to UML models and contain stereotypes. Stereotypes are applied to individual model elements and contain properties. Properties allow the modeler to specify additional information or metadata directly in the model. Metadata specified in stereotype properties is used in the model-to-model transformation to generate OCL constraints and attach various annotations that are used downstream during code generation and at runtime.



CDA Template Stereotype

The CDA Template stereotype is applied to the UML Class model element and is used to capture information about the template including the template id.

Code Generation Support Stereotype

The code generation support stereotype is applied to the UML Package model element and is used to specify parameters used during EMF code generation.

Vocabulary Specification Stereotype

The Vocabulary Specification Stereotype is applied to the UML Property model element and is used to capture information metadata for coded attributes in the template

Entry Stereotype

The Entry Stereotype is applied to the UML Association model element and is used to indicate that a Section has a required Entry or a required clinical statement. There stereotype is also used to specify an optional type code.

Entry Relationship Stereotype

The Entry Relationship Stereotype is applied to the UML Association model element and is used to indicate that there is a constraint between two clinical statements.

Null Flavor Stereotype

The Null Flavor Stereotype is applied to the UML Property model element and is used to capture information about those attributes who are fixed or defaulted to a specific NullFlavor value

Text Value Stereotype

The Text Value Stereotype is applied to the UML Property model element and is used for attributes that are constrained to a default or fixed text value (e.g. Section.title)

Model-to-Model Transformation

Input: domain template model (UML)

Process UML model elements to produce:

- OCL constraints for validating template instance
- Annotations for instance population and validation support
- Additional directives to EMF for model import / code generation

Output: implementation model (UML)

Invoke the Model-to-Model Transformation via Ant

An Ant build script is used to invoke the model-to-model transformation

1. Right-click on build.xml and select Run As > Ant Build...
2. Click on the JRE tab and make sure "Run in the same JRE as the workspace" is selected
3. Click the "Run" button
   The console will display log messages as the transformation executes

Generated OCL Constraints

One of the primary purposes for creating a template model is to express constraints on the CDA model in an intuitive, structural way. The model-to-model transformation uses this logical representation to generate OCL constraints reducing the burdern on the developer to hand code OCL expressions.

Generated Template Constraint

A template ID constraint is generated from information specified in the CDA Template stereotype as applied to classes in the template model. It is used to check whether all of the appropriate template IDs have been added to the instance

In this example, we check for the template ID associated with ncr::BirthWeight

```
self.hasTemplateId('2.16.840.1.113883.10.20.17.3.1')
```

A template relationship constraint is generated from directed associations (and information from stereotypes applied to those assciations) in the template model. It is used to enforce the "has-a" or containtment relationship between templates

In this example, the constraint is used to enforce the relationship between the ncr::EncountersSection and the ncr::NeonatalICUEncounterActivity. Also, note that stereotype properties where used to specify the value of typeCode used in the generated constraint.

```
self.entry->exists(entry : cda::Entry |
 entry.encounter.oclIsKindOf(ncr::NeonatalICUEncounterActivity) and
 entry.typeCode = vocab::x_ActRelationshipEntry::DRIV)
```

Generated Property Constraints

Property constraints are those constraints that involve attributes with fixed or default values, attributes that have a more restrictive cardinality than in the base CDA model, and attributes that have a more restrictive type.

An example of a constraint generated from template model that uses UML property redefinition to specify a more restrictive type. The ANY type of the value attribute has been restricted to PQ (Physical Quantity)

```
self.value->forAll(element | element.oclIsTypeOf(datatypes::PQ))
```

Generated Property Constraints

Vocabulary constraints are those constraints that involve coded attributes.

An example of a constraint generated from template model using information specified in the vocab specification stereotype. The code system and code value have been fixed in this example taken form Neonatal Care Report (NCR)

```
not self.code.oclIsUndefined() and self.code.oclIsKindOf(datatypes::CD) and
 let value : datatypes::CD = self.code.oclAsType(datatypes::CD) in (value.code
 = '47340003' and value.codeSystem = '2.16.840.1.113883.6.96')
```

Runtime API

The Runtime API is comprised of:

- CDA, data types, and vocabulary runtime APIs with helper methods to assist in writing constraints
- Java classes/packages generated for template model
- Utility class with convenience methods to save, load, and validate documents

Additional UML operations specified in the template model are carried through to the Java source code and can be implemented directly in the model using OCL or by specifying the method body in the generated code. This gives the modeler the ability to add convenience to the runtime API at design-time. Annotations generated from the template model are used to populate the runtime instance for default/fixed values. This reduces the number of method calls required to build a document.

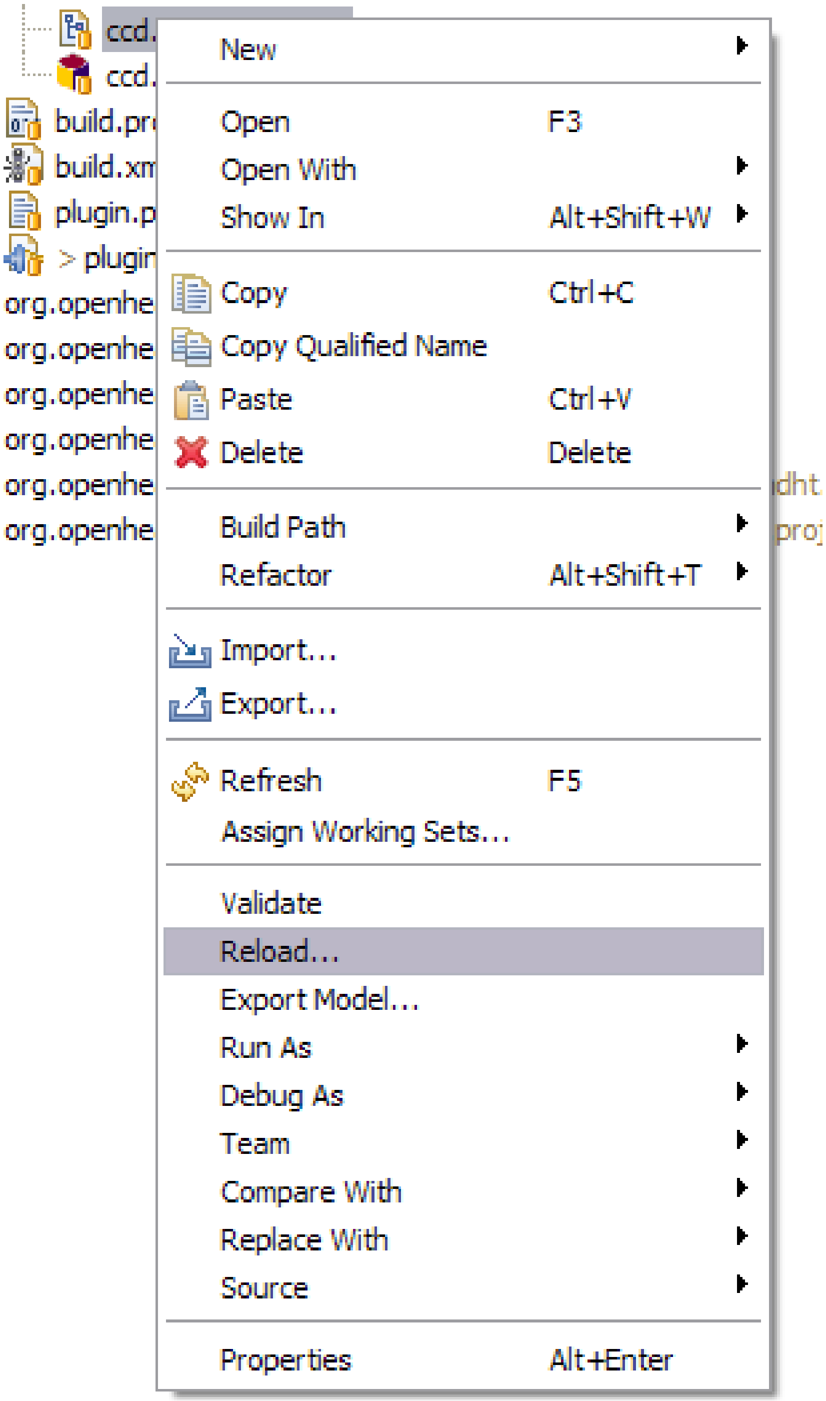
Path expression support (experimental) has also been added to construct parts of a document efficiently. This is mainly intended to be a transitional API for those familiar with the XML structure of a CDA document.

```
CDAUtil.create("/component/structuredBody/component/section",
 CCDPackage.Literals.PROBLEM_SECTION);
```

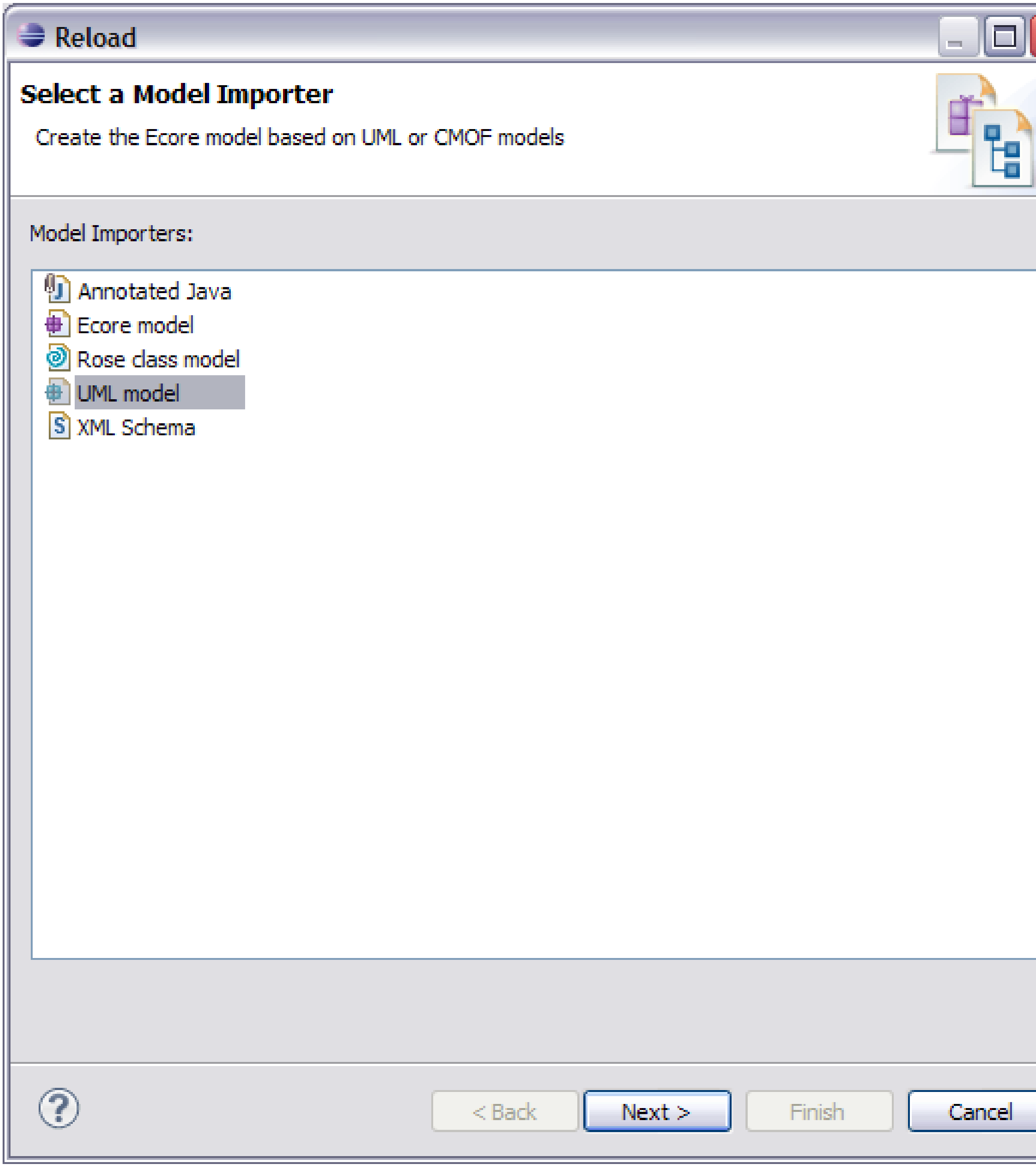
Code Generation

The implementation model is imported from UML into an Eclipse Modeling Framework (EMF) model using standard model import functionality. An EMF model is the aggregation of two different models. The first model is an Ecore model which contains annotations with constraints from the original UML model. The second model is the generator model. The generator model contains information on how to generate Java code from the Ecore model. It essentially "decorates" the Ecore model with additional information. The Generator model points to a set of custom code generation templates to generate all Java classes/packages. The code generation templates were augmented to use annotations added during the model-to-model transformation step.

1. **1.** Reload GenModel Reload GenModel context menu "Reload..."



Reload Model Importer:



Reload

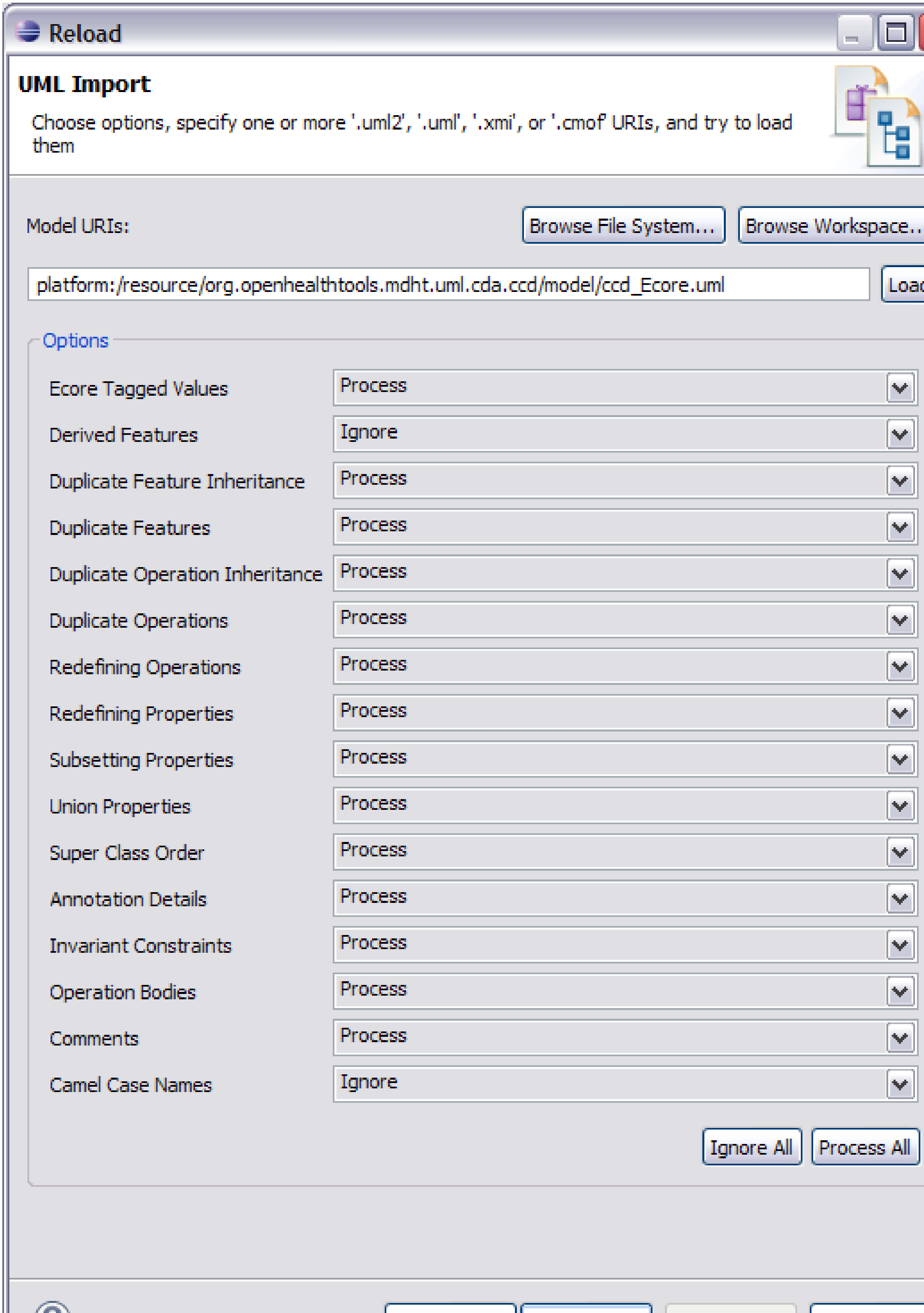
Select a Model Importer

Create the Ecore model based on UML or CMOF models

Model Importers:

- Annotated Java
- Ecore model
- Rose class model
- UML model
- XML Schema

Reload UML Import



UML Import

Choose options, specify one or more '.uml2', '.uml', '.xmi', or '.cmof' URIs, and try to load them

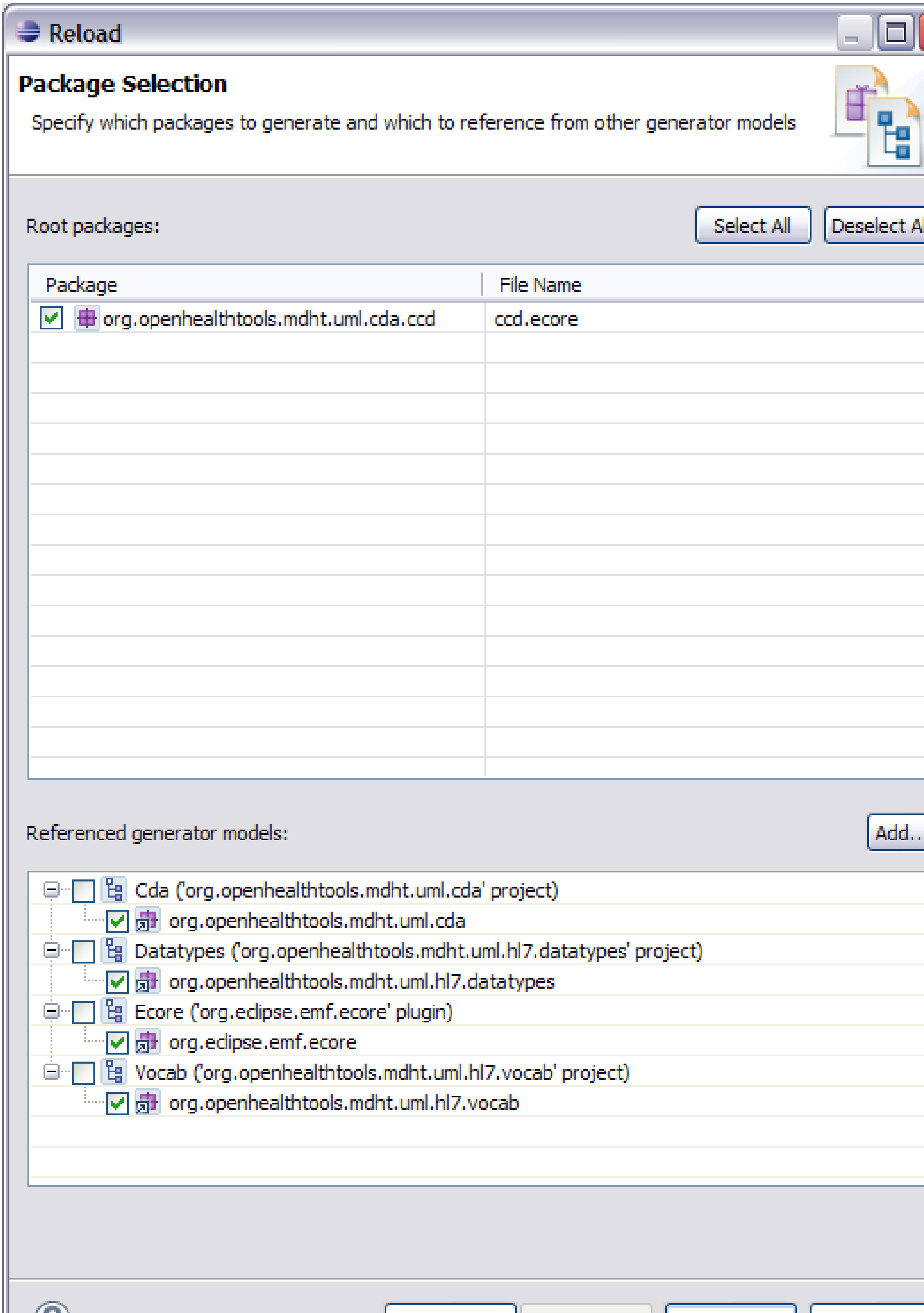
Model URIs:

platform:/resource/org.openhealthtools.mdht.uml.cda.ccd/model/ccd_Ecore.uml

Options

Ecore Tagged Values	Process
Derived Features	Ignore
Duplicate Feature Inheritance	Process
Duplicate Features	Process
Duplicate Operation Inheritance	Process
Duplicate Operations	Process
Redefining Operations	Process
Redefining Properties	Process
Subsetting Properties	Process
Union Properties	Process
Super Class Order	Process
Annotation Details	Process
Invariant Constraints	Process
Operation Bodies	Process
Comments	Process
Camel Case Names	Ignore

Reload Package Selection:



Package Selection

Specify which packages to generate and which to reference from other generator models

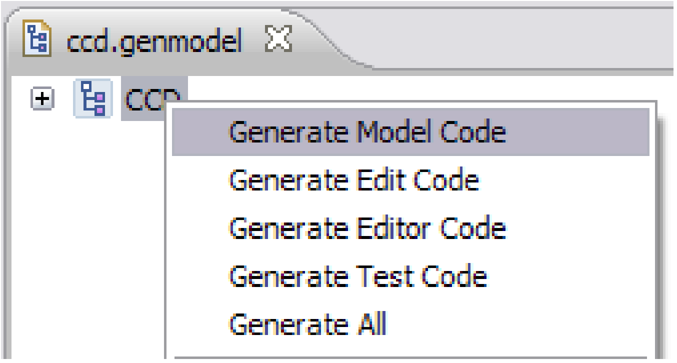
Root packages:

Package	File Name
☑ org.openhealthtools.mdht.uml.cda.ccd	ccd.ecore

Referenced generator models:

- □ Cda ('org.openhealthtools.mdht.uml.cda' project)
  - ☑ org.openhealthtools.mdht.uml.cda
- □ Datatypes ('org.openhealthtools.mdht.uml.hl7.datatypes' project)
  - ☑ org.openhealthtools.mdht.uml.hl7.datatypes
- □ Ecore ('org.eclipse.emf.ecore' plugin)
  - ☑ org.eclipse.emf.ecore
- □ Vocab ('org.openhealthtools.mdht.uml.hl7.vocab' project)
  - ☑ org.openhealthtools.mdht.uml.hl7.vocab

2. Generate Model Code Generate Model Code Context Menu:



Producing an XML instance (serialization)

Client code uses the runtime API to build instance of template model. All template models get serialized according to the underlying CDA model. EMF uses annotations at runtime to properly serialize Ecore model instance to XML document. The EMF serializer has been customized to omit xsi:type information and put all serialized elements into one namespace (as per CDA XML schema). CDA schema defines templateId element to identify places in the document where templates are used

```
NeonatalCareReport clinicalDocument =
 NCRFactory.eINSTANCE.createNeonatalCareReport().init();
PatientDataSectionNCR patientData =
 NCRFactory.eINSTANCE.createPatientDataSectionNCR().init();
EncountersSection encounters =
 NCRFactory.eINSTANCE.createEncountersSection().init();

clinicalDocument.addSection(patientData);
patientData.addSection(encounters);

NeonatalICUEncounterActivity encounter =
 NCRFactory.eINSTANCE.createNeonatalICUEncounterActivity().init();
encounters.addEncounter(encounter);

CDAUtil.save(clinicalDocument, System.out);
```

Consuming an XML instance (deserialization)

- XML instance loaded from input stream into DOM
- XML instance contains template ids
- DOM handed to EMF deserializer
- Loading mechanism is intercepted to inject type information
  - DOM gives us look ahead access to get template id
  - Template id is used to look up Ecore package/class information from CDA template registry (uses Eclipse plugin extension registry)
  - xsi:type information is dynamically added to DOM element
- Augmented DOM element is then handed back to EMF to construct Ecore model instance of the correct type according to template model

```
ClinicalDocument clinicalDocument = CDAUtil.load(new
 FileInputStream("resources/SampleCDADocument.xml"));
System.out.println(clinicalDocument);
```

Validation

Constraints specified in the form of standard UML constructs such as cardinality are automatically validated as part of the EMF framework

OCL constraints specified by the modeler or generated during the model-to-model transformation are carried through to the EMF model and Java source code

EMF validation mechanism

- Uses OCL interpreter at run time to validate these constraints
- Each Ecore model gets a separate validator (e.g. datatypes, CDA, CCD, etc.)
- Validators work together to validate Ecore model instance

Output of validation is a diagnostic tree

- Validation severity and message specified in the model are used at runtime
- Diagnostic tree can be processed using CDA utility class

```
boolean valid = CDAUtil.validate(clinicalDocument, new
 BasicValidationHandler() {
 @Override
 public void handleError(Diagnostic diagnostic) {
  System.out.println("ERROR: " + diagnostic.getMessage());
 }
 @Override
 public void handleWarning(Diagnostic diagnostic) {
  System.out.println("WARNING: " + diagnostic.getMessage());
 }
 @Override
 public void handleInfo(Diagnostic diagnostic) {
  System.out.println("INFO: " + diagnostic.getMessage());
 }
});

if (valid) {
 System.out.println("Document is valid");
} else {
 System.out.println("Document is invalid");
}
```