Clinical Element Model

Joey Coyle
Yan Heras
Tom Oniki
Stan Huff

November 14, 2008
# Contents

1 Clinical Data Models ........................................... 7
   1.1 Intro .......................................................... 7
   1.2 General Requirements ..................................... 10

2 The Clinical Element Model .................................. 11

3 CE Abstract Instance Model .................................. 15
   3.1 Introduction .................................................. 15
   3.2 Type ............................................................ 16
   3.3 Key ............................................................. 17
   3.4 Value Choice ................................................ 17
   3.5 Data ........................................................... 18
   3.6 Items ........................................................... 19
   3.7 Quals and Mods ............................................. 19
   3.8 Attribution .................................................... 21
   3.9 Instance ID .................................................. 22
   3.10 Alternative Data ............................................ 22

4 Abstract Instance Model Spec ............................... 23
   4.1 Introduction .................................................. 23
   4.2 CEInstance .................................................... 23

5 CE Abstract Constraint Model ............................... 25
   5.1 Intro ............................................................ 26
      5.1.1 name ...................................................... 27
      5.1.2 base ...................................................... 27
      5.1.3 kind ...................................................... 27
      5.1.4 constraint .............................................. 27
6 Understanding Paths
6.1 key ...................................................... 29
   6.1.1 key.code .............................................. 29
   6.1.2 key.domain ........................................... 30
6.2 data .................................................... 30
   6.2.1 data.type .......................................... 30
   6.2.2 data.cwe .......................................... 30
   6.2.3 data.pq ............................................. 31
6.3 Sequences ............................................ 32
   6.3.1 type .................................................. 32
   6.3.2 card ................................................... 33
   6.3.3 Recursed paths ....................................... 34

7 Clinical Element Kind .................................. 35
7.1 Statement ............................................ 35
   7.1.1 Simple Statement .................................. 36
   7.1.2 Compound Statement ............................... 37
7.2 Panel ................................................... 37
7.3 Component ............................................ 38
   7.3.1 Simple Component .................................. 38
   7.3.2 Compound Component ............................... 38
7.4 Noninstantiable ....................................... 38
7.5 Modifier ............................................... 39
7.6 Attribution ............................................ 39

8 Modifiers and Qualifiers ................................ 41
8.1 Introduction .......................................... 41
8.2 Modifiers ............................................. 41
   8.2.1 Negation or Certainty Modifier .................... 42
   8.2.2 Subject Modifier .................................... 42

9 Clinical Element Attribution ............................ 43
9.1 Modeling ............................................... 43
9.2 Subtypes .............................................. 44

10 Semantic Links ........................................ 47
10.1 Versus a Qualifier ..................................... 47
   10.1.1 Tightly versus Loosely Coupled ................. 48
10.2 Target as Concept ..................................... 49
10.3 Modeling .............................................. 49
10.3.1 Syntax .......................... 50
10.4 Inverse Relationships ................. 50

11 Scope .................................. 51
  11.1 Modifiers .......................... 51
  11.2 Attributions ......................... 52
  11.3 Qualifiers ......................... 52
  11.4 Modeling ......................... 53
    11.4.1 Deterministic ................. 53
    11.4.2 Non-deterministic .......... 53

12 Clinical Element Root ................. 55
  12.1 History ........................... 55
  12.2 Implementation ................... 55

13 CEML .................................. 57
  13.1 Introduction ....................... 57
  13.2 CEM versus CETypes ................ 58
  13.3 Paths .............................. 58

14 CEML Authoring Syntax ............... 59
  14.1 Root Element <ceml> ............... 60
  14.2 <header> .......................... 60
  14.3 <cetype> ......................... 61
    14.3.1 name attribute of <cetype> 62
    14.3.2 base attribute of <cetype> 62
    14.3.3 kind attribute of <cetype> 62
  14.4 <key> ............................. 62
  14.5 <data> ............................ 63
  14.6 <qual>, <item>, <att>, and <mod> 63
    14.6.1 name attribute of <qual> 65
    14.6.2 type attribute of <qual> 66
    14.6.3 card attribute of <qual> 66
  14.7 constraint ....................... 66
  14.8 absence ........................... 67
  14.9 link .............................. 67

15 Data Choice .......................... 69
  15.1 Alphabetical Rule ................. 70
  15.2 Use Requirements ................. 70
## CONTENTS

### 16 CEML Tutorial

16.1 Data ................................................. 73
16.2 Qualifiers ........................................... 73
16.3 Modifiers ........................................... 74

### 17 Understanding Instance Data

17.1 Introduction ......................................... 75
17.2 Constraints and Instances ............................ 76
17.3 Serialization ......................................... 76

### 18 Glossary

18.1 Definitions ........................................... 77

### 19 Change Log

19.1 August 15, 2008 ...................................... 83
19.2 September 15, 2008 ................................... 83
19.3 Nov 13, 2008 .......................................... 84
19.4 Nov 14, 2008 .......................................... 84
Chapter 1

Clinical Data Models

1.1 Intro

Detailed clinical models are the basis for retaining computable meaning when data is exchanged between heterogeneous computer systems. Detailed clinical models are also the basis for shared computable meaning when clinical data is referenced in decision support logic. Exactly what we mean by “detailed clinical models” and how they relate to the use of clinical data by computers will be described below.

There are a number of motives for exchanging clinical data between heterogeneous computer systems. Data can be exchanged between different computers within a facility or enterprise in order to make information available to clinicians at the point of care, with the goal of improving clinical decision making. Serology and culture results can be sent from clinical laboratories to public health departments as a means of detecting an epidemic or a bioterrorism attack. Health care providers that are participating in clinical trials of new medications or other therapies need to exchange clinical data as part of the research protocols. In all of these situations, the goal is not just to have the data available for humans to read and understand, but to have the data structured and coded in a way that will allow computers to understand and use the information.

The most common strategy for representing data that is sent between different computer systems is to send the data as name-value pairs (also known as entity-attribute-value triplets. For example, if the results of a hematocrit test were to be sent between two systems the data could be represented as:

- Test name = Hematocrit, Value = 44.1%

The Health Level 7 (HL7) and Digital Imaging and Communications in Medicine (DICOM) standards use this strategy, and the Logical Identifier Names and Codes
(LOINC) coding scheme was created in order to supply the “name” part of the name-value pair. The use of standardized codes (and their associated computable definitions) as the names of test results allows computers to use the information in decision logic, outcomes research, and other clinical calculations.

Single valued measurements like a hematocrit result are represented easily in a single name value pair as shown above. However, as soon as the clinical measurement is slightly more complicated, variations in how the data can be represented present themselves. For example, there are at least three ways of representing the results of patellar deep tendon reflexes.

- A single name/code and value
  - Left patellar deep tendon reflex intensity is 2+

- Combination of two names/codes and values
  - Patellar deep tendon reflex intensity is 2+
  - Laterality is left

- Combination of three names/codes and values
  - Deep tendon reflex intensity is 2+
  - Body part is patella
  - Laterality is left

When the complex nature of the data allows these different options for representation, it is important to understand that these representations are equivalent, otherwise a computer processing the data will not recognize the alternative forms and will fail to use the data appropriately. The ability of a computer to recognize the equivalence of these statements is based on an underlying detailed clinical model. For the example given, the detailed clinical model could be stated as:

- Type of measurement - (intensity of deep tendon reflex)
- Location of measurement - (patella, or patellar tendon)
- Laterality of measurement - (left side)

For a computer to recognize the equivalence of the three different statements, there must be a more formal way of stating the information model and of referencing standardized terminologies that are used for the names of data elements in the model. Use of standard models and associated standard reference terminologies will enable a computer to detect equivalent representations.
Many examples of alternative data representation exist. A more difficult example than patellar reflex data is the problem of lung auscultation. The results of lung auscultation can be represented either in a finding focused style or a location focused style. For example, one can state the finding “wheezing”, and then state every lung location where it was heard, such as “wheezing in the right and left upper lobes.” Alternatively, one could be location focused, and state the location “right upper lobe”, and state all the findings associated with that location, such as “the right upper lobe has wheezing, rales, and egophony.”

The need for a formal way of representing detailed clinical models is closely related to what has been termed the “curly braces problem.” The curly braces problem arises from the practical issues of trying to implement medical logic modules (MLMs) such as rules, alerts, and reminders using Arden Syntax. Arden syntax is an HL7 standard for representing medical decision logic. In Arden syntax, data slots are used to create “read” statements that retrieve data that participates in the logic of the MLM from the patient’s EMR (or some other data store). Curly braces are used within the “read” statement “to isolate institution-specific portions [of data access] to one slot. Within the data slot, the institution-specific portions are placed in mapping clauses so that the institution-specific part does not interfere with the MLM syntax.” The following snippet from an MLM shows the use of curly braces.

```plaintext
data:

/* creatinine in mg/dL*/
creatinine := read last {select value from lab where code = 237}
(more data declarations)
evoke

/* execute this logic each time a new calcium is stored*/
storage_of_calcium;

logic:

/* if creatinine is present and greater than 6, then stop now */
IF creatinine is present THEN
   IF creatinine is greater than 6.0 THEN
      conclude false
   ENDIF
ENDIF
...(more logic statements)
```

Figure 1.1: MLM snippet demonstrating Curly Braces.

In Figure 1.1, curly braces are used to enclose a SQL statement that would retrieve the patient’s creatinine from a (hypothetical) relational database containing laboratory results. Pryor and Hripscak demonstrated that a major obstacle to
1.2. GENERAL REQUIREMENTS

The need for detailed clinical models has been recognized by researchers and standards organizations. DICOM, Centre for European Normalisation (CEN), Good Electronic Health Record (GEHR), HL7, GALEN, and Stephen Johnson have either developed or plan to develop a mechanism for describing and sharing detailed clinical models. In the following chapters we will discuss the development of IHC’s third generation clinical data model called the Clinical Element Model.

1.2 General Requirements

- The model must be comprehensive - it must accommodate representation of anything that can be stated about a patient.
- The model must be flexible and extensible - it must be possible to add elements and attributes to the model without requiring changes to underlying software and database.
- It must use an existing formalism (XML Schema, ASN.1, Conceptual Graphs, etc.) without modification.
- There must be a tight linkage to standard terminologies such as LOINC, Systematized nomenclature of medicine - Clinical Terms (SNOMED CT), HL7 Vocabulary Tables, etc.
- There must exist a mechanism to state negation, in order to say that something was NOT observed or was NOT present.
- A process for change management must be followed, in order to know which version of a model was in effect at the time data was stored.
- There is a need to easily change the cardinality of values; for example, to note a single complication versus selecting all the complications that apply.
- There must exist the ability to allow any degree of arbitrary collections and batteries.
- It must be possible to retain as part of the permanent record how the data was originally seen by the user, or as it was sent by an application.
Chapter 2

The Clinical Element Model

In order to represent detailed clinical data models, we have designed The Clinical Element Model (CEM). When we state “The Clinical Element Model” we are referring to the global modeling effort as a whole, or in other words, our approach to representing detailed clinical data models and the instances of data which conform to these models. The Clinical Element Model is the combination of an Abstract Instance Model and an Abstract Constraint Model. The Abstract Instance Model defines a structure to represent instances of medical data, and the Abstract Constraint Model defines constraints on values in the Abstract Instance Model. (See Figure 2.1)

The Abstract Instance Model is a structure which can represent individual instances of collected data; for example, the Systolic Blood Pressure measurement collected on John Doe, on May 13, 2007, at 2:45 P.M. The values in this Systolic Blood Pressure data, must conform to the constraints or rules stated in the corresponding Constraint Model for Systolic Blood Pressure.

Both the Abstract Instance Model and the Abstract Constraint Model are, as stated, described abstractly, thus they must both be implemented using an Implementation Technology Specification (ITS). Later in this document, we will present an implementation of both the Abstract Instance Model and the Abstract Constraint Model using XML, but it should be understood, that a different ITS could use XML, Java, C, or Objective-C. And each of these abstract models could use a different ITS. For example, the Abstract Instance Model could be implemented in Java, and the Abstract Constraint Model Could be implemented in XML. For those familiar with Clinical Element Modeling Language (CEML), CEML is an implementation of the Abstract Constraint Model using XML as a syntax. (See Figure 2.2)

In the following chapters we will define each of these abstract models,
Figure 2.1: The CE Abstract Constraint Specification describes the constraints on the CEM Abstract Specification Instance

which together make up what is called The Clinical Element Model. The reader should be fully knowledgeable about our defined datatypes before continuing on. If you do not understand the datatypes such as CWE, PQ, or TS, then please browse the document titled CE Datatypes to learn a little about these datatypes prior to continuing on with this document.
Figure 2.2: The use of the Clinical Element Model involves implementing both the Abstract Instance Model and the Abstract Constraint Model.
Chapter 3

CE Abstract Instance Model

In this chapter, we will examine the Clinical Element Abstract Instance Model. The Abstract Instance Model defines the structure which is used to represent instances of medical data. An instance of medical data is created each time the patient has information added to the medical record. For example, if John Doe had 3 blood pressures taken, then 3 instances of medical data would be added to his medical record. Each of these instances would describe the details of a particular blood pressure. If John Doe then had a serum glucose measurement, then an instance describing this result would be added to the medical record. The patient’s medical record thus becomes a collection of thousands of individual instances of medical data. The Abstract Instance Model describes the structure to represent all of these possible instances. We call these instances, Clinical Element Instances. In this chapter, if the term Instance Model is used, we are referring to the Clinical Element Abstract Instance Model.

The Instance Model is a structure designed to hold instance data, and the structure does not change regardless of the type of instance data. Because of this, nonsense data can be stored in a CE Instance, but this is avoided because each CE Instance is linked to a specific constraint specification called a Clinical Element Constraint Type, which is an instance of the Abstract Constraint Model (Figure 2.2).

3.1 Introduction

The heart of our approach is a recursive model with the core recursive element being the Clinical Element. Thus the Instance Model is a tree of Clinical Element nodes. In this chapter, we will describe the parts of parts of this recursive Clinical Element which makes up the abstract instance model.
3.2. Type

If we examine the most basic skeleton of the Clinical Element (Figure 3.1), there is a type, a key, and a value choice. The type is a coded value which identifies the CE Constraint Type to which the instance will conform. The key is a coded value for the real world concept that is important or key to what the instance is attempting to describe.\(^1\) The value choice is a choice between a data property or items, where data is a derivative\(^2\) of the HL7 version 3 datatype ANY, and items is a sequence of one or more Clinical Elements which gives the model its recursive nature. Later we will learn about the remaining Clinical Element properties such as mods, quals, and atts.\(^3\)

Figure 3.1: Clinical Element Instance Model

![Clinical Element Instance Model Diagram]

3.2 Type

The Clinical Element type property is a coded value of type CNE\(^4\), which specifies the CE Constraint Type, known as a cetype, to which this instance conforms. The allowable values for type consist of the domain of all defined CE Constraint Types. Our current ITS for the Abstract Constraint Model is called CEML, and the defined

---

\(^1\) Note, need to resolve type-key dependency, as type has no real meaning except as a receptacle for the constraints to which this instance conforms.

\(^2\) We have defined our own datatypes based on the HL7 version 3 datatypes which are described elsewhere in this document.

\(^3\) links are interesting case, they are not physically part of the instance model, but conceptually they are part of the instance model. Allowable links are constrained by the abstract constraint model, so maybe we should consider them part of the instance model to be consistent.

\(^4\) We could make type a CS
CHAPTER 3. CE ABSTRACT INSTANCE MODEL

3.3 Key

The key is a coded value represented by a CWE datatype. The key code is a code for a real world concept that is important or key to what the model is attempting to describe. An example of a real world concept is Serum Sodium, which has nothing to do with Clinical Elements or computers. The concept of Serum Sodium exists in the real world and has a known meaning in the field of medicine. It can be said that the key code links the Clinical Element Instance to a real world concept. In Figure 3.2 is an example of some partial instance data, that conforms to the constraints specified in the constraint type LabObservation.

Figure 3.2: Clinical Element instance with key and data values, and constrained by type LabObservation

3.4 Value Choice

The heart of a Clinical Element Instance is it’s value, which is the payload. The value is a choice between either the property Data or Items where the former is a leaf node, and the latter is a list of children Clinical Elements. The next two sections will discuss these in more depth.

---

5LabObservation is a type used to represent Quantitative Lab Observation data such as 140 mEq/L or 4.2 mmol/L.
3.5 Data

The *data* property is represented by an HL7 version 3 datatype, and is used to represent values such as numbers, strings, and codes. At IHC, we are actually using a subset of all the allowed HL7 datatypes and in fact we have modified these slightly. These datatypes are described in detail in a separate document called Clinical Element Datatypes. If a Clinical Element Instance instantiates the *data* attribute instead of *items* attribute, this Clinical Element node becomes a leaf node in the Clinical Element instance tree.

**Clinical Element Figures in this Book**

From this point on, to reduce the size of figures for this book, we are going to typographically place values for the Clinical Element Instance property *type* in the place we have been putting the word Clinical Element. An example of this can be seen in Figure 3.3. Another typographic form that will be used to conserve a lot of page space will move the value for the *key* code next to the the *type* name. This can be seen in Figure 3.4. Do not be confused by these diagrams into thinking the constraint types indicates structure, because the constraint types only limit values in the single structure which is the Clinical Element Instance.

Figure 3.3: Type placement to reduce figure size

![Figure 3.3](image1)

Figure 3.4: Key code placement to reduce figure size

![Figure 3.4](image2)
3.6 Items

The Abstract Instance Model defines the items property as a sequence of child Clinical Element nodes. This is the functionality that gives the Clinical Element Instance it’s recursive nature with the ability to represent complex nested data. An Example of a Clinical Element Instance that uses items is seen in Figure 3.5. Here we have instance data that conforms to the constraint type BloodPressurePanel, which allows two child instances in items: One that conforms to the constraint type SystolicBloodPressure and a second that conforms to the constraint type DiastolicBloodPressure.

Figure 3.5: Instance Data with Items that conforms to the constraint type BloodPressurePanel

![BloodPressurePanel Diagram]

3.7 Quals and Mods

The Abstract Instance Model also defines two other collections of Clinical Element nodes which serve to alter the meaning of the instance. These two lists of Clinical Elements are called quals and mods, which stands for Qualifiers and Modifiers. They are named to represent the extent to which they alter the meaning of the instance. See Figure 3.6 to see where the properties quals and mods fit into the Abstract Instance Model. A Clinical Element node that is in mods, which we will call a Modifier, alters the meaning of the instance to such an extent that one can never use the instance data without considering the effect of the Modifier. A
Clinical Element node that is in *quals*, which we will call a Qualifier, is considered to add information to the value choice and doesn’t actually change the meaning of the value choice in a way that makes it dangerous to ignore this change.⁶

Figure 3.6: CE Abstract Instance Model with Mods and Quals

In Figure 3.7 is an example of adding a qualifier to the instance constrained as a BloodPressurePanel in Figure 3.5. In Figure 3.7, we specify the BodyPosition of the patient as “Sitting” when the Blood Pressure was measured. This qualifier applies to the value choice, which is *items* in this case, which means the qualifier applies to SystolicBloodPressure and DiastolicBloodPressure.⁷ In Figure 3.8 is an equivalent instance example of how the BodyPosition qualifier affects SystolicBloodPressure. Here this BodyPosition applies to the *data* attribute containing 120 mmHg which is the actual measurement. So this is a simple example demonstrating the scope of a qualifier, and how a qualifier within a panel applies to the children in the panel.⁸

---

⁶Whether or not qualifiers can actually ever be truly ignored is debated in informatics circles.
⁷Specific rules stating how qualifiers and modifiers affect the children contained in *items* will be explained later.
⁸The children of a panel are either statements or other panels.
3.8 Attribution

The Abstract Instance Model also defines another collection of Clinical Element Instance nodes similar to qualifiers called attributions. An attribution has a specific structure which defines an action, and the who, where, why, and when information.
regarding that action. This list of attributions is called atts.

3.9 Instance ID

The Abstract Instance Model defines the property instanceId to be a unique identifier for each Clinical Element Instance node. It must be unique across the enterprise, but we recommend it to be globally unique across all enterprises. To guarantee a globally unique identifier we have chosen to use a GUID to populate this property.

3.10 Alternative Data

The Abstract Instance Model defines the property alt to be a choice between a CWE, PQ, or an ED datatype. The purpose of alt is to allow the collection of unexpected or alternative data representations in an instance. For example, suppose Systolic Blood Pressure data was being collected into instances, and those instances were constrained by a constraint type called SystolicBloodPressure which required data to be a PQ. But then the system received a Systolic Blood Pressure measurement with a coded value of “HIGH”. Since the Abstract Instance Model supports all data, this coded value of “HIGH” could be put into the data property as a CWE, but then the instance would fail validation by the constraint type SystolicBloodPressure. So instead, the coded value of “HIGH” can be placed into a CWE within alt, and “null” can be placed in the data property, with a corresponding null flavor. An example can be seen in figure 3.9 where it can be seen that the expected data section instead has a nullFlavor9, and the coded value of HIGH is put in the alt section.

Figure 3.9: Instance Data demonstrating the alt property

<table>
<thead>
<tr>
<th>SystolicBloodPressure</th>
<th>SystolicBP</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>nullFlavor</td>
</tr>
<tr>
<td>alt</td>
<td>HIGH</td>
</tr>
</tbody>
</table>

9 nullFlavor is a property of the HL7 version 3 datatypes.
Chapter 4

Abstract Instance Model Spec

4.1 Introduction

The properties of CEInstance are defined by the Abstract Instance Model. The CHOICE construct indicates that one datatype is chosen from the set. The SEQUENCE construct indicates 0 to Many of the indicated type.¹

4.2 CEInstance

Table 4.1: The properties of CEInstance

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Cardinality</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>CNE</td>
<td>1</td>
</tr>
<tr>
<td>key</td>
<td>CNE</td>
<td>1</td>
</tr>
<tr>
<td>data</td>
<td>CHOICE&lt;CWE, CO, ST, PQ, IVLPQ, RTOPQ, TS, II, INT, REAL, ED&gt;</td>
<td>0-1</td>
</tr>
<tr>
<td>items</td>
<td>SEQUENCE&lt;CEInstance&gt;</td>
<td>0-1²</td>
</tr>
<tr>
<td>quals</td>
<td>SEQUENCE&lt;CEInstance&gt;</td>
<td>0-1</td>
</tr>
<tr>
<td>mods</td>
<td>SEQUENCE&lt;CEInstance&gt;</td>
<td>0-1</td>
</tr>
<tr>
<td>atts</td>
<td>SEQUENCE&lt;CEInstance&gt;</td>
<td>0-1</td>
</tr>
<tr>
<td>instanceId</td>
<td>ST</td>
<td>1</td>
</tr>
<tr>
<td>alt</td>
<td>CHOICE&lt;CWE, PQ, ED, ST&gt;</td>
<td>0-1</td>
</tr>
</tbody>
</table>

¹Issue: Instead of SEQUENCE, should we change lists like items to item where item can occur 0-M times. This would then correlate exactly with constraint model.
Actually, there is a choice between the properties data and items, and one or the other must exist.
Chapter 5

CE Abstract Constraint Model

The Clinical Element Abstract Constraint Model defines the allowable constraints that may be placed on the CE Abstract Instance Model. It is important to understand that any information, even nonsense patient data can be instantiated in a CE instance, so it is the role of the Constraint Model to ensure that instances contain medically meaningful data. In the Abstract Constraint Model, the constraints are defined abstractly, but in a later chapter, CEML is defined, which is an XML ITS of the CE Abstract Constraint Model.

Figure 5.1: The CE Abstract Constraint Model describes the constraints on the CE Abstract Instance Model
5.1 Intro

The properties of the constraint model are very simple. In Figure 5.2, you can see that it is actually just a named collection of individual constraints. First, there is a name property which is used to name the collection of constraints. We call this named collection of constraints a CE Constraint Type, or a CEType. Next, is the base property which can be optionally used to import another named collection of constraints as a starting point; (base) is a reference to another CEType. And finally, there is a list of one to many constraint structures, where each states the part of the instance model to constrain, and how to constrain it.

Figure 5.2: The CE Abstract Constraint Model describes the constraints on the CE Abstract Instance Model

<table>
<thead>
<tr>
<th>CEType</th>
<th>name</th>
<th>base</th>
<th>kind</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraint</td>
<td>path</td>
<td>value</td>
<td></td>
</tr>
<tr>
<td>constraint</td>
<td>path</td>
<td>value</td>
<td></td>
</tr>
<tr>
<td>constraint</td>
<td>path</td>
<td>value</td>
<td></td>
</tr>
<tr>
<td>constraint</td>
<td>path</td>
<td>value</td>
<td></td>
</tr>
<tr>
<td>constraint</td>
<td>path</td>
<td>value</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.1: CEType Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Cardinality</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>one</td>
</tr>
<tr>
<td>base</td>
<td>zero to one</td>
</tr>
<tr>
<td>kind</td>
<td>one</td>
</tr>
<tr>
<td>constraint</td>
<td>one to many</td>
</tr>
</tbody>
</table>
5.1.1 name

The name property represents the unique textual identifier for this collection of constraints; it is the name of the CEType. Some textual examples could include VitalSignsPanel, SystolicBloodPressure, and AbnormalFlag. A named collection of constraints is called a CE Constraint Type or CEType. So VitalSignsPanel is the name of a CEType\(^1\) which defines the constraints for instance data which would represent a vital signs panel.\(^2\)

5.1.2 base

The base property identifies an external named collection of constraints, or more specifically, an external CEType that is used as the starting point for this new collection of constraints. For example, when defining the constraints for Serum-Sodium, the base may be set to LabObservation which imports all the general constraints of a LabObservation, and then the specifics of a SerumSodium can be specified in the constraint section.

5.1.3 kind

The kind property declares a functional category for the defined CEType. The options include “noninstantiable”, “component”, “statement”, “panel”, “modifier”, and “attribution”. See Table 5.2.

5.1.4 constraint

The constraint structure represent individual constraints on the Abstract Instance Model. The constraints are where everything happens, and is where the Abstract Constraint Model is tied to the Abstract Instance Model. Each constraint consists of a path and a value. The path property identifies a specific location in the CE Abstract Instance Model tree, and the value property is the value this location is constrained to.

path

The path property represents the location in the CE Abstract Instance Model which is being constrained.

---

\(^1\)We have been very loose in our use of terminology, and many people call these CE Models, or CEMs, but to be accurate they should be called a CE Constraint Type or CEType.

\(^2\)It should be noted, that the name of a CEType is meaningless, and VitalSignsPanel could have been called VSignsPanel, VitalsPanel, or anything the modeler wished.
Table 5.2: Allowable values for kind

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noninstantiable</td>
<td>No instantiations will be constrained directly with this CEType. An incomplete definition that must be further defined to become a statement, panel, or component.</td>
</tr>
<tr>
<td>component</td>
<td>CEType will be used to constrain part of a storable instance, such as an internal qualifier.</td>
</tr>
<tr>
<td>attribution</td>
<td>CEType will be used to constrain an attribution.</td>
</tr>
<tr>
<td>modifier</td>
<td>CEType will be used to constrain a modifier</td>
</tr>
<tr>
<td>statement</td>
<td>CEType will be used to constrain a complete storable instance using data</td>
</tr>
<tr>
<td>panel</td>
<td>CEType will be used to constrain a complete storable instance using items, and the items must be either statements or panels.</td>
</tr>
</tbody>
</table>

value

The name property represents the value to which the chosen position in the CE Abstract Instance Model should be constrained.
Chapter 6

Understanding Paths

The only difficulty in understanding the Abstract Constraint Model, is understanding the allowable paths that can be constrained. The follow sections detail the allowable paths and explain the corresponding values. A complete understanding of the CE Abstract Instance Model, as well as a complete understanding of the datatypes is essential to understanding these paths, because the paths are paths into the Instance Model and into the datatypes. When a path gets to the level of a datatype, then you should consult the CE Datatypes document to learn about all the allowable properties for that datatype.

The following sections will explore the more commonly used paths, but is not a complete list, as this chapter will not list every property of every allowable datatype. The properties within a datatype are divided into value and rule types. A value type is simply constraining a direct instance property within the datatype, and to validate one must simply make sure the value in the instance matches the constraint. A rule type is constraining one of the rule properties that were added to the datatypes, and logic must be used to understand and enforce this constraint.

6.1 key

The Abstract Instance Model declares the property key as a CWE, so all the properties within CWE are available for constraint. Table 6.1 lists the most common properties of CWE that are used within key for constraint. See the CE Datatypes document for additional properties.

6.1.1 key.code

Constrains the key property to have a code with the specified value.
6.2 DATA

CHAPTER 6. UNDERSTANDING PATHS

Table 6.1: Examples of constraint paths for key

<table>
<thead>
<tr>
<th>Path</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>key.code</td>
<td>value</td>
</tr>
<tr>
<td>key.domain</td>
<td>rule</td>
</tr>
</tbody>
</table>

6.1.2 key.domain

 Constrains the key property to have a code within the specified domain.

6.2 data

 The data property is constrained to hold one of the allowed datatypes as indicated in the Abstract Instance Model. This constraint is indicated with the path data.type

6.2.1 data.type

 To indicate the datatype the path data.type is used, and then a lowercase version of the datatype is used as a value, such as cwe, co, st, pq, ivlpq, rtopq, ts, ii, int, real, or ed.\(^1\)

 Depending on what datatype is selected, the path will continue on with that datatype, followed by the allowable properties of that datatype. We will demonstrate a few possible paths using CWE and PQ as an example. Also note that this is not a complete list of all the properties of either CWE or PQ. View the CE Datatypes document to see all the properties of every datatype.

6.2.2 data.cwe

 This is the beginning of a path into data, and then into the datatype cwe. The path could be continued at this point with any of the the properties in CWE. See Table 6.2 for a couple of examples.

 data.cwe.code

 Constrains the cwe datatype instance to have a code with the specified value.

---

\(^1\)Recently we have implement datatype choices which is a comma delimited list of datatypes. Please read the specific chapter on datatype choices for more on this.
Table 6.2: Examples of constraint paths for data.cwe

<table>
<thead>
<tr>
<th>Path</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>data.cwe.code</td>
<td>value</td>
</tr>
<tr>
<td>data.cwe.domain</td>
<td>rule</td>
</tr>
</tbody>
</table>

**data.cwe.domain**

Constrains the cwe datatype instance to have a *code* within the specified domain.

### 6.2.3 data.pq

This is the beginning of a path into *data*, and then into the datatype *pq*. The path could be continued at this point with any of the properties in PQ. See Table 6.3 for a few examples.

Table 6.3: Examples of constraint paths for data.pq

<table>
<thead>
<tr>
<th>Path</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>data.pq.unit.code</td>
<td>value</td>
</tr>
<tr>
<td>data.pq.unit.domain</td>
<td>rule</td>
</tr>
<tr>
<td>data.pq.unit.normal</td>
<td>rule</td>
</tr>
</tbody>
</table>

**data.pq.unit.code**

Constrains the *unit* property within the *pq* datatype instance to have a *code* with the specified value.

**data.pq.unit.domain**

Constrains the *unit* property within the *pq* datatype instance to have a *code* within the specified domain.

**data.pq.normal**

Declares the unit that should be used for the normalized value in the persistent datastore.
6.3 Sequences

The CE Abstract Instance Model declares items, quals, atts, and mods to be a list of 0 to many Clinical Element Instance nodes. These are the points of recursion in the CE Abstract Instance Model. In the Abstract Constraint Model, we have defined a path to indicate constraints on this sequence, and the recursion allows us to continue the path to the key, data, etc that exist in the recursed CE Instance node. The syntax to constrain a CE Instance node within one of these lists is defined by beginning the constraint path with either “item.”, “qual.”, “att”, or “mod.”, followed by a unique identifier, and then assigning the value to an existing defined CEType. See Figure 6.1. We realize that according the Abstract Instance Model, the true path should be “items.”, “quals.”, “atts”, and “mods.”, but in use we have found the singular form to be more readable and understandable for most.² The unique identifier has no meaning so it can be any textual string, but the person creating the unique identifier usually follows the convention to use a lowercase version of the referenced CEType.

Figure 6.1: A unique textual identifier is used to constrain a particular CE Instance node within items, quals, atts, and mods.

<table>
<thead>
<tr>
<th>path</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>qual.identifier.type</td>
<td>ReferencedConstraintType</td>
</tr>
<tr>
<td>qual.status.type</td>
<td>Status</td>
</tr>
<tr>
<td>item.serumSodium.type</td>
<td>SerumSodium</td>
</tr>
<tr>
<td>mod.subject.type</td>
<td>Subject</td>
</tr>
<tr>
<td>att.verified.type</td>
<td>Verified</td>
</tr>
</tbody>
</table>

6.3.1 type

The type property assigns a referenced CEType to the identifier.

²Issue: Should we change the Abstract Instance Model to use the singular form.
Figure 6.2: Any nonsense identifier can be used, but it is standard to use a lower-case version of the referenced CEType

\[
\text{path} = \text{"qual.myStatus.type"} \quad \text{value} = \text{"Status"}
\]

\[
\text{path} = \text{"qual.status.type"} \quad \text{value} = \text{"Status"}
\]

### 6.3.2 card

The `card` property constrains the cardinality of the referenced CEType. This constraint is a rule based constraint rather than a value based constraint. The allowable values for `card` are indicated in Table 6.4. The `path` to assign the cardinality is indicated in Table 6.5.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Zero</td>
</tr>
<tr>
<td>1</td>
<td>Exactly One</td>
</tr>
<tr>
<td>0-1</td>
<td>Zero to One</td>
</tr>
<tr>
<td>0-M</td>
<td>Zero to Many</td>
</tr>
<tr>
<td>1-M</td>
<td>One to Many</td>
</tr>
</tbody>
</table>

Table 6.4: Allowable values for the property `card`

<table>
<thead>
<tr>
<th>Path</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>item.\text{identifier}.\text{card}</td>
<td>rule</td>
</tr>
<tr>
<td>qual.\text{identifier}.\text{card}</td>
<td>rule</td>
</tr>
<tr>
<td>mod.\text{identifier}.\text{card}</td>
<td>rule</td>
</tr>
<tr>
<td>att.\text{identifier}.\text{card}</td>
<td>rule</td>
</tr>
</tbody>
</table>

Table 6.5: Paths to constrain the `card` property
6.3.3 Recursed paths

The paths item_identifier, qual_identifier, att_identifier, and mod_identifier can be continued down into the CE Instance, and thus anything described prior in this chapter can be appended on to the identifier.
Chapter 7

Clinical Element Kind

Clinical Element Constraint Types (CETypes) may be classified into several categories. This classification is identified by using the property *kind* in a Clinical Element Constraint Definition. Currently there are 6 possible values for the property.

Table 7.1: The values of the property *kind*

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>statement</td>
<td>The definition defines a complete assertion</td>
</tr>
<tr>
<td>panel</td>
<td>The definition defines a collection of complete assertions</td>
</tr>
<tr>
<td>component</td>
<td>The definition defines a portion of an assertion, qualifier, or attribution</td>
</tr>
<tr>
<td>attribution</td>
<td>The definition defines an attribution</td>
</tr>
<tr>
<td>modifier</td>
<td>The definition defines a modifier</td>
</tr>
<tr>
<td>noninstantiable</td>
<td>An incomplete definition that must be further defined to become a statement, panel, or component.</td>
</tr>
</tbody>
</table>

7.1 Statement

A Statement is a complete assertion about a particular aspect, characteristic or condition of a patient. A statement can be thought of as a complete sentence, such as “The patient John Doe had a Hematocrit of 38 percent on June 1, 2007”. When a Clinical Element Constraint Model is designed to model a statement, the
kind property is set with a value of “statement”. Since a statement is a complete assertion, it can stand on its own in a patient’s medical record. Because of this, any data instances which conform to a CEType designated as a statement, are storable on their own in the electronic patient record.

The individual parts of the statement, including modifiers and qualifiers, which also conform to their own CETypes can not exist on their own. This is because these individual parts that make up the statement are not meaningful by themselves out of context of the Statement. One can think of a Statement as a complete sentence.

Two types of statements exist, which are simple statements and compound statements. The basic difference between the two is that a simple statement contains a single datatype value, and a compound statement is made up of a collection of dependent individual values.

Figure 7.1: A Simple Statement vs. a Compound Statement

7.1.1 Simple Statement

A Simple Statement is a statement whose meaning is conveyed by a single clinical value, with associated modifiers and qualifiers. In The Clinical Element Model, a Simple Statement has the value choice of data rather than items. This means that the value is an HL7 like data type, such as a PQ or a CWE. An example of a simple statement is a hematocrit lab result.
7.1.2 Compound Statement

A Compound Statement is a statement whose meaning is conveyed by multiple clinical values, with associated modifiers and qualifiers. The meaning of the Compound Statement is dependent on a set of elements with values being interpreted together within the context of the collection. In The Clinical Element Model, a Compound Statement has the value choice of items rather than data. Each item within a compound statement must have it’s own kind property with a value of “component”, as they can not be statements or panels, because they can never exist on their own. An example of a compound statement is a pharmacy order.

7.2 Panel

A Panel represents a common grouping of clinical observations. It is a collection of statements or other panels that could each exist independently outside the panel. A Chemistry 7 lab result is an example of a common lab panel. A Chemistry 7 contains statements representing Serum Sodium, Serum Choloride, and other measurements, which could each exist independently of the enclosing panel. If the goal is to build a CEType to represent a panel, the kind property is set with a value of “panel”. When a CEType is designated as “panel”, then all it’s items must have their kind with a value of “statement” or “panel”.

Figure 7.2: A Panel vs. a Compound Statement
7.3 Component

A Component is CE Constraint Type that is only used within another Constraint Type and on its own does not constitute a complete statement. A data instance which conforms to a Clinical Element Constraint Type designated as a component, cannot be persisted alone in the electronic patient record. Instead, the data instance must only be persisted as an internal part of another data instance which conforms to a statement or panel. Examples of components include dates, times, and measuring devices. Components can be used as qualifiers or as items in a compound statement or compound component. Thus it is easy to see, it would nonsensical to store a measuring device such as a blood pressure cuff in a patient record outside the context of a blood pressure statement.

7.3.1 Simple Component

A Simple Component is a component whose meaning is conveyed by a single clinical value, with associated modifiers and qualifiers. In The Clinical Element Model, a Simple Component has the value choice of data rather than items. This means that the value is an HL7 like data type, such as a PQ or a CWE. An example of a simple component is a Status.

7.3.2 Compound Component

A Compound component is a component whose meaning is conveyed by multiple clinical values, with associated modifiers and qualifiers. The meaning of the Compound Component is dependent on a set of elements with values being interpreted together within the context of the collection. In The Clinical Element Model, a Compound Component has the value choice of items rather than data. Each item within a compound component must have its own kind property with a value of “component”, as they can not be statements or panels, because they can never exist on their own. An example of a compound component is SpecimenDescription.

7.4 Noninstantiable

A Noninstantiable Clinical Element Constraint Type can not be instantiated with patient data even within a statement or panel. It is only used as a modeling tool as a parent type from which subtypes can be created. It is a skeleton type that does not contain enough information (items, modifiers, qualifiers, etc.) to be instantiated.
7.5 Modifier

A Modifier is similar to a component, except that they can only be used in mods within the Clinical Element Instance Model. In the past we did use component for CETypes destined to be used in mods, but this gives our compiler an extra check to make sure modelers aren’t making mistakes. To designate a CEType as a modifier, set the kind property to “modifier”.

7.6 Attribution

An Attribution is similar to a component, except that they can only be used inatts within the Clinical Element Instance Model. In the past we did use component for CETypes destined to be used in atts, but this gives our compiler an extra check to make sure modelers aren’t making mistakes. To designate a CEType as an attribution, set the kind property to “attribution”.
Chapter 8

Modifiers and Qualifiers

8.1 Introduction

As previously mentioned, Modifiers and Qualifiers both alter the meaning of the data in the Clinical Element. Modifiers do this to such an extent, that they actually significantly change the meaning of the data, and hence we say they “modify” the data. Qualifiers may slightly change the meaning of the data, but to a much lesser extent, and thus we say they “qualify” the data.

Let’s look at the difference between a modifier and qualifier with an example. Say we have a Clinical Element Constraint Type which represents the laboratory test for Protein C, which is a protein involved in coagulation. An example of a qualifier for this Protein C model could be the technician who drew the the blood from the patient. The qualifier adds extra information but doesn’t change the meaning of the Protein C model in relation to the Patient. An example of a modifier would be the subject whose blood was tested. It may seem that the subject is always the patient, but this is not the case. A patient may have relatives that are tested for Protein C and these results are stored in this patients record. In this case, the subject would identify the relative.

8.2 Modifiers

Modifiers are themselves Clinical Element nested within a Clinical Element Instance. Modifiers are considered dangerous, because they significantly change the meaning of the Clinical Element Instance. The data in a Clinical Element Instance with a modifier can not even be considered without also considering the impact any Modifiers have on that data. The most easily understood modifiers are those that cause negation.
8.2. MODIFIERS

8.2.1 Negation or Certainty Modifier

Originally the model contained a modifier for negation that was called Negation. In further review of this issue, it was determined there are really two shades of negation. These two shades of Negation are called CertaintyOfExistance and CertaintyOfOccurance. Both of these are children of a supertype called Certainty.

**CertaintyOfExistance.** *The degree of certainty that what is in data is true. The range of values are “No, Probably Not, Maybe, Mightbe, Probably, Affirmative”*

**CertaintyOfOccurance.** *The degree of certainty that a procedure or action was performed. The range of values are “Absolutely Not, Unlikely, May have, Mightbe, Absolutely has”* ¹

The absence of certainty defaults to Absolutely Certain. The Negation modifiers are only allowed in leaf node Clinical Element Instances. Leaf Node Clinical Elements are those that have an HL7 datatype in contrast to items as the value choice.²

8.2.2 Subject Modifier

Another important modifier is called Subject. This modifier indicates who the data in the Clinical Element is about. In the absence of a Subject Modifier, Subject defaults to self. This modifier allows us to add Clinical Elements in a patients record that contain data about family members, household members, or donors.³

---

¹These values need to be made consistent between the two  
²Need image  
³Need image
Chapter 9

Clinical Element Attribution

Attribution is used to define an action along with the details of that action, including the who, what, where, when, and why the action was performed. Attribution is modeled as a noninstantiable CEType and thus is never used directly, but instead, Attribution is subtyped for further use. Some examples of subtypes of Attribution are Observed, Verified, Created, Reported. From these subtype names, it can be seen that our convention is to name subtypes of Attribution with a verb in the past tense indicating the action that is intended.

9.1 Modeling

Attribution is modeled with a coded value in the data section, which is constrained in subtypes to always have a value of the action performed. The kind property is declared as “noninstantiable” which means that no instances can be created using Attribution itself. When subtypes are created the kind property is redeclared as “attribution”, and these subtypes can then be used within instantiable CETypes.

---

1Our Attribution grew from our need to document actions, and at the time we included the reason for the action. Stan believes that attributions according to Rector may not include reason. Our current decision is to keep the reason.

2Issue: This is a big problem. We have overloaded the kind property. We would like to say Attribution is noninstantiable and attribution. We need to add another property which denotes instantiable which is separate from kind.
9.2. SUBTYPES

Table 9.1: The qualifiers of Attribution

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>startTime</td>
<td>When the action started</td>
</tr>
<tr>
<td>endTime</td>
<td>When the action ended</td>
</tr>
<tr>
<td>participant</td>
<td>A participant in the action</td>
</tr>
<tr>
<td>patientLocation</td>
<td>The patient’s location during the action</td>
</tr>
<tr>
<td>providerLocation</td>
<td>The provider’s location during the action</td>
</tr>
<tr>
<td>reason</td>
<td>The reason for the action</td>
</tr>
</tbody>
</table>

9.2 Subtypes

To actually use an Attribution, it must be subtyped. Examples of subtypes include Observed, Verified, Created, Reported. When Attribution is subtyped, the \textit{data.cwe} property is constrained to a specific action or a domain of actions. In figure 9.1 is an example of an instance conforming to the \textbf{Observed} subtype which is declared above.
Figure 9.1: A CE Instance conforming to the Attribution subtype called Observed.
Chapter 10

Semantic Links

Once clinical element instances are stored in the patient information system, we need the ability to link one instance with another instance and then assign a relationship type to this link. This link is called a semantic link because the relationship is a concept with a computable definition.

An example of a need for a semantic link would be linking a Throat Culture instance that was positive for Streptococcus pyogenes to the resulting Order for Penicillin. Clinical element instances are linked via their instance identifiers, by a relationship code such as “result in” or “caused by”.

10.1 Versus a Qualifier

A qualifier differs from a semantic link, in that a qualifier is designed as an internal part of the source or target model, and the instance data for that qualifier is stored as an internal part of the instance. On the other hand, a semantic link is physically separate from the source and target model, and no change to the source and target instance is needed to create a link between them.

In almost every case where a semantic link could be used, it is possible that qualifiers could have been modeled instead, and the same goes for the reverse case. For example, in Figure 10.1, ThroatCulture could have been modeled with a qualifier called ResultingMedOrder, and the Order could have been modeled with a qualifier to indicate reason it was ordered.

Because of this, the modeler needs to keep the aspects of each approach in mind:

- A qualifier can not be a statement, but a semantic link can link to a statement.
- Semantic links are by their nature bi-directional.
Figure 10.1: A Throat Culture Instance positive for Strep pyogenes is semantically linked to an Order for Penicillin

- Qualifiers don’t really work well for joining two existing instances.
- Semantic links don’t require a change to stored data instances.
- A semantic link should never change the meaning of the source or target instance, thus it should always be safe to ignore semantic links.

10.1.1 Tightly versus Loosely Coupled

We have decided not to make this distinction between tightly coupled and loosely coupled semantic links for the time being, but it is described here for discussion.

When there is a debate as to whether one should choose a qualifier or semantic link, and a semantic line is chosen, the resulting semantic link is usually more important to the meaning of an instance than other semantic links. One possible solution to signify this difference is the notion that a semantic link can be coupled as either “loose” or “tight”. This could be a binary value, but it would be also possible to use a scale, such as a range from 0 to 10. An advantage of this, is that queries could automatically bring back important semantically linked instances.

To limit the complexity of semantic links, we have decided not to make this distinction for the time being. Instead we leave the responsibility in the hands of the query. Important semantic links can be included in queries. A benefit of this is
that all semantic links are treated identically, and it is always known that you must query a semantic link to retrieve it.

10.2 Target as Concept

Discussion is still ongoing, and we have not decided whether we will allow semantic links to concepts.

Recently we have discussed the possibility that the target of a semantic link could actually be a coded concept as opposed to only another instance. If we allowed this, a semantic link with the target of a coded concept would actually replace our need for labels, or classifications as they were originally called.

10.3 Modeling

We could allow any semantic links to be applied to a source instance, but the danger is that users could create nonsensical semantic links between instances. So instead we have decided to put semantic link constraints into our constraint language.

Below are the areas we need to identify:

Table 10.1: The things we need to identify in a semantic link constraint

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target instance properties</td>
<td>type, key, data value, etc</td>
</tr>
<tr>
<td>Relationship</td>
<td>This is the forward relationship between the source and target</td>
</tr>
<tr>
<td>cardinality</td>
<td>The number of semantic links allowed</td>
</tr>
</tbody>
</table>

Table 10.2: Other things we may need to identify in a semantic link constraint at some point in the future.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target concept</td>
<td>If the target is a concept</td>
</tr>
<tr>
<td>Strength</td>
<td>This is the coupling strength which is either “loose” or “tight”</td>
</tr>
</tbody>
</table>
### 10.4. INVERSE RELATIONSHIPS

#### CHAPTER 10. SEMANTIC LINKS

10.3.1 Syntax

In Figure 10.2, is an example of the CEML syntax for a semantic link constraint. Each link is given a name, similar to how qualifiers, modifiers, and items are given a name. The allowed relationship is defined in the `relation` property with a code from the terminology server. At the moment, we do not have support for a domain of allowed relationships, and each relationship must be specifically defined. The number of allowed links is defined with the `card` property. Individual constraint values within the target instance are specified with the `target` element, which contains a `path` and `value` property analogous to the `constraint` element.

Figure 10.2: The CEML Syntax to constrain allowable semantic links

```xml
<link name="myLinkName" relation="resultedFrom_ECID" card="0-1">
  <target path="type.code" value="PAIN_MODEL_ECID"/>
  <target path="key.code" value="Assertion_KEY_ECID"/>
</link>
```

10.4 Inverse Relationships

Although a semantic link is only the forward relationship, it would be useful for the implementation to have both a forward and reverse relationship. We have decided to have this work automatically. In the terminology server, every relationship code will have an inverse relationship. In this way, the modeler only worries about the forward relationship.

---

1Issue: We could define relation as a CNE, so that we can constrain to a domain of relations.
Chapter 11

Scope

This chapter deals with the issue of whether instantiated qualifiers, attributions, and modifiers in panels apply to the child statements and panels within that panel instance. An example of this issue can be seen in Figure 11.1, which shows an instance conforming to a BloodPressure Panel. This panel contains the two statements SystolicBP and DiastolicBP, as well as a qualifier for BodyLocation at the panel level. The issue at hand occurs when one examines SystolicBP or DiastolicBP outside the context of this Panel, and the question is whether BodyLocation is applicable to the individual statements.

There are three scenarios we consider.

override In the “override” case, the instantiated qualifier in the panel will apply to the child statement. But, if the child statement already has an instantiated qualifier of this type, then this local qualifier overrides the qualifier from the panel.

local In the “local” case, the instantiated qualifier in the panel will not apply to the child statement.

additive In the “additive” case, the instantiated qualifier in the panel will apply to the child statement. If the child statement already has an instantiated qualifier of this type, then the qualifier from panel will apply along side the existing qualifier.

11.1 Modifiers

The default scope for modifiers is “override”. At the moment, the only two modifiers we have defined are Subject and Negation. Negation would never be used
Figure 11.1: A CE Instance of BloodPressure with a qualifier at the panel level.

in a panel due to modeling style rules, so it is not a use case. Subject is used in panels, and we would always want it to follow the scope rules of “override”. So in the end, there is no use here for anything but “override”.

11.2 Attributions

The default scope for attributions is “override”.¹

11.3 Qualifiers

The default scope for qualifiers is “override”.²

¹We are still discussing attributions and scope rules, and specifically if whether there are use cases for “additive” or “local”.

²We are still discussing qualifiers and scope rules, and specifically if whether there are use cases for “additive” or “local”.
11.4 Modeling

We are still discussing the syntax to describe scope. There are two basic approaches, one which is deterministic by CEType, and the other which is non-deterministic, and we may use either or both. In the deterministic approach, the scope property is declared at the CEType level, so that every time the CEType is used as a qualifier, modifier, or attribution, the CEType has that declared scope. In the non-deterministic approach, the scope property is not declared at the CEType level, so it is required to declare the scope when the CEType is used as a qualifier, modifier, or attribution. In either case, the value “override” would be the default if the scope was not declared.

11.4.1 Deterministic

```xml
<cetype name="MyType" kind="component" scope="additive"/>
```

11.4.2 Non-deterministic

Using qual tag

```xml
<qual name="myQualifier" type="MyQualifier" card="0-1" scope="additive"/>
```

Using constraint tag

```xml
<constraint path="qual.myQualifier.scope" value="additive"/>
```
Chapter 12

Clinical Element Root

This is a deprecated construct, but we describe this for historical reasons.

The Clinical Element of type Root was the default base class for all clinical element constraint types. When a \texttt{cetype} is declared, it can be declared de novo, or the \texttt{base} property can be used to declare a parent \texttt{CEType}. If no \texttt{base} is declared, the \texttt{CEType} was given a \texttt{base} of Root.

### 12.1 History

Initially Root had 6 qualifiers, but over time we realized we didn’t want most of these qualifiers in every \texttt{CEType}. In the end, only one qualifier remained in Root, and this was \texttt{Comment}.

```xml
<cetype name="Root" kind="noninstantiable">
  <qual name="comment" type="Comment" card="0-M"/>
  <constraint path="qual.comment.context" value="additive"/>
</cetype>
```

The original 6 qualifiers we had in Root were \texttt{Subject}, \texttt{Classification}, \texttt{Actuality}, \texttt{Negation}, \texttt{Aggregate}, and \texttt{Comment}.

### 12.2 Implementation

Engineering needs have required us to have the qualifier \texttt{Comment} moved out of the \texttt{Qualifiers} section, and be made a sibling of \texttt{Qualifiers} and \texttt{Modifiers}. Because

---

\textsuperscript{1}Is now deprecated - Please read the Implementation section of this chapter
Table 12.1: Reason for removal from Root

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subject</td>
<td>Only needed in Statements and Panels</td>
</tr>
<tr>
<td>Classification</td>
<td>Renamed to Label, which was subsequently removed because this is now handled with Semantic Links</td>
</tr>
<tr>
<td>Actuality</td>
<td>Not being used at the moment, so temporarily removed</td>
</tr>
<tr>
<td>Negation</td>
<td>Decision to only add this as needed, because it's use doesn't make sense for all types, especially panels</td>
</tr>
<tr>
<td>Aggregate</td>
<td>Need to discuss, as this may still belong here, or this could become part of the base model.</td>
</tr>
<tr>
<td>Comment</td>
<td>Remains</td>
</tr>
</tbody>
</table>

of this, we no longer have anything in Root, so de novo Clinical Element Constraint types inherit no constraints at the moment.
Chapter 13

CEML

13.1 Introduction

Our implementation of the Abstract Constraint Model is called Clinical Element Modeling Language or CEML. This ITS is defined using an XML syntax with specific elements and attributes which conform one to one to the constraint model. As of this writing we have two forms of CEML, one that strictly follows the Abstract Constraint Model, and another form for modelers that has some shortcuts or macros to ease the authoring purposes. This chapter will deal with only the strict CEML implementation of the Abstract Constraint Model.

A simple example of a Clinical Element Constraint Type defined in Strict CEML can be seen in Figure 13.1.

Figure 13.1: Strict CEML Example of a Constraint Type called MyType

```xml
<ceml>
  <cetype name="MyType" base="MyBaseType" kind="statement">
    <constraint path="key.code" value="MyModelRealWorldConcept_CODE"/>
    <constraint path="data.type" value="pq"/>
    <constraint path="qual.myQualifier" value="MyQualifier"/>
    <constraint path="qual.myQualifier.card" value="0-1"/>
  </cetype>
</ceml>
```
13.2  CEM versus CETypes

Frequently people refer to assets such as the VitalSignsPanel Clinical Element Model, or the VitalSignsPanel CEM. What they are really referring to, is the constraint type for VitalSignsPanel instance data. The collection of constraints is called a CETYPE.

13.3  Paths

The allowable paths for the constraints are defined by the Abstract Instance Model, the properties of the Clinical Element Datatypes, and the Abstract Constraint Model.
Chapter 14

CEML Authoring Syntax

Clinical Element Instances are constrained using the Clinical Element Abstract Constraint Model. Our implementation of the Abstract Constraint Model is called Clinical Element Modeling Language or CEML. This implementation is defined using an XML syntax with specific elements and attributes which will be defined in this chapter. As of this writing we have two forms of CEML, one for authoring purposes called Authoring CEML, and another called Strict CEML which strictly follows the Abstract Constraint Model. This chapter will deal with only the Authoring CEML Syntax. It should be understood that everything that is allowed in Strict CEML is allowed in Authoring CEML, but Authoring CEML adds additional shortcuts or macros, which can replace some of the constraint statements of Strict CEML.

A simple example of a Clinical Element Constraint Type defined in Authoring CEML can be seen in Figure 14.1.

Figure 14.1: A Simple Authoring CEML example defining the constraints of a Clinical Element Type named **MyType**

```xml
<ceml>
  <cetype name="MyType" base="MyBaseType" kind="statement">
    <key code="MyModelRealWorldConcept_CODE"/>
    <data type="pq"/>
    <qual name="myQualifier" type="MyQualifier" card="0-1"/>
  </cetype>
</ceml>
```
14.1 Root Element <ceml>

The <ceml> element is the root element for a CEML authored definition.\(^1\) It has no attributes, and the only allowable elements are one header element and then the one cetype element.\(^2\) The filename that contains this CEML definition should match the name attribute of the contained cetype element.

![Figure 14.2: The backbone of a CEML definition. This definition would be stored in a file named MyType.xml](image)

```xml
<ceml>
  <cetype name="MyType">
    ...
  </cETYPE>
</ceml>
```

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Cardinality</th>
</tr>
</thead>
<tbody>
<tr>
<td>header</td>
<td>element</td>
<td>one</td>
</tr>
<tr>
<td>cetype</td>
<td>element</td>
<td>one(^3)</td>
</tr>
</tbody>
</table>

Table 14.1: Properties of <ceml>

14.2 <header>

The header element is used to store authoring information regarding the authoring lifecycle of the CEType. Initially this was a temporary solution until we had an authoring database server implemented. But currently, we have no plans to remove header. Recently, a few non-authoring lifecycle content has made its way into header. These include group and context. We should consider moving these into CEType.

\(^1\)In the past we called the strict form compiled ceml which is no longer valid. We need new names and tags.

\(^2\)In the past, we have allowed one to many cetype elements within the ceml tag during the authoring process, but when finalized and definitions are published to the official definition store, then we required that each ceml element contain one and only one cetype.

\(^3\)During the initial authoring phase, it was acceptable to have many cetype elements within ceml, and the CEDAR authoring tool allows this functionality. We should remove this as an option.
14.3 <cetype>

The <cetype> element is used to define the constraints of a specific Clinical Element Type. This element has 3 attributes; name, base, and kind. The name attribute is the required name of the Clinical Element being defined. This name must be unique within the context of all CETypes, and is eventually submitted to the vocabulary server to receive a unique concept code. The base attribute is an optional parent CEType from which this CEType will inherit. If base is absent, then no constraints are inherited. The required kind attribute declares whether this CEType is a statement, panel, component, attribution, modifier, or is noninstantiable.

Figure 14.3: A possible example of a Systolic Blood Pressure CEType definition inheriting the constraints of Observation

```xml
<ceml>
  <cetype name="MySystolicBP" base="Observation" kind="statement">
    ...
  </cetype>
</ceml>
```

Table 14.2: CEML Element Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Cardinality</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>attribute</td>
<td>one</td>
</tr>
<tr>
<td>base</td>
<td>attribute</td>
<td>zero to one</td>
</tr>
<tr>
<td>kind</td>
<td>attribute</td>
<td>one</td>
</tr>
<tr>
<td>key</td>
<td>element</td>
<td>zero to one</td>
</tr>
<tr>
<td>data</td>
<td>element</td>
<td>zero to one</td>
</tr>
<tr>
<td>qual</td>
<td>element</td>
<td>zero to many</td>
</tr>
<tr>
<td>mod</td>
<td>element</td>
<td>zero to many</td>
</tr>
<tr>
<td>att</td>
<td>element</td>
<td>zero to many</td>
</tr>
<tr>
<td>item</td>
<td>element</td>
<td>zero to many</td>
</tr>
<tr>
<td>constraint</td>
<td>element</td>
<td>zero to many</td>
</tr>
<tr>
<td>link</td>
<td>element</td>
<td>zero to many</td>
</tr>
</tbody>
</table>

4In the past, an absent base would inherit from a built in CEType call Root. See the chapter titled Clinical Element Root for some historical information regarding this.
14.3.1 name attribute of <cetype>

The name attribute represents the unique textual identifier for the cetype that is being modeled. Some examples include VitalSignsPanel, SystolicBloodPressure, and AbnormalFlag.

14.3.2 base attribute of <cetype>

The base attribute identifies another cetype which acts as the supertype for the current cetype being defined. The cetype being defined inherits all parts of the base but may override any part by restriction.

14.3.3 kind attribute of <cetype>

The kind attribute represents the function of the cetype and affects the allowable persistence of instances in the datastore.

Table 14.3: Allowable values for <cetype> kind attribute

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>statement</td>
<td>A complete stand alone medical sentence</td>
</tr>
<tr>
<td>panel</td>
<td>A collection of medical sentences</td>
</tr>
<tr>
<td>component</td>
<td>Part of a medical sentence</td>
</tr>
<tr>
<td>modifier</td>
<td>Modifying part of a medical sentence</td>
</tr>
<tr>
<td>attribution</td>
<td>Part of a medical sentence describing an Action</td>
</tr>
<tr>
<td>noninstantiable</td>
<td>A preliminary definition which must be subtyped for use</td>
</tr>
</tbody>
</table>

14.4 <key>

The key element is a shortcut used to constrain the key of the CE Instance. One should remember that according to the Abstract Instance Model, key is defined as a CWE, so all the properties of CWE are available for constraint. The key element provides a shortcut to the modeler for two of the possible constraint paths, which are key.code and key.domain. Examples of these can be seen in Figures 14.4 and 14.5.
14.5 <data>

The **data** element is a shortcut used to constrain the *data* of the CE Instance. According to the Abstract Instance Model, *data* can contain any of our datatypes, and a constraint is used to limit this to one or more datatypes. The **data** element provides a shortcut to the modeler for the constraint path *data.type*. This shortcut can be seen in Figure 14.6. In addition, if the *data.type* is constrained to a **CWE**, then the *data.cwe.domain* constraint can also be collapsed into this shortcut. An example of this can be seen in Figure 14.7.

Figure 14.6: *data.type* constraint shortcut

```xml
<constraint path="data.type" value="pq"/>
<data type="pq"/>
```

14.6 <qual>, <item>, <att>, and <mod>

The syntax for constraining quals, items, atts, and mods all use the identical shortcut syntax structure, but use the **qual**, **item**, **att**, and **mod** tag respectively. Here we describe how to define **qual**, but the reader should realize that **item**, **att**, and **mod** follow the same pattern.
### Table 14.4: Properties of `<key>`

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Cardinality</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>attribute</td>
<td>choice of one</td>
</tr>
<tr>
<td>domain</td>
<td>attribute</td>
<td>choice of one</td>
</tr>
</tbody>
</table>

### Table 14.5: Properties of `<data>`

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Cardinality</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>attribute</td>
<td>one</td>
</tr>
<tr>
<td>domain</td>
<td>attribute</td>
<td>zero to one</td>
</tr>
</tbody>
</table>

### Table 14.6: Allowable values for `<data>` type

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cwe</td>
<td>Coded With Exceptions</td>
</tr>
<tr>
<td>cne</td>
<td>Coded No Exceptions</td>
</tr>
<tr>
<td>co</td>
<td>Coded Ordinal</td>
</tr>
<tr>
<td>pq</td>
<td>Physical Quantity</td>
</tr>
<tr>
<td>ivlpq</td>
<td>Interval Physical Quantity</td>
</tr>
<tr>
<td>rtopq</td>
<td>Ratio Physical Quantity</td>
</tr>
<tr>
<td>st</td>
<td>String</td>
</tr>
<tr>
<td>ed</td>
<td>Encapsulated Data</td>
</tr>
<tr>
<td>ii</td>
<td>Instance Identifier</td>
</tr>
<tr>
<td>int</td>
<td>Integer</td>
</tr>
<tr>
<td>real</td>
<td>Real Number</td>
</tr>
<tr>
<td>ts</td>
<td>Coded Simple</td>
</tr>
</tbody>
</table>
The **qual** element is the shortcut for the constraints of a qualifier. It contains 3 attributes, `name`, `type`, and `card`, as can be seen in Table 14.7. The `name` attribute is a unique textual identifier for this qualifier to be used in paths, and it is a shortcut for the path `qual.identifier`. The `type` attribute identifies the referenced `cetype` for this qualifier, and is a shortcut for the path `qual.identifier.type`. The `card` attribute specifies the cardinality of this qualifier, and is the shortcut for the path `qual.identifier.card`. An example of this shortcut can be seen in Figure 14.8.

**Table 14.7: Properties of `<qual>`**

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Cardinality</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>attribute</td>
<td>one</td>
</tr>
<tr>
<td>type</td>
<td>attribute</td>
<td>one</td>
</tr>
<tr>
<td>card</td>
<td>attribute</td>
<td>one</td>
</tr>
</tbody>
</table>

**14.6.1 name attribute of `<qual>`**

The `name` attribute represents the unique textual identifier for the qualifier. It can be any text string, but usually it is formed from the value in the `type` attribute by lower casing the first letter.
14.7. CONSTRAINT

14.6.2 *type* attribute of <qual>

The *type* attribute identifies the referenced *CEType* for this qualifier.

14.6.3 *card* attribute of <qual>

The *card* attribute specifies the cardinality for this qualifier. The allowable values for *card* can be seen in Table 14.8.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Zero</td>
</tr>
<tr>
<td>0-1</td>
<td>Zero to One</td>
</tr>
<tr>
<td>0-M</td>
<td>Zero to Many</td>
</tr>
<tr>
<td>1</td>
<td>Exactly One</td>
</tr>
<tr>
<td>1-M</td>
<td>One to Many</td>
</tr>
</tbody>
</table>

Table 14.8: Allowable values for *card* attribute in <qual>

14.7 constraint

Regular constraints as defined in Strict CEML are still allowed. It must be remembered that Authoring CEML just contains additional shortcuts in addition to what is defined by the Abstract Constraint Model. For example, instead of using the *key* tag to constrain the key, it is still possible to use a plain old constraint, as seen in Figure 14.9.

Figure 14.9: An example of using a constraint instead of the key tag

```xml
<ceml>
  <cetype name="MyType">
    ...
    <constraint path="key.code" value="MyType_ECID"/>
  </cetype>
</ceml>
```
14.8 absence

Many times we want default values to be indicated when a qualifier or modifier is not present in a CEMlInstance. For example, the modifier Subject defaults to a coded value of “Patient” when it is not present. To represent this, there is an absence tag with the properties of path and value (Figure 14.10).

Figure 14.10: An example of using an absence tag

```xml
<ceml>
   <cetype name="Subject">
      ...
      <absence path="data.cwe.code" value="Patient_ECID"/>
   </cetype>
</ceml>
```

14.9 link

In Figure 14.11, is an example of the CEMl syntax for a semantic link constraint. The link tag is used to represent the allowable semantic links. Each link is given a name, similar to how qualifiers, modifiers, and items are given a name. The allowed relationship is defined in the relation property with a code from the terminology server. At the moment, we do not have support for a domain of allowed relationships, and each relationship must be specifically defined. The number of allowed links is defined with the card property. Individual constraint values within the target instance are specified with the target element, which contains a path and value property analogous to the constraint element.

---

5Issue: We need to discuss this, I don’t know if engineering is aware of this. This needs to be explained in the Abstract Constraint Model and Strict CEMl as well as here.

6Issue: Ramifications to the Abstract Instance and Constraint Model needs to be Explored
Figure 14.11: The CEML Syntax to constrain allowable semantic links

```xml
<link name="myLinkName" relation="resultedFrom_ECID" card="0-1">
  <target path="type.code" value="PAIN_MODEL_ECID"/>
  <target path="key.code" value="Assertion_KEY_ECID"/>
</link>
```
Chapter 15

Data Choice

Previously we haven’t allowed choices for data types, and we forced the modeler to create two separate constraint types, one with each of the data types required. But we have reversed this decision, and we have introduced a syntax to allow data type choices within a single CETYPE. For example, this allows a CETYPE to contain a choice between a PQ and a CWE datatype. The new syntax requires no change to the CEML Schema, as the only change is in the value of the type property within data.

In order to allow a choice, one must simply list each data type of the choice in an alphabetical comma delimited list, with no spaces. A list of our currently used choices is in Table 15.1

Table 15.1: The Current List of Datatype Choices

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cne,st</td>
<td>Zero</td>
</tr>
<tr>
<td>cwe,pq,st</td>
<td>Zero to One</td>
</tr>
<tr>
<td>cwe,real</td>
<td>Zero to Many</td>
</tr>
<tr>
<td>cwe,st</td>
<td>Exactly One</td>
</tr>
</tbody>
</table>

Here is a previous Authoring CEML constraint example for a CWE datatype.

<data type="cwe"/>

Below is a new Authoring CEML constraint example for a choice between CWE, and REAL, and note that “cwe” must be placed before “real” because of the alphabetical rule.
15.1 Alphabetical Rule

The alphabetical rule for choices of data types functions to create a unique string for each choice type, so it is easy for us to identify all CWE, REAL choices. Without the alphabetical rule, there would be 2 possible strings which represent this same choice. And if the choice contained 3 types, then we would have 6 possible strings.

15.2 Use Requirements

A datatype choice can only used when the follow two conditions are true...

1. All of the components of the model (qualifiers and modifiers) must be valid for a CE Instance that uses any of the possible data types in the choice.
2. The domains of those components are not constrained differently depending on the data type. If either of these rules does not apply, the data type choice should not be used.

If the above conditions are not both true, then instead, multiple CETypes should be created, each with their own datatype.

For example, say we have a CEType with a data choice between CWE and PQ, and this also has two qualifiers, qual1 and qual2. If qual1 were only valid for a CE Instance using CWE and qual2 was only valid for a CE Instance using PQ then this would fail the first condition. So instead of creating a CEType referencing both qual1 and qual2 with data constrained to a type of “cwe,pq”, two different CETypes should be created; one with data constrained to CWE and with a reference to qual1, and the other with data constrained to PQ and with a reference to qual2.

As another example, say qual1 is valid for both a CE Instance of type PQ and also for a CE Instance of type CWE. But, the data in qual1 needs to be constrained to “domain1” when a CE Instance uses PQ, and the data in qual1 needs to be constrained to “domain2” when a CE Instance uses CWE, then this would fail the second condition. So instead of creating a single CEType with data constrained to a type of “cwe,pq”, two different CETypes should be created; one with data constrained to CWE and with qual1’s data constrained to “domain1”, and the other with data constrained to PQ and with qual1’s data constrained to “domain2”.

70
Chapter 16

CEML Tutorial

To understand how to define a CETYPE with CEML it is probably easiest to just give a few simple examples and describe the example. We will be using the Authoring CEML syntax for this walk through. First let's start by describing a type for a Blood Pressure Panel in Figure 16.1. Remember that in a Clinical Element Instance, the value choice can contain either a list of child Clinical Element Instance nodes or an HL7 datatype. In this first example, we create a CETYPE that constrains the instance to a value choice of items which contains two children; one for systolic blood pressure and one for diastolic blood pressure.

```
<ceml>
  <header>
    ...
  </header>
  <ctetype name="BloodPressurePanel" kind="panel">
    <key code="BloodPressurePanelKey_CODE"/>
    <item name="systolicBloodPressure" type="SystolicBP" card="0-1"/>
    <item name="diastolicBloodPressure" type="DiastolicBP" card="0-1"/>
  </ctetype>
</ceml>
```

Figure 16.1: A Blood Pressure Panel with no Qualifiers

Let's examine the parts of this CETYPE definition in Figure 16.1. Every definition is defined using a ceml element as the root element. Inside of the ceml element, we have one header element and one ctetype element. We will not be discussing the header element in this chapter, so let's focus on the ctetype element.

The ctetype element is the basis of the Clinical Element Constraint Type. The
name attribute of cetype will act as a globally unique name for the cetype we are creating with this definition. This cetype has been declared a “panel” using the kind attribute. A list of the allowable attributes allowed in cetype can be seen in Figure 16.2.

- **name** - The name of the cetype being declared.
- **base** - The name of a parent cetype.
- **kind** - Identifies the function of this cetype.

Figure 16.2: Attributes of cetype

Next, inside the cetype, other constraints can be declared using various elements. In our example in Figure 16.1, there are two types of elements within the definition. The first is a key constraint and the second is two item constraints.

In the key element, the allowable key code for an instance can be restricted to a particular code, as it is here using the code attribute. An alternative to using the code attribute, is to use the domain attribute which restricts the allowable key code in an instance to be within a certain domain of codes. The attributes of the key element are listed in Figure 16.3.

- **code** - Restricts all instances of this type to have this key code.
- **domain** - Restricts all instances of this type to have a key code within this domain.

Figure 16.3: Attributes of the key constraint

Next in our BloodPressurePanel example in Figure 16.1, there are two item constraints. These item constraints declare the child Clinical Elements that are required or allowed in the BloodPressurePanel we are defining. Each of the item constraints has a name, type, and card attribute. The type attribute defines the referenced cetype to constrain the child, and the name attribute assigns a local name to this type. This is just like the normal variable-type assignments used in most programming languages. The card attribute is set to “0-1” for both of the examples, and this defines the number of each child type that can occur in the CE Instance. The attributes available in an item constraint are listed in Figure 16.4.

- **name** - The local name of this item.
- **type** - The Clinical Element type of this item.
- **card** - Restricts the number of occurrences for this item.

Figure 16.4: Attributes of the item constraint
16.1 Data

In the last example we saw a `cetype` defined where the value choice was two child Clinical Elements rather than an HL7 datatype. So in this next example seen in Figure 16.5 we define a Systolic Blood Pressure `cetype` that uses an HL7 datatype. In the `data` definition there is an attribute called `type` which is used to declare the HL7 datatype to be used. In this example, it has been declared to be an `HL7:PQ`.

```xml
<ctypename="SystolicBP" base="Observation" kind="statement">
  <key code="SystolicBPKey_CODE"/>
  <data type="pq"/>
</ctypename>
```

Figure 16.5: A Systolic Blood Pressure definition with no Qualifiers

16.2 Qualifiers

The examples in Figures 16.1 and 16.5 only declared constraints of the `key` and value choice (`data` or `items`). Next in Figure 16.6, we have added qualifier constraints pertinent to the systolic blood pressure. These `qual` constraints have the same attributes as the `item` constraints. See Figure 16.7.

```xml
<ctypename="SystolicBP" base="Observation" kind="statement">
  <key code="SystolicBPKey_CODE"/>
  <data type="pq"/>
  <qual name="measurementMethodOrDevice" type="MeasurementMethodOrDevice" card="0-1"/>
  <qual name="bodyPosition" type="BodyPosition" card="0-1"/>
  <qual name="intravascularBodySite" type="IntravascularBodySite" card="0-1"/>
  <qual name="breathingPhase" type="BreathingPhase" card="0-1"/>
</ctypename>
```

Figure 16.6: A Systolic Blood Pressure definition with Qualifiers

These constraints for qualifiers will allow the CE Instance to contain information about the measured systolic blood pressure, such as indicating the body position of the patient during the measurement, and what method or device was used to make the systolic blood pressure measurement.
16.3. MODIFIERS

In Figure 16.8, is an example of a modifier constraint pertinent to the systolic blood pressure or any clinical element. This mod constraint has the exact same attributes as the qual structure. The modifier in the example is called Subject, and is used to represent the subject of this systolic blood pressure. In the CE Instance, it would have a value of “self” to indicate that the measurement was taken on the patient.

```xml
<ctype name="SystolicBP" base="Observation" kind="statement">
  <key code="SystolicBPKey_CODE"/>
  <data type="pq"/>
  <qual name="measurementMethodOrDevice" type="MeasurementMethodOrDevice" card="0-1"/>
  ...
  <mod name="subject" type="Subject" card="0-1"/>
</ctype>
```

Figure 16.8: A Systolic Blood Pressure definition with a Modifier
Chapter 17

Understanding Instance Data

17.1 Introduction

We have discussed that CEML is used to author the Clinical Element Constraint Types which contain rules that constrain the values in a Clinical Element instance. But what is instance data? People outside of computer science seem to have a hard time understanding what we mean by instance data. Instead of a formal definition, it is easier to understand instance data by example.

In the following examples we will use an English sentence to state a definition or constraint, and follow this with examples of many instances each also represented as an English sentence.

**Constraint**  A blood pressure panel contains a Systolic Blood Pressure and a Diastolic Blood Pressure each with a numerical value and with units of mmHg. The values should never be negative. In addition, you can specify the patient’s body position.

**Instances**  

- 120/80 mmHg
- 142/101 millimeters Mercury
- 114/68 mmHg while patient was sitting
- 132/96 mmHg while patient was standing

From this example, it should be clear, that you have ONE constraint definition and you can then have an UNLIMITED number of instances which conform to the definition. Every time a blood pressure measurement is taken on a patient, a new instance is generated. These instances are then stored and make up a patient’s electronic medical record, available for later retrieval.
17.2 Constraints and Instances

There are many existing formalisms that allow one to state a constraint definition, and then create many instances that conform to this definition. For example, XML Schema can be used to declare a constraint definition and then many XML documents can be created as instances which conform to the XML Schema definition. ASN.1 source is also used to declare a definition and then instances exist as BER strings. Java source code is used to declare a definition or Class, and then many instances of that class can be created.

![Diagram showing formalisms to define constraints/rules for instances.]

Figure 17.1: Formalisms to define constraints/rules for Instances.

17.3 Serialization

Frequently, you will hear the term serialization used when reference is made to instances. When many instances are created that conform to a definition, these instances must be able to be used by computer software. There also must exist a way to send this instance over the network as well as a way to store it.

Taking XML as an example...

more to come.
Chapter 18

Glossary

18.1 Definitions

Authoring CEML See CEML

Clinical Element Abstract Instance Model The Clinical Element Abstract Instance Model defines a recursive structure that can hold patient instance data. It is defined abstractly, and thus this model must be implemented in an actual language such as XML or Java.

Clinical Element Abstract Constraint Model The Clinical Element Abstract Constraint Model is the model used to constrain and describe allowable instances of patient data. This is the abstract description of our constraint formalism, which is then actually implemented. We have an XML implementation which is called CEML or Clinical Element Modeling Language.

<cetype/> A CEML construct. The element which is used to define a Clinical Element Constraint Type, or CEType.

CE See Clinical Element. This should not be confused with HL7:CE which is a now deprecated HL7 version 3 datatype Coded With Equivalents.

CEM See Clinical Element Model.

CEML See Clinical Element Modeling Language.

CEO See Clinical Element Object.

CEType See Clinical Element Constraint Type
18.1. DEFINITIONS

Clinical Element  The terms which unfortunately is used rather loosely and incorrectly. The correct usage is that a Clinical Element is the recursive structure in the Abstract Instance Model, also called a Clinical Element Instance Node. It is NOT a Clinical Element Constraint Type.

Clinical Element Instance Node  A Clinical Element Instance Node is the recursive structure in the Abstract Instance Model. It consists of a key, a type, a value choice (data or items), modifiers, and qualifiers.

Clinical Element Constraint Type  This is a construct defined by the Abstract Constraint Model which is a collection of constraints used to validate an instance from the Abstract Instance Model. The definition of a particular Clinical Element Constraint Type such as a BloodPressurePanel. Instances of this BloodPressurePanel must conform to the constraints stated within this constraint definition.

Clinical Element Instance  An instance of patient data in a format that conforms to the Clinical Element Abstract Instance Model. This instance data can then be constrained by a Clinical Element Constraint Type. For example, an instance of patient data such as a hematocrit with a value of 38.9 stored in the EMR for John Doe, that conforms to the constraint type called LabObservationHematocrit.

Clinical Element Model (CEM)  Unfortunately, this term is erroneously used to refer to an individual Clinical Element Constraint Type (CEType).

The Clinical Element Model (The CEM)  The Clinical Element Model is a term used to denote the global modeling effort as a whole. It is the combination of the Abstract Instance Model and the Abstract Constraint Model.

Clinical Element Modeling Language (CEML)  This is our Implementation Technology Specification of the Abstract Constraint Model. It is an XML based syntax. We have 2 forms a Strict CEML and Authoring CEML. Strict CEML follows the Abstract Constraint Model constructs exactly. Authoring CEML includes the syntax of Strict CEML, but also adds shortcut or macro elements to make the definitions more succinct.

Clinical Element Object (CEO)  This is a programatic object that is used to manipulate Clinical Element Instance Data. It is analogous to an XML DOM.

Compound Statement  A Statement whose meaning is conveyed by multiple clinical values, with associated modifiers and qualifiers. The meaning of the Compound Statement is dependent on a set of elements with values being
interpreted together within the context of the collection. In The Clinical Element Model, A Compound Statement has the value choice of items rather than data. For instance, a pharmacy order is a compound statement. See Statement.

**Component**  A CE Cosntraint Type that is only used within another CE Constraint Type (as an item, qualifier or modifier). A CE Instance that conforms to a component CEType can NOT be stored in the patient EMR on it’s own, but only as an internal part of another instance. Examples of Component CETypes include Specimen, MethodAndDevice, BodyPosition, and Length.

**Data**  A Construct of the Abstract Instance Model which contains an HL7-derived data type that serves as the `value` of a Simple Statement.

**<item/>**  An Authoring CEML construct used to constrain Clinical Element Instance Nodes within the Items of a Clinical Element Instance.

**Key**  An property within a Clinical Element Instance that is an HL7:CWE. The key’s code links the Clinical Element instance to a real world coding system.

**<key/>**  An Authoring CEML construct used to constrain the Key of a Clinical Element Instance.

**Label**  DEPRECATED : These are now replaced by Semantic Links with a target of a Coded Concept.

**<mod/>**  An Authoring CEML construct used to constrain Clinical Element Instance Nodes within the Mods of a Clinical Element Instance.

**Modifier**  A Clinical Element Instance node which modifies the content of the Value Choice in the containing Clinical Element Instance. The extent of this modification is so great, that the value choice can never be considered independently without simultaneously considering the effect of the modifier on the value choice.

**noninstantiable**  noninstantiable is a potential value of the property “kind” in a Clinical Element Constraint Type that indicates that no patient data can be instantiated using this constraint type. Instead, this constraint type is used as a starting point to define other constraint types. A Noninstantiable Clinical Element Constraint Type is a type that does not contain enough information (items, modifiers, qualifiers, etc.) to be instantiated.
Panel Represents a common grouping of clinical observations. A chem7 lab result is an example of a common lab panel. A panel is a collection of statements that can exist independently. Synonyms used for panel include battery and collection.

Qualifier Clinical Element Instance node which give more information about the Value Choice in the containing Clinical Element Instance. The degree to which this qualification changes the meaning of the value choice varies, but it is never to the degree of a modifier. In medical informatics circles, some argue you can never even truly ignore a qualifier, so why make the distinction between a qualifier and a modifier.

<qual/> An Authoring CEML construct used to constrain Clinical Element Instance Nodes within within the Quals of a Clinical Element Instance.

Semantic Link Semantic Link is the construct that is used to establish a relationship between separate independent Clinical Element Instances. The semantic link specifies a coded relationship between CE instances; they are not used for static, a priori relationships.

For example, a semantic link may appropriately be used to express the relationship between a particular medication order for Tom and the reason for the order by reference to a specific problem in Tom’s problem list only that particular order is related to that particular problem.

An inappropriate use of a semantic link would be to express the relationship between a lab observation and a specimen all lab observations are related to specimens. This type of static relationship is best defined within the lab observation CE Constraint Type.

Simple Statement A Statement whose meaning is conveyed by a single clinical value, with associated modifiers and qualifiers. In The Clinical Element Model, A Simple Statement has the value choice of data rather than items. An example of a simple statement is a hematocrit lab result. See Statement.

Specializable CEType A Specializable Clinical Element Constraint Type does not specify one particular value to be used as its Key.code. Instead, the Constraint Types specifies a domain of permitted Key.codes.

For example, quantitative lab result is a Specializable CE Constraint Type, in which the Type is quantitative lab result, and the Key is constrained to the domain of specific quantitative labs like hct, serum sodium, etc.
Specific CEType  A Specific Clinical Element Constraint Type specifies a single code as the permissible value of its Key.code.

LabObservationHematocrit is an example of a Specific Clinical Element Constraint Type. Its parent type is the Specializable CEType LabObservationQuantitative. The LabObservationHematocrit CE’s Type is LabObservationHematocrit, and the only Key value allowed is Hct.

A Specializable Clinical Element Constraint Type may be used to validate patient data where no more specific model is available, but the ultimate goal is to have a Specific Clinical Element Constraint Type for everything stored in a patient record.

Statement  A complete assertion about a particular aspect, characteristic or condition of a patient. A statement contains a value choice (data or items), and may also contain modifiers and qualifiers. The parts of a Statement are not meaningful by themselves out of context of the Statement. There are two types of statements: simple statements and compound statements.

Strict CEML  See CEML
Chapter 19

Change Log

Changes to this document are listed below. It should be noted that we did not track changes initially so version Aug 15, 2008 is the beginning of change tracking.

19.1 August 15, 2008

- **Addition** - This Change Log is added
- **Addition** - Attribution chapter is added
- **Addition** - CE Root chapter is added
- **Addition** - Context Control chapter is added
- **Addition** - Data Choice chapter is added
- **Addition** - Semantic Links chapter is added

19.2 September 15, 2008

- **Update** - CE Kind Chapter update with new images
- **Update** - Semantic Link Chapter update with syntax
- **Update** - CEML Constraint Syntax Chapter added “att” element
- **Update** - Abstract Instance Model Chapter update for Attribution
- **Update** - CEML Author Syntax Chapter update for Attribution, semantic links
- **Update** - Context Control Chapter Changed to Scope
19.3 Nov 13, 2008

- **Update** - Complete Edit of entire document except Chapter One. Goal was to remove inconsistencies that have arisen in the document due to the evolving nature of the work.

19.4 Nov 14, 2008

- **Update** - Data Choice Chapter - Added rules for use.