CEM to CIMI Conversion White Paper

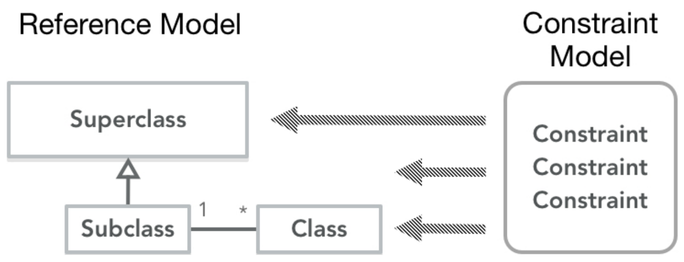
Table of Contents

1. Introduction

This paper will describe our most recent approach to the transformation of detailed clinical models from the Clinical Element Model (CEM) standard to the Clinical Information Modeling Initiative (CIMI) version 2 standard. CIMI is currently a working group within Health Level Seven (HL7). The CEM models used as the source for the translation will be provided by Intermountain Healthcare.

In detailed clinical modeling there is a general approach shown in Figure 1, "Reference Model Constraint Paradigm", that most standards have taken where there is a reference model which is then constrained with constraint models. In Figure 2, "Detailed Clinical Modeling Standards", the reference model and constraint model of various standards are shown. Additional information shown is the syntax used to describe the reference or constraint model, and if the standard is shaded grey, this symbolizes that clinical content is modeled in this part of the standard.

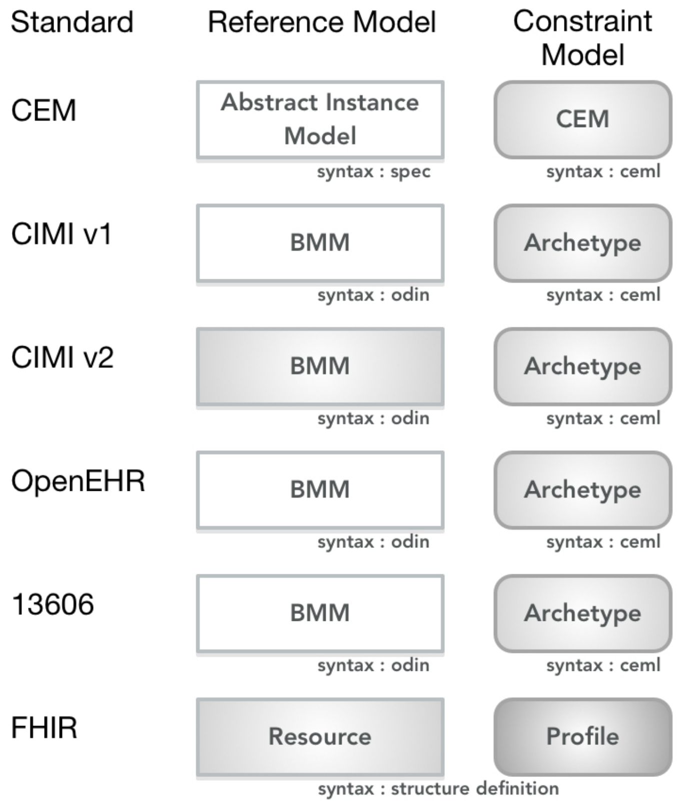
Figure 1. Reference Model Constraint Paradigm



The reference model for the CEM standard is called the Abstract Instance Model. It is not formally defined with a syntax, but instead is defined in a specification which can then be implemented in various implementations such as java or xml. This reference model is very small and has no defined clinical content. Instead, clinical content is defined only in the constraint models of the CEM standard which are written in Clinical Element Modeling Language (CEML).

The reference model for the CIMI v2 standard describes clinical content in both the reference and constraint models using the Archetype formalism developed by OpenEHR. The reference model is described in the Basic Meta-Model (BMM) using Object Data Instance Notation (ODIN) and the constraint models are described as Archetypes with Archetype Definition Language (ADL).

Figure 2. Detailed Clinical Modeling Standards



2. Clinical Element Model

The Clinical Element reference model, as seen in Figure 3, “Clinical Element Abstract Instance Model”, is a recursive model. In the figure, the stacks of grey disks within 'items', 'mods', and 'quals' are meant to represent a collection of referenced Clinical Elements. These Clinical Elements could then reference other Clinical Elements, and so on, leading to a recursive tree with infinite variety.

An instance of a Clinical Element could look like Figure 4, “Clinical Element Instance”. In reality, one could put any data into these slots in the reference model, and the reference model has no inherent mechanism to validate whether this is good data or nonsense. This is where the Clinical Element Constraint Model comes into play. These constraint models are called CEMs and are written in CEML. A possible CEML example for the previous instance is shown in Example 1, “CEML for Systolic Blood Pressure”. If we compare the instance to the CEML, we can see the instance contains a field called type with a value of 'SystolicBloodPressure'. This value is the name of the CEM that will be used to validate the instance, which is the CEML example given. The CEML then states the instances must contain a key with a code of 'SystolicBloodPressure_CODE', must have a datatype of type 'Quantity', and has a constraint that states the Quantity must have a unit of 'MillimetersOfMercury_CODE'. Also stated in the CEML is that an allowable qualifier can reference the 'BodyPosition' CEM, but that the valueset within 'BodyPosition' has been constrained to 'SBP_BodyPosition_VALUESET_CODE'.

Figure 3. Clinical Element Abstract Instance Model

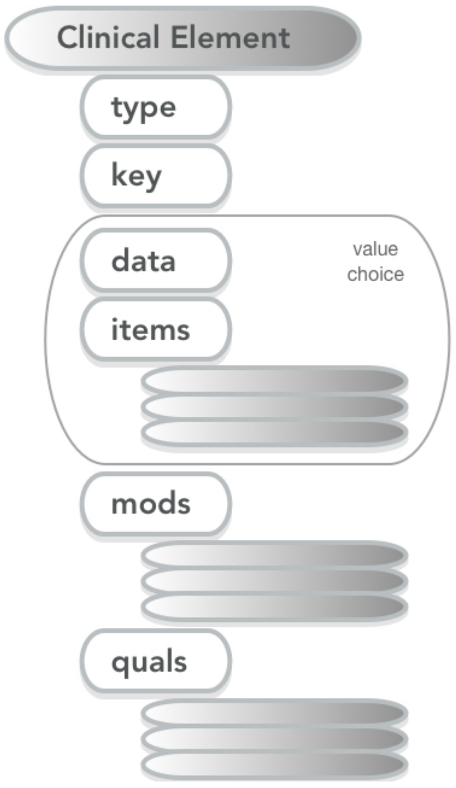
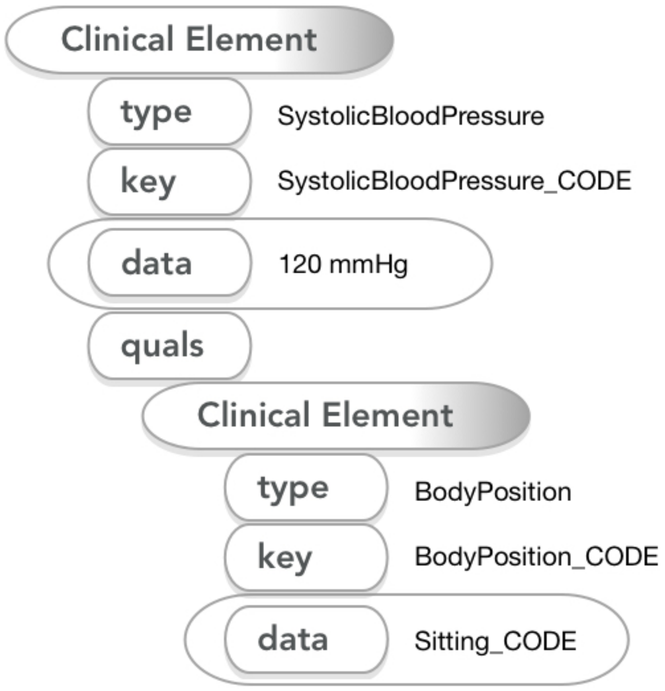


Figure 4. Clinical Element Instance



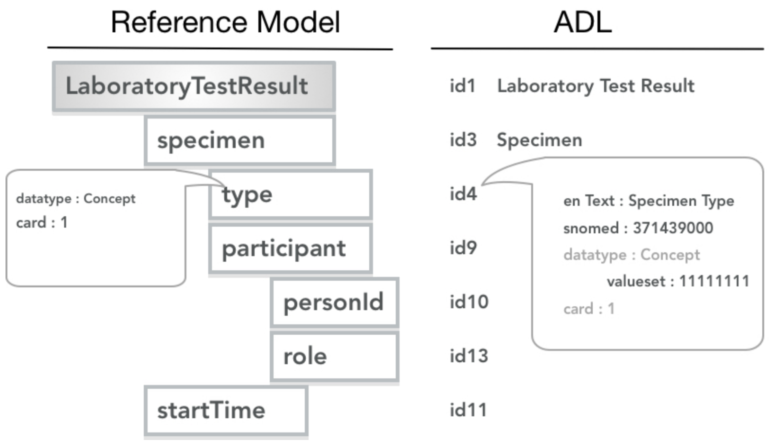
Example 1. CEML for Systolic Blood Pressure

```
cem SystolicBloodPressure
    key     SystolicBloodPressure_CODE
    data    Quantity
    qual    BodyPosition
       id   bodyPosition
       card 0-1
    constraint data.quantity.unit_code
       value   MillimetersOfMercury_CODE
    constraint bodyPosition.data.codeableConcept.valueset
       value   SBP_BodyPosition_VALUESET_CODE
```

3. CIMI Model

The CIMI Reference Model, as seen in Figure 5, "CIMI model with Archetype Constraint", contains clinical content and is very similar to a UML model, but is declared in a BMM. In fact, CIMI modelers are currently modeling in UML and then generating the BMM from the UML model. On the left hand side of the figure, is an example of a possible standard lab observation model described with the BMM. In the BMM, models are declared along with their named properties. The properties can either reference other defined models or datatypes, and a cardinality can be assigned.

Figure 5. CIMI model with Archetype Constraint

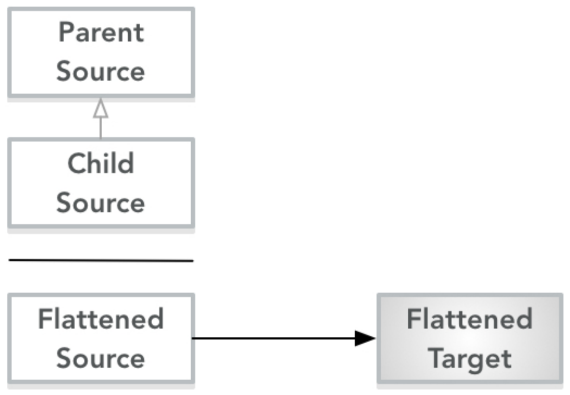


The CIMI BMM is then constrained using Archetypes defined in ADL to bind standard terminology such as 'about' codes and allowable valuesets for coded fields. Also bound can be descriptive text in various languages. And finally, the cardinality can be further specified. In the figure, an illustrative example of an Archetype is given on the right which constrains the example BMM model on the left. The 'type' node from the reference model is bound to an id of 'id4' and then various constraints are bound to 'id4'.

4. Transformation

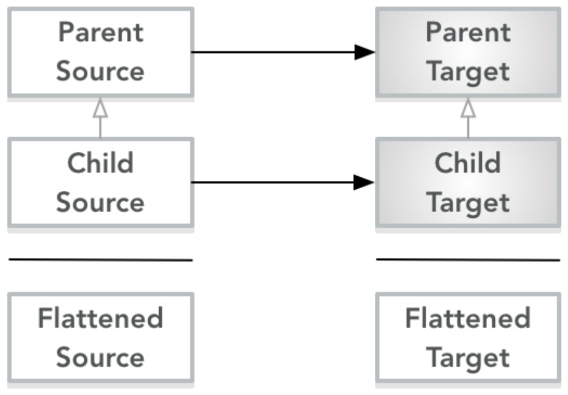
The transformation from CEMs to CIMI models with respect to inheritance hierarchy could take one of two approaches. The first approach, as seen in Figure 6, "Asymmetric Hierarchy Transformation", would be to compile and collapse the inheritance hierarchy of the CEMs and then transform the resulting collapsed model. In this approach, the original inheritance hierarchy is lost in the resulting target CIMI models.

Figure 6. Asymmetric Hierarchy Transformation



In the second approach, as seen in Figure 7, “Symmetric Hierarchy Transformation”, each model in the hierarchy is transformed to a parallel model in the target hierarchy, which results in a preservation of the original inheritance hierarchy. This second approach is the approach we are taking in the transformation of CEMs to CIMI models for two reasons. First, it allows us the possibility to reverse the transformation process for parts of the transformation where needed. Second, it will allow the generated CIMI models to fit within the hierarchy as if CIMI had modeled them rather than simply creating large CIMI compliant models that stand alone.

Figure 7. Symmetric Hierarchy Transformation

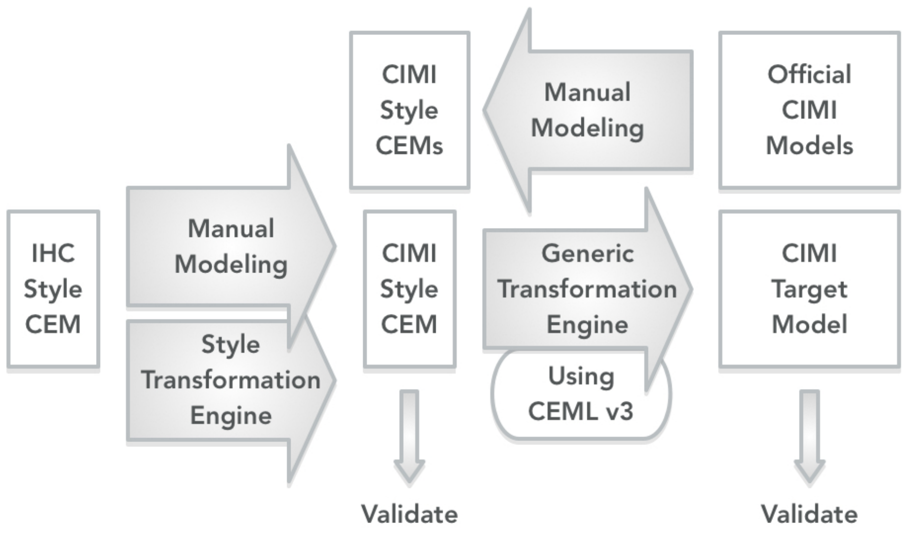


4.1. Transformation Strategy

The goal is to transform the entire library of Intermountain CEM models into an improved CEML syntax we are calling CEML v3 and then to the final target being official CIMI ADL/BMM. In our previous attempts at transformation we discovered some limitations in CEML to represent some of the complexities of the BMM/ADL models being used in CIMI. These limitations in CEML v2 include but are not limited to documentation, datatype naming, and array slicing. After this process is completed, CEML v2 models will be retired, and subsequent modeling with continue using CEML v3.

The overall strategy for transformation of Intermountain CEMs to CIMI models is shown in Figure 8, “Transformation Strategy”. The first step starts in the upper right hand corner of the diagram with the existing official CIMI models. These CIMI models will be manually modeled as CEMs in the improved CEML syntax of CEML v3.

Figure 8. Transformation Strategy

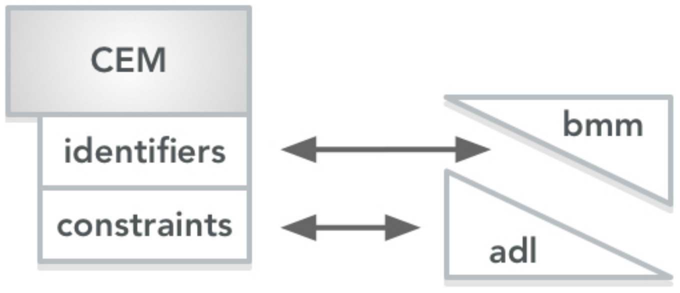


The next step in the transformation starts on the left hand side of the diagram, where Intermountain Healthcare style CEMs are transformed to CIMI style CEMs in CEML v3. We will be both manually transforming and programatically transforming diffirent sets of models where appropriate. Collections of models with similar patterns are better candidates for programatic transformation and will be transformed using XSLT.

At this point, the stack of transformed CIMI Style CEMs can be compiled and validated in the CEML v3 compiler. If the transformed CEMs are all valid, the transformation can continue with a generic syntax transformation to CIMI BMM and ADL Archetypes. With the final target completed, the entire stack of CIMI models can be compiled and validated.

Despite the different paradigm between CIMI and CEM, where CIMI describes clinical content in both the reference and constraint model, and CEM uses only the constraint model, there is actually a close parallel between the two. Figure 9, "CEML to BMM/ADL Transformation", shows that the identifier declarations within CEML map closely to the CIMI BMM reference model. And the constraints within CEML map closely to the CIMI Archetype ADL constraints. This parallelism will allow us to build the generic transformation between these two formalisms.

Figure 9. CEML to BMM/ADL Transformation



4.2. Datatype Issues

Even with the similarities, there still remains a few problems for the generic transformation engine. First, CIMI and CEM use a different set of datatypes. Some of these align quite nicely, such as CIMI's CODED_TEXT and CEM's CodeableConcept which are both used to represent coded fields. Others such as CIMI's YESNO and Boolean datatypes have only a partial alignment to CEM's Boolean datatype. A second problem is that the CIMI BMM allows properties to be named datatypes. In other words, the property specimenType could be a coded field in the BMM. This is not allowed in a CEM, where the property specimenType must point to another CEM model and NOT directly to a datatype.

CEML v3 solves both of these problems as we are expanding our allowed datatypes to be more closely aligned with CIMI. Also, we are adding a syntactic feature which allows an optional property name to be applied to a datatype.

4.3. BMM Packaging

Another issue regarding the generic transformation involves the packaging of CIMI models within a BMM. CIMI currently has three BMM files called 'Core', 'Foundation', and 'Clinical' and within each of these, there are multiple packages, and then a random order to the models within these packages. CEM models, on the other hand, each exist in their own file and currently have no method to indicate a package. In CEML v3 we are allowing each CEM to be assigned to a package similar to JAVA packages.

4.4. Recursion Issues

Another problem with the generic transformation is that CIMI models within the BMM allow recursion. For example, a Substance model could reference an Ingredient model which could then reference the original Substance model creating an infinite recursive tree from Substance on down. Although the CEM Abstract Instance Model is recursive with respect to the generic Clinical Element, CEML currently does not allow recursive constraint models. The simplest solution here is to modify the CEM compiler to allow recursion. It should be noted that the CEM compiler did allow recursion in the past, and it is being reintroduced for CEML v3.

4.5. ADL and CEML Constraint Consistencies

ADL and CEML Constraints used to bind terminology to the model, such as simple binding of about codes and valuesets have high consistency and should pose no problems in the transformation. Also, cardinality constraints within the two should pose no problems.

4.6. ADL and CEML Constraint Inconsistencies

One major difference that exists between CEML and ADL is the ability of one constraint model to peek into another constraint model and further constrain that model. This is called an 'inner constraint', as the outer model is constraining the inner model. This can be seen in Example 1, “CEML for Systolic Blood Pressure”, where the SystolicBloodPressure model constrains the valueset of the inner BodyPosition model. ADL does not have a mechanism to step into another ADL model and further constrain that inner model. But ADL does have the ability to walk a deep path in the BMM to apply a constraint, so in many cases it won't be a problem. In the CEML example above, this problem could be solved by creating a SystolicBloodPressureBodyPosition model thus constraining with a new model rather than an inner constraint. CEML v3 will still allow 'inner constraints', but they will not be used to constrain Complex models in our collection of CIMI style models. Also, CIMI is currently evaluating the need for inner constraints and could result in ADL changes in the future.

Another problem is the random assignment of id's in ADL such as 'id2', 'id5', and 'id34'. CIMI has made the decision that these id's are a problem for all tooling, and are currently developing an algorithm to dictate their naming. This algorithm will allow any transformation to calculate the correct 'id'

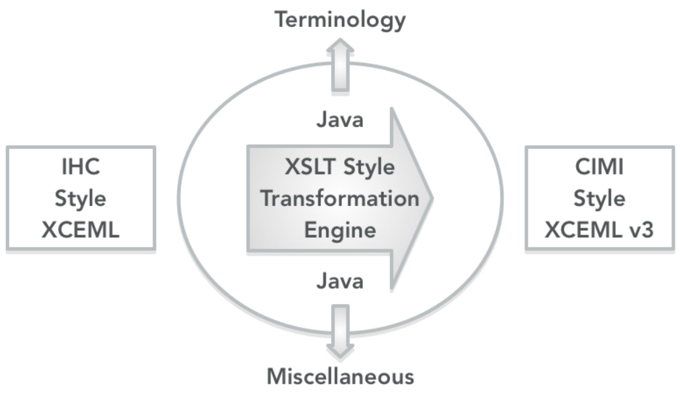
4.7. Style Transformation

The Style Transformation, as seen in Figure 10, “Style Transformation”, will use existing Intermountain Healthcare CEMs as the source of the transformation. These CEMs currently exist both in CEML and in

XCEML with the latter being an XML representation of the CEML syntax. It is the XCEML form that will be used as input for both manual and the style transformation engine which will use XSLT for the transformation, and generate CIMI style CEMs in XCEML v3 syntax. The XSLT engine will provide XSLT functions to simplify writing transformation rules for the modeler. Functions will also be provided to call external java functions for terminology lookups or complexities that arise such as CEML path parsing.

It should be noted that although Intermountain Healthcare has finished the models we require, they have not finished all of the terminology mapping for these models. It is likely that all terminology mapping will not be completed and will be an ongoing endeavor.

Figure 10. Style Transformation



5. Conclusion

Our previous transformation work between CEMs and CIMI revealed expressivity problems with CEML v2 which are now being addressed in CEML v3. Although it is probable we will encounter issues not discussed in this white paper, Intermountain is committed to solve these problems with any needed changes to CEML v3. Using the transformation methodology presented will give us a high probability for success in the transformation of all Intermountain CEMs to CIMI models.