Model-Driven Health Tools (MDHT) CDA Tools Overview

http://mdht.projects.openhealthtools.org

John T.E. Timm, IBM Research
David A. Carlson, Contractor to the Veterans Health Administration

Motivation

- Healthcare interoperability standards for clinical documents have:
  - A steep learning curve due to lengthy and complex specifications
  - Lack of tooling to support template model design and implementation
  - No formal methodology/best practices for developing templates and implementation guides
- Current implementation approaches are inadequate

OPEN HEALTH TOOLS

Current Implementation Approaches

- Low-level XML processing technologies
  - e.g. SAX, DOM, XPath
  - API is generic, not domain-specific
  - Instance validation through XML Schema and Schematron
  - Requires intimate knowledge of CDA XML structure (and lots of code)
- XML binding techniques
  - e.g. JAXB, EMF-XSD, XMLBeans
  - Not suited for code generation with very complex schemas
  - If code generation succeeds, API is oriented towards base CDA XML structure
- RIM-based approaches
  - Take HL7 artifacts (e.g. MIF) as input
  - RIM-based object model is taken together with instance data and serialized/deserialized to/from XML
  - API is oriented towards abstract RIM structure

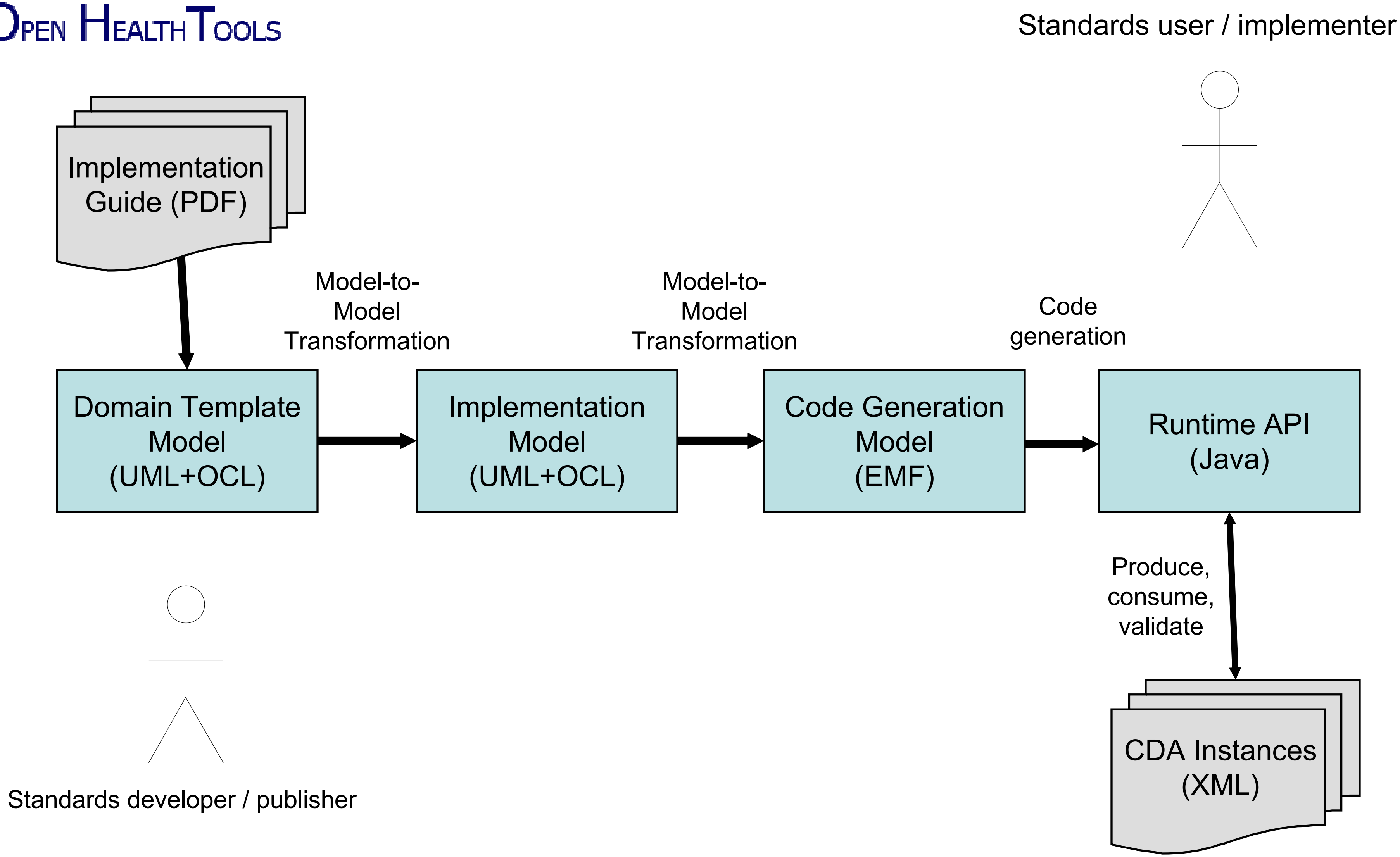
CDA Tools Objectives

- Accelerate and lower cost of adopting CDAr2 standard
- Provide standard OOAD-based methodology/tooling for modeling CDA templates
- 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

Open Health Tools

Users

- 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 / implementer
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

UML Profile for CDA

- What is a UML Profile?
  - Generic mechanism supported by most modeling tools to refine/customize a UML model
  - Profiles are applied to models and contain stereotypes
  - Stereotypes are applied to model elements and contain properties
  - Allows the modeler to specify additional information (metadata) directly in the template model
- Metadata is used in the model-to-model transformation to:
  - Generate OCL constraints that correspond to conformance rules in IG
  - Attach various annotations used downstream during code generation and at runtime

CDA Template Modeling

- Constraints are modeled using UML properties and directed associations in conjunction with stereotypes from the CDA Profile
- Types of constraints:
  - Property constraints
    - Used to specify fixed or default value
    - Used to restrict cardinality and/or data type
  - Vocabulary constraints on coded attributes
    - Used to restrict code, codeSystem, etc.
- Template-related constraints
  - Template id
  - Template relationships (is-a, has-a)

Open Health Tools

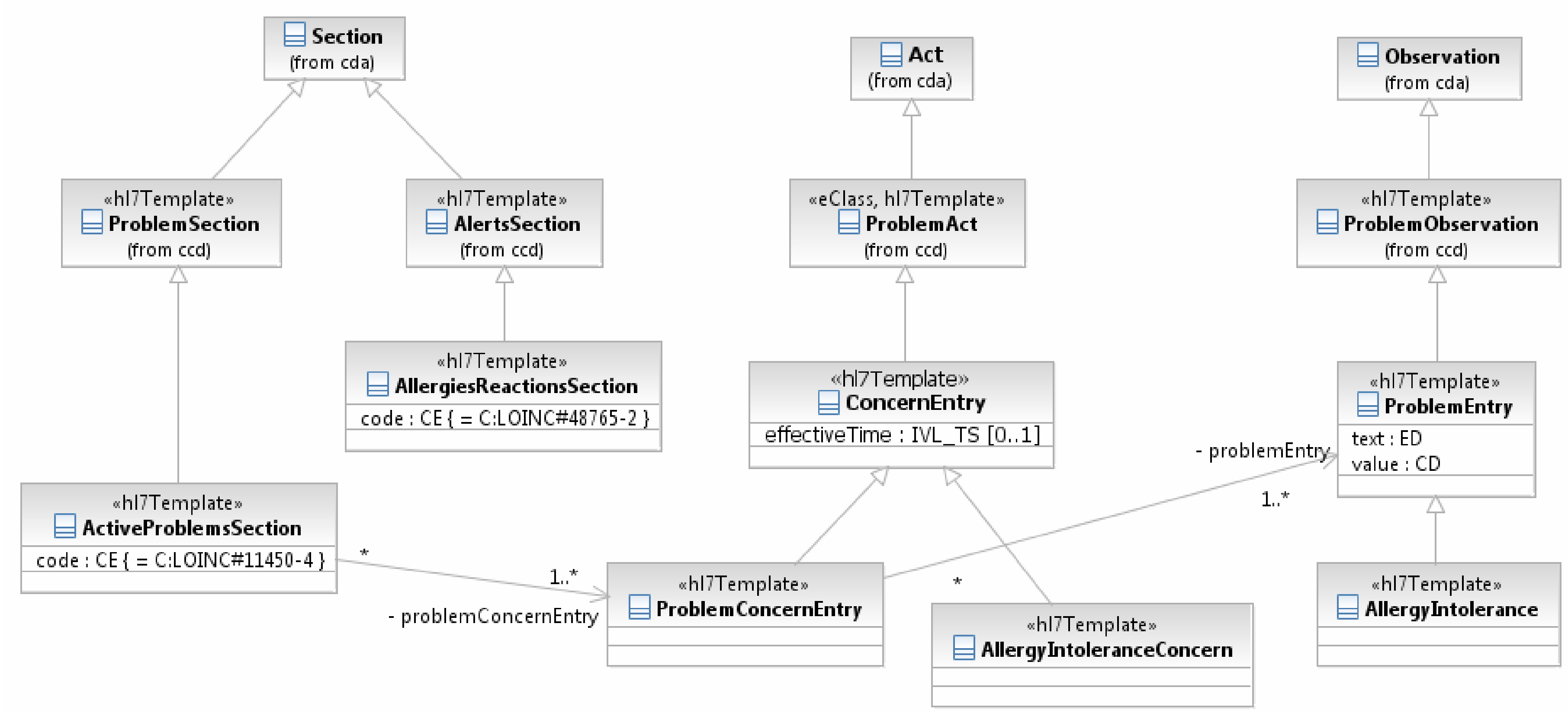
CDA Template Modeling - relationships

- **Generalization/inheritance/"is-a"**
  - **Modeled using standard UML generalization**
  - **Examples:**
    - **hitsp::Condition is a ihe::ProblemConcernEntry**
    - **ihe::ProblemConcernEntry is a ihe::ConcernEntry**
    - **ihe::ConcernEntry is a ccd::ProblemAct**
    - **ccd::ProblemAct is a cda::Act**
- **Association/containment/"has-a"**
  - **Modeled using UML directed association**
  - **Examples:**
    - **ccd::ContinuityOfCareDocument has a ccd::ProblemSection**
    - **ccd::ProblemSection has a ccd::ProblemAct**
    - **ccd::ProblemAct has a ccd::ProblemObservation**

CDA Template Models

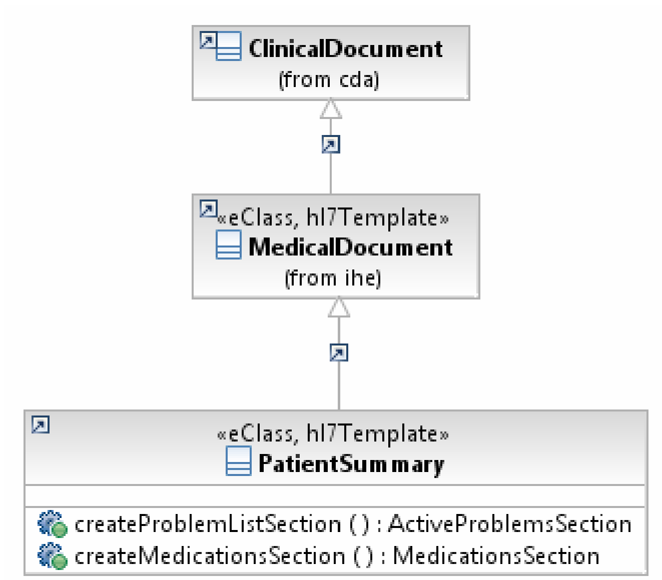
- We are currently working on implementations of the following CDA-based document types:
  - Continuity of Care Document (CCD)
  - IHE Content Profiles
  - HITSP C32 Patient Summary
  - Public Health Case Report (PHCR)
  - IHE Lab Report Document
  - HITSP C74 Personal Health Monitoring Report
  - Essential Hypertension

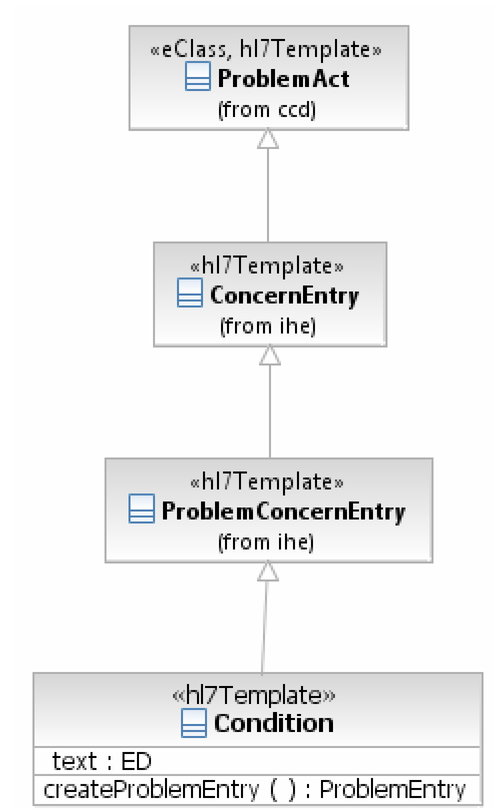
IHE Template Model (subset, work-in-progress)



HITSP Template Model (C32 and C83)

•





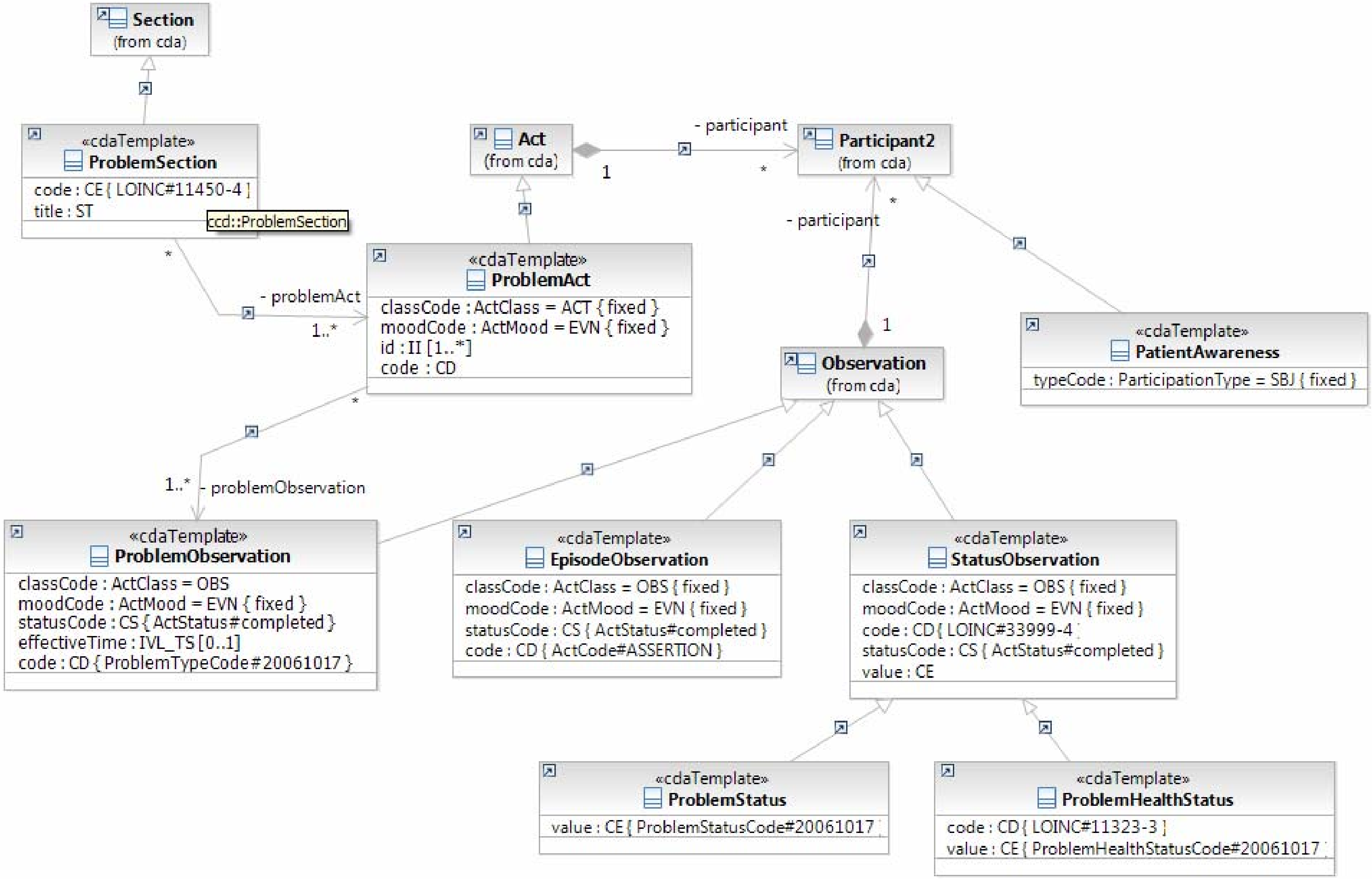
CCD Example: Conformance Rule

- **CONF-140**: CCD **SHOULD** contain exactly one and **SHALL NOT** contain more than one Problem section (templateId 2.16.840.1.113883.10.20.1.11). The Problem section **SHALL** contain a narrative block, and **SHOULD** contain clinical statements. Clinical statements **SHOULD** include one or more problem acts (templateId 2.16.840.1.113883.10.20.1.27). A problem act **SHOULD** include one or more problem observations (templateId 2.16.840.1.113883.10.20.1.28).

CCD Example: Modeling the Conformance Rule

1. **Create classes that extend the CDA model (one for each template):**
   a. ContinuityOfCareDocument (extends cda::ClinicalDocument)
   b. ProblemSection (extends cda::Section)
   c. ProblemAct (extends cda::Act)
   d. ProblemObservation (extends cda::Observation)
2. **Apply stereotypes from CDA profile to classes:**
   a. <<cdaTemplate>> stereotype is applied to all classes
   b. templateId stereotype property value is specified
3. **Create directed associations between classes:**
   a. ContinuityOfCareDocument -> ProblemSection
   b. ProblemSection -> ProblemAct
   c. ProblemAct -> ProblemObservation
   d. Multiplicity (upper bound) of the association may be specified to indicate exactly one or more than one
4. **Apply stereotypes to associations:**
   a. <<associationValidation>> stereotype is applied to all associations
   b. severity (ERROR, WARNING, INFO) and message property values are specified

CDA Template Model for CCD Problem Section



Model-to-Model Transformation

- **Input**: domain template model (UML)
- Process UML model elements and 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)

Generated OCL constraints

```
ContinuityOfCareDocument_templateId:
  self.hasTemplateId('2.16.840.1.113883.10.20.1')

ContinuityOfCareDocument_problemSection: self.getSection()->one(sect :
  cda::Section | sect.oclIsKindOf(ccd::ProblemSection))

ProblemSection_templateId:
  self.hasTemplateId('2.16.840.1.113883.10.20.1.11')

ProblemSection_problemAct: self.getAct()->exists(act : cda::Act |
  act.oclIsKindOf(ccd::ProblemAct))

ProblemAct_templateId:
  self.hasTemplateId('2.16.840.1.113883.10.20.1.27')

ProblemAct_problemObservation: self.getObservation(obs :
  cda::Observation | obs.oclIsKindOf(ccd::ProblemObservation))

ProblemObservation_templateId:
  self.hasTemplateId('2.16.840.1.113883.10.20.1.28')
```

Code generation

- Implementation model is imported into Eclipse Modeling Framework (EMF)
  - Ecore model – contains annotations with constraints from original UML model
  - Generator model – contains information on how to generate Java code from Ecore model
- Generator model with customized code generation templates used to generate Java classes/packages
- Code generation templates use annotations added in model-to-model transformation

Runtime API

- Comprised of:
  - CDA, data types, and vocabulary runtime APIs
  - Java classes/packages generated for template model
  - Additional utility classes
- Convenience methods added to CDA implementation model to assist in building constraints and constructing 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
  - By specifying the method body in the generated code
  - Gives the modeler the ability to add convenience into runtime API at design-time
- Annotations generated from template model used to populate runtime instance for default/fixed values
  - Reduces number of method calls required to build document
- Path Expression Support
  - Ability to create/query parts of the document
  - Transitional API for XML developers

Client Code for HITSP C32 Patient Summary

```
PatientSummary doc = HitspFactory.eINSTANCE.createPatientSummary().init();
II id = DatatypesFactory.eINSTANCE.createII("2.16.840.1.113883.3.72",
                  "CCD_HITSP_C32v2.4_16SectionsWithEntries_Rev6_Notes");
doc.setId(id);

ActiveProblemsSection problemList = doc.createProblemListSection();

Condition condition = HitspFactory.eINSTANCE.createCondition().init();
problemList.addAct(condition);

ProblemObservation obs =
    CCDFactory.eINSTANCE.createProblemObservation().init();
condition.addObservation(obs);

ProblemHealthStatus healthStatus =
    CCDFactory.eINSTANCE.createProblemHealthStatus().init();
obs.addObservation(healthStatus);
CE healthStatusValue = DatatypesFactory.eINSTANCE.createCE("xyz",
    "2.16.840.1.113883.1.11.20.12", "ProblemHealthStatusCode", null);
healthStatus.getValue().add(healthStatusValue);
```

Producing an XML instance (serialization)

- Client code uses 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
- 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

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 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

OPEN HEALTH TOOLS

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

Future Work

- Collaboration with other OHT Projects
- Publish Implementation Guide from template model
- Integration with Template Registry
- Integration with Terminology Services
- Extend approach to support all RIM derivations and serializations
- Integration with Semantic Web Technologies
- Model-driven design of services