ARCHETYPE MODEL

The *open*EHR Data Types Archetype Model

Editors:{T Beale, S Heard}[1], {D Kalra, D Lloyd}[2]

Revision: 0.6.2

Pages: 37

[1] Ocean Informatics Australia

[2] Centre for Health Informatics and Multi-professional Education, University College London

The *open*EHR foundation

is an independent, non-profit community, facilitating the creation and sharing of health records by consumers and clinicians via open-source, standards-based implementations.

email: info@openEHR.org **web**: http://www.openEHR.org

Copyright Notice

Amendment Record

Issue	Details	Who	Date
0.6.2	CR-000023. TERM_MAPPING.*match* should be coded/enumerated.	G Grieve	10 Feb 2004
0.6.1	CR-000041. Visually differentiate primitive types in openEHR documents. CR-000013. Rename key classes - rename COMPOUND to CLUSTER to conform with CEN 13606.	T Beale	04 Oct 2003
0.6	CR-000003, CR-000004 changes. Changed package naming, improved heading structures. (Formally validated).	T Beale	21 Mar 2003
0.5	Included DSTC documentation corrections. Corrected STATE class. (Formally validated)	R Shackel	25 Feb 2003
0.4	**Formally validated using ISE Eiffel 5.2**. Updated to conform to Data Types RM 1.7. Numerous small changes to most classes.	T Beale	20 Feb 2003
0.3	Rewritten to conform to Data Types RM 1.5.9	T Beale	10 Nov 2002
0.2	Added external_id constraint classes, checked text, quantity and date/time constraint classes.	T Beale	14 Aug 2002
0.1	Initial Writing	T Beale	16 Jul 2002

Acknowledgements

The work reported in this paper has been funded by a number of organisations, including The University College, London; Ocean Informatics Pty Ltd, Australia.

Table of Contents

1 Introduction 7
1.1 Purpose 7
1.2 Related Documents 7
1.3 Status 7
1.4 Peer review 7
2 Overview 9
2.1 Package Structure 9
3 AM.DATA_TYPES.BASIC Package 10
3.1 Class Descriptions 10
3.1.1 C_DATA_VALUE Class 10
3.1.2 C_DV_BOOLEAN Class 11
3.1.3 C_DV_STATE Class 11
3.1.4 STATE_MACHINE Class 12
3.1.5 STATE Class 12
3.1.6 TRANSITION Class 12
4 AM.DATA_TYPES.TEXT Package 14
4.1 Overview 14
4.2 Constraints on Plain Text Items 14
4.3 Constraints on Coded Text Items 15
4.3.1 Constraints on Names 15
4.3.2 Constraints on Values 16
4.3.3 Constraint Representation 16
4.3.4 Pre-evaluation 18
4.4 Class Descriptions 19
4.4.1 C_DV_TEXT Class 19
4.4.2 C_DV_CODED_TEXT Class 19
4.4.3 C_COORDINATED_TERM Class 19
4.4.4 C_TERM_MAPPING Class 20
4.4.5 C_PARAGRAPH Class 20
5 AM.DATA_TYPES.QUANTITY Package 21
5.1 Overview 21
5.1.1 Requirements 21
5.2 Class Descriptions 21
5.2.1 C_DV_ORDERED Class 21
5.2.2 C_DV_INTERVAL<T : DV_ORDERED> Class 22
5.2.3 C_DV_ORDINAL Class 22
5.2.4 C_DV_QUANTIFIED Class 23
5.2.5 C_DV_QUANTITY Class 23
5.2.6 C_DV_QUANTITY_RATIO Class 23
5.2.7 C_DV_CUSTOMARY_QUANTITY Class 24
6 AM.QUANTITY.DATA_TYPES.DATE_TIME Package 25
6.1 Overview 25
6.2 Class Descriptions 25
6.2.1 C_DV_WORLD_TIME Class 25

6.2.2 C_DV_DATE Class...25
6.2.3 C_DV_TIME Class...26
6.2.4 C_DV_DATE_TIME Class..26
6.2.5 C_DV_DURATION Class 26
6.2.6 C_DV_PARTIAL_DATE...27
6.2.7 C_DV_PARTIAL_TIME 27

7 AM.DATA_TYPES.TIME_SPECIFICATION Package.... 28
7.1 Overview 28
7.2 Class Descriptions 28
7.2.1 C_DV_TIME_SPECIFICATION...28
7.2.2 C_DV_PERIODIC_TIME_SPECIFICATION 28
7.2.3 C_DV_TIME_SPECIFICATION...29

8 AM.DATA_TYPES.ENCAPSULATED Package............... 30
8.1 Overview 30
8.2 Class Descriptions 30
8.2.1 C_DV_ENCAPSULATED Class...30
8.2.2 C_DV_MULTIMEDIA Class 30
8.2.3 C_DV_PARSABLE Class...31

9 AM.DATA_TYPES.URI Package....................................... 32
9.1 Overview 32
9.2 Class Descriptions 32
9.2.1 C_DV_URI Class...32
9.2.2 C_DV_EHR_URI Class..32

A References... 35
A.1 General 35
A.2 European Projects..35
A.3 CEN 35
A.4 GEHR Australia..35
A.5 HL7..35
A.6 openEHR 36
A.7 Resources..36

1 Introduction

1.1 Purpose

This document describes the *open*EHR Data Types Archetype Model (AM), which describes the semantics of constraints on data instances of the types defined in the *open*EHR Data Types Reference Model (RM). The intended audience includes:

- Standards bodies producing health informatics standards
- Software development organisations using *open*EHR
- Academic groups using *open*EHR
- The open source healthcare community

1.2 Related Documents

Prerequisite documents for reading this document include:

- The *open*EHR Modelling Guide

Other documents describing related models, include:

- The *open*EHR Data Types Reference Model

1.3 Status

This document is under development, and is published as a proposal for input to standards processes and implementation works.

Currently the UML diagrams are hand-produced. Various tool versions exist (Rose, Objecteering), but the visual quality is still being improved; when this is complete, the tool-generated images will be used.

Also in the future, specific design principles will be referred to throughout the model text, so that readers can easily find the theoretical discussion on which any part of the model is based.

The latest version of this document can be found in PDF and HTML formats at http://www.openEHR.org/doculist.htm. New versions are announced on openehr-announce@openehr.org.

1.4 Peer review

Known omissions or questions are indicated in the text with a “to be determined” paragraph, as follows:

TBD_1: (example To Be Determined paragraph)

Areas where more analysis or explanation is required are indicated with “to be continued” paragraphs like the following:

To Be Continued: more work required

Reviewers are encouraged to comment on and/or advise on these paragraphs as well as the main content. Please send requests for information to info@openEHR.org. Feedback should preferably be discussed on one of the appropriate mailing lists, openehr-technical@openehr.org or openehr-clinical@openehr.org.

2 Overview

2.1 Package Structure

The package structure is illustrated in FIGURE 1.

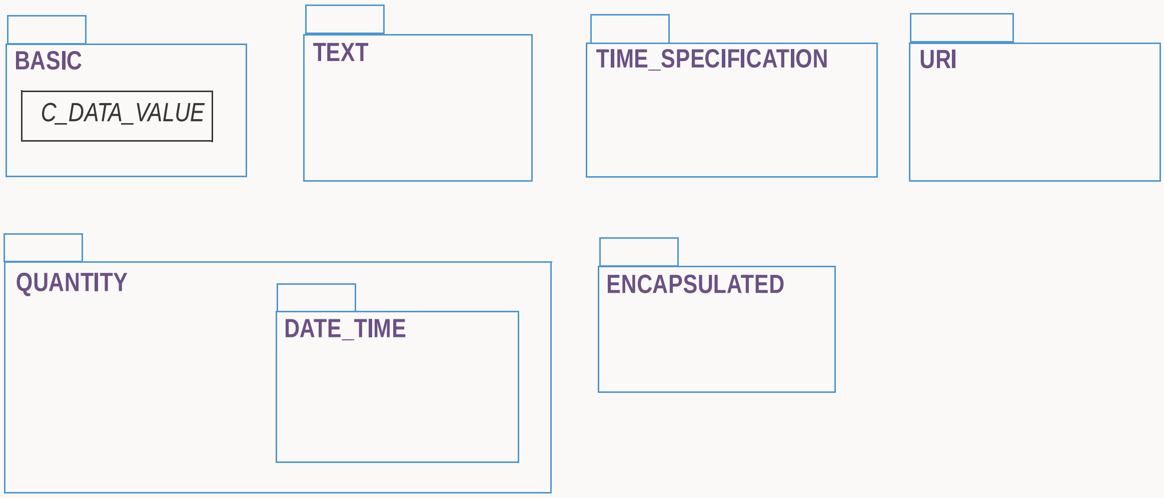

FIGURE 1 AM.DATA_TYPES Package

3 AM.DATA_TYPES.BASIC Package

The BASIC package, illustrated in FIGURE 2, contains types representing the concepts of “boolean” and “state”.

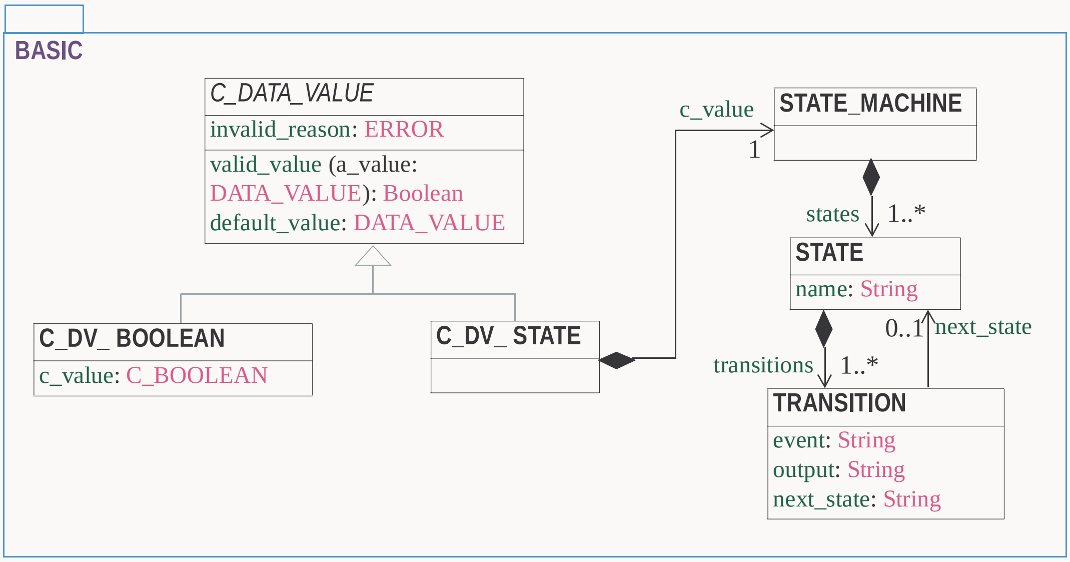

FIGURE 2 AM.DATA_TYPES.BASIC Package

3.1 Class Descriptions

3.1.1 C_DATA_VALUE Class

CLASS	*C_DATA_VALUE (abstract)*	
Purpose	Serves as a common ancestor of all archetyped data value types in the this model. Defines the abstract signature of the features *valid_value* and *default_value*.	
Attribute	Signature	Meaning
	invalid_reason : ERROR	Error posted if valid_value returns False
Abstract	Signature	Meaning
	valid_value (a_value: like ***Current***)**:**Boolean ***require*** a_value /= void Result ***xor*** invalid_reason /= Void	Test if a_value is valid according to this constraint object
	default_value: DATA_VALUE ***ensure*** Result /= void ***and then*** valid_value (Result)	Generate a default value. Note that other default values may be provided in local templates.
Invariants		

3.1.2 C_DV_BOOLEAN Class

CLASS	C_DV_BOOLEAN	
Purpose	Constrainer type for `DV_BOOLEAN` instances. The attributes *c_value_true* and *c_value_false* indicate which values of the constrained datum are allowed.	
Use	`C_DV_BOOLEAN` is used to constrain boolean data items in certain archetypes. For example:	
Inherit	`C_DATA_VALUE`	
Attributes	Signature	Meaning
	c_value: `C_BOOLEAN`	Allowed truth values.
Invariants	**c_value_exists**: c_value /= Void	

3.1.3 C_DV_STATE Class

CLASS	C_DV_STATE	
Purpose	Constrainer type for `DV_STATE` instances. The attribute *c_value* defines a state/event table which constrains the allowed values of the attribute *value* in a `DV_STATE` instance, as well as the order of transitions between values.	
Inherit	`C_DATA_VALUE`	
Attributes	Signature	Meaning
	c_value: `STATE_MACHINE`	
Invariants	**c_value_exists**: c_value /= Void	

A example of a state machine to model the state of a medication order is illustrated in FIGURE 3. This state machine is defined by an instance of the class `STATE_MACHINE`.

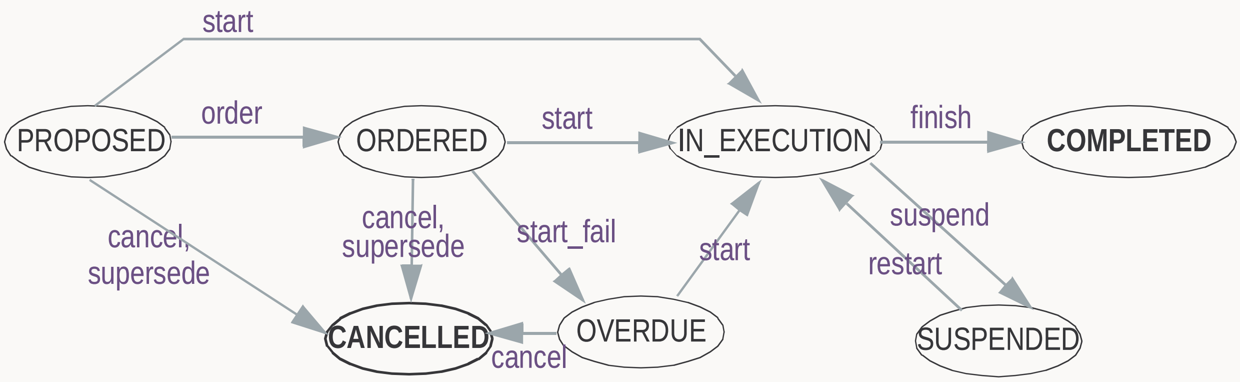

FIGURE 3 Example State Machine for Medication Orders

3.1.4 STATE_MACHINE Class

CLASS	STATE_MACHINE	
Purpose	Definition of a state machine in terms of states, transition events and outputs, and next states.	
Use		
Attributes	Signature	Meaning
	states: `Set <STATE>`	
Invariants	**states_valid**: states /= Void ***and then not*** states.empty	

3.1.5 STATE Class

CLASS	STATE	
Purpose	Definition of one state in a state machine.	
Use		
Attributes	Signature	Meaning
	name: `String`	name of this state
	transitions: `Set <TRANSITION>`	
Invariants	**transitions_valid**: transitions /= Void ***and then not*** transitions.empty	

3.1.6 TRANSITION Class

CLASS	TRANSITION	
Purpose	Definition of a state machine transition.	
Attributes	Signature	Meaning
	event: `String`	Event which fires this transition
	guard: `String`	Guard condition which must be true for this transition to fire
	action: `String`	Side-effect action to execute during the firing of this transition
	next_state: `STATE`	Target state of transition

CLASS	TRANSITION
Invariants	**event_valid**: event /= Void ***and then not*** event.empty **action_valid**: action /= Void ***implies not*** action.empty **guard_valid**: guard /= Void ***implies not*** guard.empty

4 AM.DATA_TYPES.TEXT Package

4.1 Overview

The TEXT package contains classes for expressing constraints on instances of the types defined in the TEXT package in the Data Types RM. It is illustrated in FIGURE 4.

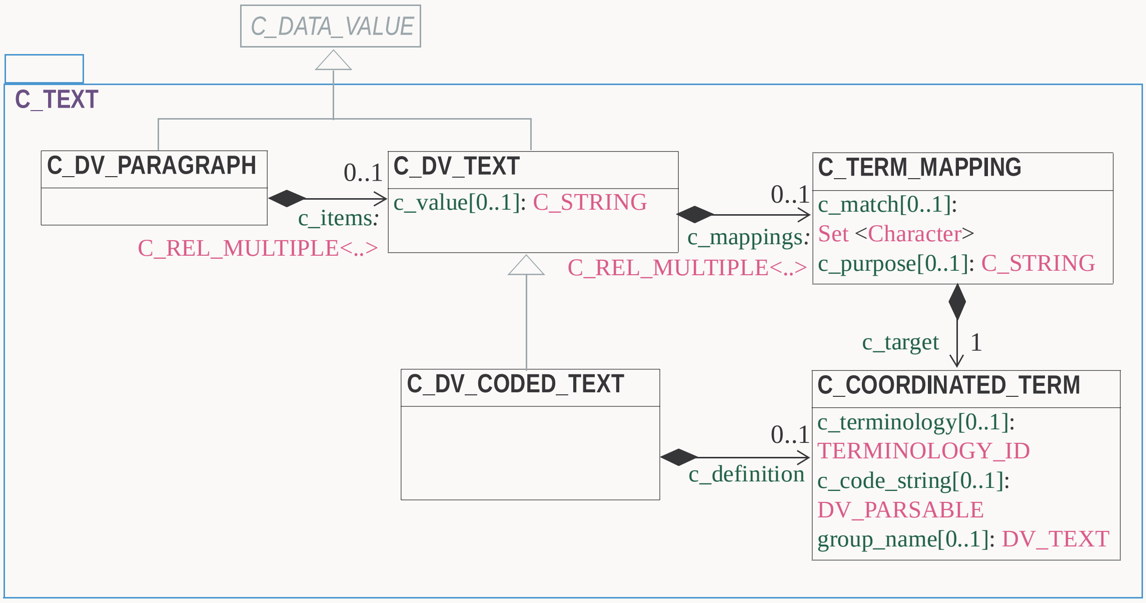

FIGURE 4 AM.DATA_TYPES.TEXT Package

4.2 Constraints on Plain Text Items

Plain text occurs in the record where subjective, imprecise or unstructured narrative is used. Constraints on plain text are expressed using “regular expressions” [4], a well-known syntactical expression language for matching string patterns. Typical constraints on plain text expressed this way include:

- `“..*”` - any non-empty string, e.g. forces the application to get a non-empty response;
- `“[a-zA-Z0-9]*”` - matches a string of any number of alphabetic characters, e.g. as might be used in a person’s address;
- `“[A-Z].*”` - string with at least one character which must be capitalised first letter, e.g. as in a person name;
- alternatives where coded terms are not available or not being used, e.g.:
  - `“sitting|lying down|standing up”` - matches any of the strings `“sitting”`, `“lying down”`, `“standing up”`;
  - `“A|B|O|AB”` - blood groups

Note that neither of the above examples are desirable ways of constraining values like blood group or patient position: it is much preferable to constrain a coded term for this purpose.

4.3 Constraints on Coded Text Items

Unlike plain text, the reasons and possibilities for constraining coded terms are quite involved, and require some explanation. One of the main complexities is the existence of multiple terminologies, which are not only often mutually inconsistent in their terms, but inconsistent in design. Some recent projects such as SNOMED-CT [15] and Galen [14] attempt to overcome inconsistencies using comprehensive structured approaches, while HL7 has taken a more pragmatic approach with numerous small "domains" - each a complete set of coded terms which define the domain of some datum. Other complicating factors include licencing costs and conditions (ensuring that some health care failities cannot afford them), unavailability (e.g. due to technology problems), and language translations.

Despite this situation, it is essential that archetypes can be created in such a way as to avoid direct dependency on particular *models* of terminologies, while being able to assume some abstract model of terminology.

Referring to the model for DV_CODED_TEXT [13], there are two attributes which can be constrained, in order to constrain instances of this class, namely *value* and *definition*. It must be remembered that in the model of DV_CODED_TEXT, the value/definition combination represent a rubric and a key from a terminology service, not a specific terminology. The key is in the form of a triplet <terminology id, code_string, language> representing the term, which may be a coordination of terms from a particular terminology, generated by the terminology service itself. The general approach taken here is that an instance of the class C_DV_CODED_TEXT expresses a constraint which evaluates to a candidate set of DV_CODED_TEXTs which are allowed values in a particular data context. There are two places in data where coded terms appear: as names, and as values. *All constraints can only be meaningfully evaluated against a terminology service, which has access to the relevant terminologies*. In general, a coded text constraint will be an expression to be evaluated by the terminology service - the only exceptions are the simplest constraints where the set of allowed terms is actually enumerated, and requires no further evaluation.

4.3.1 Constraints on Names

Where coded names occur in data e.g. in instances of FOLDER.*name*, SECTION.*name*, and CLUSTER.*name*, the following types of constraints are needed:

- require the term to be a particular one from a particular terminology, e.g. the ICD10 term "diabetes mellitus" (here the terminology is not limited to one value set);
- require the term to be any term from a particular terminology constrained by some relationship within the terminology, e.g. "is-a"; for example, "any term in ICD10 which is-a 'tropical infection'";
- require the term to be any term from a particular terminology, e.g. the HL7 PracticeSetting domain (here the terminology itself is limted to one value set);
- require the term to be one from any terminology, as long as it has a particular rubric.

The last of these could be achieved in two ways:

- enumerating some or all possible allowed coded terms from various terminologies;
- stating a concept which must be matched, e.g. a UMLS [16] meta-thesaurus concept unique indentifier (CUI).

Of the above, the latter should be the more correct solution, since it would make terminologies and terminology services responsible for resolving ambiguities.

4.3.2 Constraints on Values

The second kind of constraint on coded terms is used where terms appear as values. In this case, the intention is to specify a set of allowed terms, for example blood groups, diagnoses which may be relevant in the particular clinical setting, or the characteristics of a lump on palpation. More complex constraints specify that the set of terms is the union of two or more groups (the OR operator in queries), or is a member of a number of groups (the AND operator in queries), or even some more complex combination. In all cases, we can think of the constraint as returning a "candidate set of terms" when evaluated against real terminologies.

A candidate set of terms can be obtained from a terminology in a number of ways. First, via the use of *relationships* encoded in the terminology, such as: "X is-a-kind-of coronary disease", where classification relationships such as "is-a-kind-of" are defined in the terminology of interest. Second, by identifying terms which belong in some kind of group or category. Consider a constraint such as "X has-category palpable-body-part" which will return the set of terms which describe palpable body parts. These two methods may be mixed as in "X is-a-kind-of body-part AND has-category palpable", which uses both a relationship and a category - and is equivalent to the previous category described. Note that a constraint like "X is-a-kind-of body-part" is likely to return a long list of body parts, while the category of "palpable" body-parts would reduce this significantly. Such constraints should only be specified if there is likely to be a mechanism to implement the categorisation - this might not be in the terminology but must be available to the terminology service (i.e. it is an addition to the terminology proper, within the terminological knowledge environment accessible to the terminology service).

Further constraining can be achieved by the use of more boolean relationships on candidate sets produced by the method above, however it should always be understood that every time this is done, it in some sense usurps the role of knowledge / terminology. In theory only terminologies and ontologies can say that more than one candidate set of terms can be meaningfully intersected (AND operator) or unioned (OR operator) to produce a final meaningful set. However, the current reality is that very few terminologies implement even a small percentage of the possible knowledge relationships, and such constraints will indeed need to be made inside archetypes or other parts of the knowledge environment.

An example of such a constraint is:

```
    X is-a ‘surface body region’ OR (X is-a ‘organ’ AND has-category ‘palpable’)
```

The general case for value sets of coded terms is nested boolean expressions, where each expression element is one of the following:

- a particular term
- a named relationship
- a named category

For such expressions to be safe, all terms, relationships and categories must come from the same version of the same terminology, or an intentionally designed adjunct to it. This is the only way that *intended* meanings can be accessed. To arbitrarily mix terms and relationships from different terminologies is effectively side-stepping the known semantics of each of the systems, and creating value sets based on semantics not defined by anyone.

4.3.3 Constraint Representation

The kinds of constraints described above are essentially represented by two attributes in the model. The *c_value* in the class `C_DV_TEXT` encodes regular expression constraints on textual values, while coded terms constraints are expressed by the values of the *c_code_string* attribute of the `C_COORDINATED_TERM` class.

The attribute *c_code_string* is a syntax string which enables nested boolean expressions of atomic term constraints to be expressed, such as the following examples:

-
  ```
  {terminology_id=ICD10(1998);
  code_string = [F43.1(post traumatic stress disorder)]
  }
  ```

-
  ```
  {group_name = "acute stress reactions"(en);
  terminology_id=ICD10(1998);
  code_string in {
      [F43.00(acute stress reaction, mild)],
      [F43.01(acute stress reaction, moderate)],
      [F32.02(acute stress reaction, severe)]
  }}
  ```

-
  ```
  {group_name = "acute stress reactions"(en);
  terminology_id=ICD10(1998);
  code_string in {F43.0*}
  }
  ```

-
  ```
  {group_name = "treatment procedures & medication"(en);
  terminology_id=ICPC(1989);
  code_string in {*.3*}
  }
  ```

-
  ```
  {group_name = "treatment procedures & medication of eye & ear"(en);
  terminology_id=ICPC(1989)};
  code_string in {[F-H]*.3*};
  }
  ```

-
  ```
  {group_name = "body structures"(en);
  terminology_id = SNOMED-CT(2002);
  code_string in {
      has-relation [102002(is-a)]
      with-target [128004(body structure)]
  }}
  ```

-
  ```
  {group_name = "body structures"(en)
  terminology_id= SNOMED-CT(2002);
  code_string in {
      has-relation [XXXXXX(in-subset)] with-target
      [YYYYYY(palpable body structures \(surface anatomy\))]
  }}
  ```

-
  ```
  {group_name = "BP measurement positions"(en);
  terminology_id = UMLS(2002);
  code_string in {
      [XXXXXX(lying)],
      [YYYYYY(sitting)],
      [ZZZZZZ(standing)]
  }}
  ```

-
  ```
  {group_name = "investigation types"(en);
  terminology_id= UMLS(???);
  code_string in {
      has-relation [102002(is-a)] with-target [?????(urinalysis
  test)] OR
  ```

```
        has-relation [102002(is-a)] with-target [?????(electrical ana-
    log test)] OR
        has-relation [102002(is-a)] with-target [?????(imaging test)]
    OR
        has-relation [102002(is-a)] with-target [?????(biochemistry
    test)]
    }}
```

Each expression in braces ({}) is a *constraint*. The braces can be thought of delimiting the *set* of possibilities which the target item has to match. Any such constraint can be evaluated against a real terminology, Thus, the first example can be read as “the set containing the term ‘some term’”. All coded terms (including coordinated ones) are shown in the standard form `<“rubric” [code_phrase]>`.

The representation of constraints on terms is syntactical rather than structural, for two reasons. Firstly, it is easier to represent (and store) a potentially complex boolean expression as a syntax string (the equivalent structural form might be quite complex, and in any case, may not be the optimum form for evaluation). Secondly, the use of a single attribute of type `String` does not prevent changes to the syntax specification, allowing different syntaxes to be used in the future, without requiring changes to the archetype model or software or databases.

Syntax Definition

To Be Continued: THIS IS NOW OUT OF DATE; TO BE SUPERSEDED BY SHARED ARCHETYPE LANGUAGE SYNTAX

The syntax of *c_code_string* is defined as follows:

```
expression: term_constraint |
            ‘(’ expression ‘)’ |
            expression BINARY_BOOL_OP term_constraint

term_constraint: UNARY_BOOL_OP term_constraint |
            constraint_relation term_reference

constraint_relation: EQ |
            HAS_REL term_reference |
            HAS_CAT category_name

term_reference: ‘<x’ ‘”’ TEXT ‘”’ ‘[’ CODE_TEXT ‘]’

category_name: ‘”’ TEXT ‘”’

EQ: ‘=’
HAS_REL: ‘has-relation’
HAS_CAT: ‘has-category’
BINARY_BOOL_OP: ‘AND’ | ‘OR’ | ‘XOR’
UNARY_BOOL_OP: ‘NOT’
CODE_TEXT: ‘[a-zA-Z0-9-]+’
TEXT: ‘[a-zA-Z_- ][a-zA-Z0-9_- ]*’
```

4.3.4 Pre-evaluation

An archetype containing instances of `C_DV_CODED_TEXT` could be evaluated in advance against a terminology, to generate the actual sets of candidate terms, allowing the populated archetype to be distributed and used for coding even by sites without access to coding systems.

To Be Continued:

4.4 Class Descriptions

4.4.1 C_DV_TEXT Class

CLASS	*C_DV_TEXT (abstract)*	
Purpose	Constrainer for DV_TEXT instances.	
Inherit	C_DATA_VALUE	
Attributes	Signature	Meaning
	c_value: C_STRING	Constrain plain text by regular expression.
	c_mappings: C_REL_MULTIPLE <C_TERM_MAPPING>	Constrain mappings to text item.
Invariants		

4.4.2 C_DV_CODED_TEXT Class

CLASS	C_DV_CODED_TEXT	
Purpose	Express constraints on instances of DV_CODED_TEXT.	
Use		
GEHR	A_TERM_TEXT	
Inherit	C_DV_TEXT	
Attributes	Signature	Meaning
	c_definition: C_COORDINATED_TERM	Syntax string expressing constraint on allowed primary terms
Invariants		

4.4.3 C_COORDINATED_TERM Class

CLASS	C_COORDINATED_TERM	
Purpose	Constraint on a coordinated term instance.	
Use		
Attributes	Signature	Meaning

CLASS	C_COORDINATED_TERM	
	c_terminology_id: TERMINOLOGY_ID	Constraint on terminology used - can only be one
	c_code_string: DV_PARSABLE	Constraint on term itself
	group_name: DV_TEXT	Name of this group, provided by archetype author
Invariants	***c_terminology_id_valid***: c_terminology_id /= Void ***c_code_string_valid***: c_code_string /= void ***and*** c_code_string.formalism.is_equal(“openehr::coded text constraint”)	

4.4.4 C_TERM_MAPPING Class

CLASS	C_TERM_MAPPING	
Purpose	Constraint on instances of TERM_MAPPING	
Attributes	**Signature**	**Meaning**
	c_target: C_COORDINATED_TERM	Constraint on target term.
	c_match: Set<Character>	Allowed values for the *match* attribute.
Invariants		

4.4.5 C_PARAGRAPH Class

CLASS	C_PARAGRAPH	
Purpose	Constraint on instances of PARAGRAPH	
Attributes	**Signature**	**Meaning**
	c_items: C_REL_MULTIPLE<DV_TEXT>	Constraint on text items list in Paragraph.
Invariants	***c_items_valid***: c_items /= Void	

5 AM.DATA_TYPES.QUANTITY Package

5.1 Overview

The QUANTITY package is illustrated in FIGURE 5.

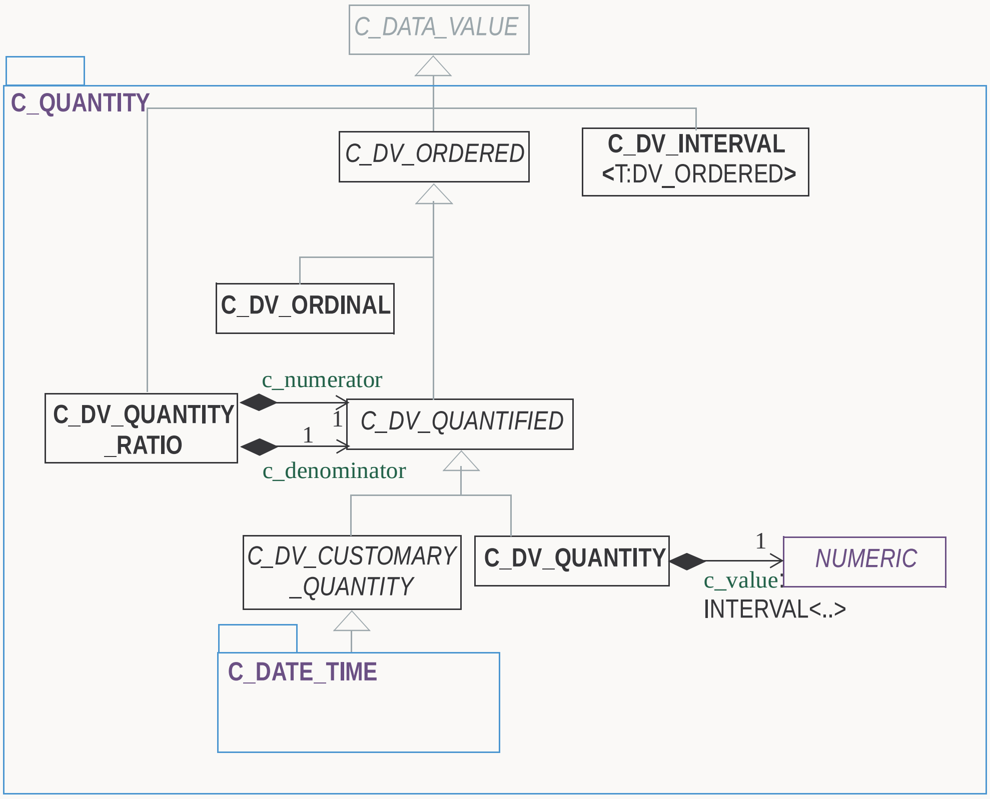

FIGURE 5 AM.DATA_TYPES.QUANTITY Package

5.1.1 Requirements

Constraints are expressible on instances of the four concrete types DV_INTERVAL, DV_ORDINAL, DV_QUANTITY_RATIO and DV_QUANTITY, and on the concrete date/time types.

5.2 Class Descriptions

5.2.1 C_DV_ORDERED Class

CLASS	*C_DV_ORDERED (abstract)*	
Purpose	Abstract class defining constraints applicable to all ordered types.	
Inherit	C_DATA_VALUE	
Attributes	**Signature**	**Meaning**
Invariants		

5.2.2 C_DV_INTERVAL<T : DV_ORDERED> Class

CLASS	C_DV_INTERVAL<T : DV_ORDERED>	
Purpose	Class defining constraints on intervals of ordered types.	
Attributes	**Signature**	**Meaning**
	c_lower: `DV_INTERVAL<T>`	Range constraining allowable values of lower end of an interval. If the two values are the same, *lower* is constrained to a point value. If either end of the interval is left open, *lower* is constrained to be any value from the closed end to - infinity. If no interval is supplied, there is no constraint.
	c_upper: `DV_INTERVAL<T>`	Range constraining allowable values of upper end of an interval. If the two values are the same, *upper* is constrained to a point value. If either end of the interval is left open, *upper* is constrained to be any value from the closed end to + infinity. If no interval is supplied, there is no constraint.
Functions	**Signature**	**Meaning**
	valid_lower_any: `Boolean` ***ensure*** c_lower = Void ***implies*** Result	*c_lower* does not impose any constraint on *lower*.
	valid_upper_any: `Boolean` ***ensure*** c_upper = Void ***implies*** Result	*c_upper* does not impose any constraint on *upper*.
Invariants		

5.2.3 C_DV_ORDINAL Class

CLASS	C_DV_ORDINAL
Purpose	Class specifying constraints on instances of `DV_ORDINAL`. Constrainer type for instances of `DV_ORDINAL`. Specified in terms of name(s) of `DV_ORDINAL` value sets, or 'domains' - just as for terminology value sets. The actual set of `DV_ORDINALs` defining the allowed values for a given datum is defined elsewhere, e.g. in a quantitative data server.
Inherit	`C_DV_ORDERED`

CLASS	C_DV_ORDINAL	
Attributes	Signature	Meaning
	c_type: Set<String>	Set of allowed DV_ORDINAL value sets.
Invariants		

5.2.4 C_DV_QUANTIFIED Class

CLASS	*C_DV_QUANTIFIED (abstract)*	
Purpose	Constrain quantified types.	
Inherit	C_DV_ORDERED	
Abstract	Signature	Meaning
	c_accuracy_is_percent: C_BOOLEAN	Constraint on whether accuracy_is_percent must be true, false or don't care.
Invariants		

5.2.5 C_DV_QUANTITY Class

CLASS	C_DV_QUANTITY	
Purpose	Constrain instances of DV_QUANTITY.	
GEHR	A_QUANTITY	
Inherit	C_DV_QUANTIFIED	
Attributes	Signature	Meaning
	c_value: INTERVAL<NUMERIC>	Value must be inside the supplied interval.
	c_units: C_STRING	Optional constraint on units
	c_property: C_DV_CODED_TEXT	Optional constraint on units property
Invariants		

5.2.6 C_DV_QUANTITY_RATIO Class

CLASS	C_DV_QUANTITY_RATIO
Purpose	Constrain instances of DV_QUANTITY_RATIO.

CLASS	C_DV_QUANTITY_RATIO	
GEHR	A_QUANTITY_RATIO	
Inherit	C_DATA_VALUE	
Attributes	Signature	Meaning
	c_numerator: C_DV_QUANTIFIED	Optional constraint on the numerator
	c_denominator: C_DV_QUANTIFIED	Optional constraint on the denominator
Invariants		

5.2.7 C_DV_CUSTOMARY_QUANTITY Class

CLASS	*C_DV_CUSTOMARY_QUANTITY (abstract)*	
Purpose	Abstract class defining constraints applicable to all customary quantity.	
Inherit	C_DV_QUANTITY	
Attributes	Signature	Meaning
Invariants		

6 AM.QUANTITY.DATA_TYPES.DATE_TIME Package

6.1 Overview

The DATE_TIME package is illustrated in FIGURE 6.

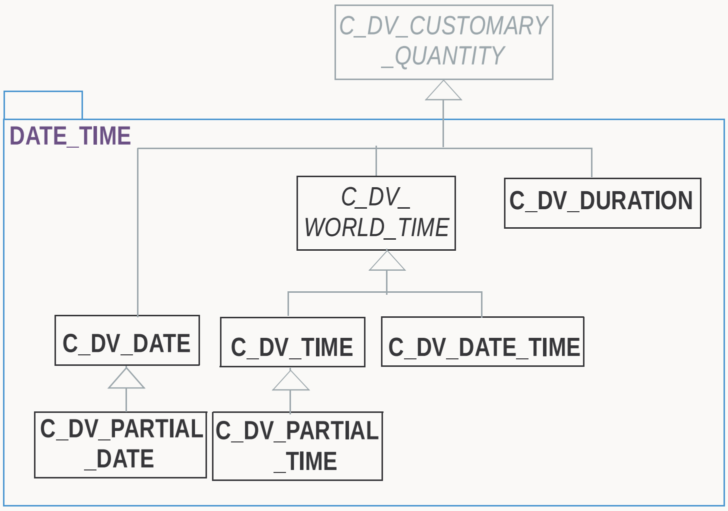

FIGURE 6 AM.DATA_TYPES.QUANTITY.DATE_TIME Package

6.2 Class Descriptions

6.2.1 C_DV_WORLD_TIME Class

CLASS	***C_DV_WORLD_TIME (abstract)***	
Purpose	Constrain instances of `DV_WORLD_TIME`.	
Inherit	`C_DV_CUSTOMARY_QUANTITY`	
Attributes	**Signature**	**Meaning**
Invariant		

6.2.2 C_DV_DATE Class

CLASS	**C_DV_DATE**
Purpose	Constrain instances of `DV_DATE`.
GEHR	`A_DATE`
Inherit	`C_DV_WORLD_TIME`

CLASS	C_DV_DATE	
Attributes	Signature	Meaning
	c_value: DV_INTERVAL<DV_DATE>	
Invariant		

6.2.3 C_DV_TIME Class

CLASS	C_DV_TIME	
Purpose	Constrain instances of DV_TIME.	
GEHR	A_TIME	
Inherit	C_DV_CUSTOMARY_QUANTITY	
Attributes	Signature	Meaning
	c_value: DV_INTERVAL<DV_TIME>	
Invariant		

6.2.4 C_DV_DATE_TIME Class

CLASS	C_DV_DATE_TIME	
Purpose	Constrain instances of DV_DATE_TIME.	
GEHR	A_DATE_TIME	
Inherit	C_DV_WORLD_TIME	
Attributes	Signature	Meaning
	c_value: DV_INTERVAL<DV_DATE_TIME>	
Invariant		

6.2.5 C_DV_DURATION Class

CLASS	C_DV_DURATION
Purpose	Constrain instances of DV_DURATION.

CLASS	C_DV_DURATION	
GEHR	A_DATE_TIME_DURATION	
Inherit	C_DV_CUSTOMARY_QUANTITY	
Attributes	Signature	Meaning
	c_value: DV_INTERVAL<DV_DURATION>	
Invariant		

6.2.6 C_DV_PARTIAL_DATE

CLASS	C_DV_PARTIAL_DATE	
Purpose	Constrain instances of DV_PARTIAL_DATE.	
Inherit	C_DV_DATE	
Attributes	Signature	Meaning
	c_month_known: C_BOOLEAN	
Invariant		

6.2.7 C_DV_PARTIAL_TIME

CLASS	C_DV_PARTIAL_TIME	
Purpose	Constrain instances of DV_PARTIAL_TIME.	
Inherit	C_DV_TIME	
Attributes	Signature	Meaning
	c_minute_known: C_BOOLEAN	
Invariant		

7 AM.DATA_TYPES.TIME_SPECIFICATION Package

7.1 Overview

These are illustrated in FIGURE 7.

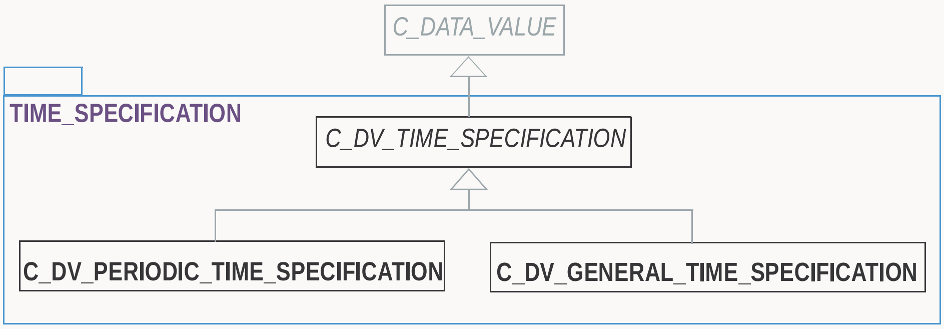
C_DATA_VALUE
TIME_SPECIFICATION
C_DV_TIME_SPECIFICATION
C_DV_PERIODIC_TIME_SPECIFICATION
C_DV_GENERAL_TIME_SPECIFICATION

FIGURE 7 AM.DATA_TYPES.TIME_SPECIFICATION Package

7.2 Class Descriptions

7.2.1 C_DV_TIME_SPECIFICATION

CLASS	***C_DV_TIME_SPECIFICATION (abstract)***	
Purpose	Constrain instances of `C_DV_TIME_SPECIFICATION`.	
Inherit	`C_DATA_VALUE`	
Attributes	**Signature**	**Meaning**
	c_calendar_alignment: `C_STRING`	
	c_event_alignment: `C_STRING`	
	c_institution_specified: `C_BOOLEAN`	
Invariant		

7.2.2 C_DV_PERIODIC_TIME_SPECIFICATION

CLASS	**C_DV_PERIODIC_TIME_SPECIFICATION**	
Purpose	Constrain instances of `C_DV_PERIODIC_TIME_SPECIFICATION`.	
Inherit	`C_DV_TIME_SPECIFICATION`	
Attributes	**Signature**	**Meaning**

CLASS	C_DV_PERIODIC_TIME_SPECIFICATION
Invariant	

7.2.3 C_DV_TIME_SPECIFICATION

CLASS	C_DV_GENERAL_TIME_SPECIFICATION	
Purpose	Constrain instances of C_DV_GENERAL_TIME_SPECIFICATION.	
Inherit	C_DV_TIME_SPECIFICATION	
Attributes	Signature	Meaning
Invariant		

8 AM.DATA_TYPES.ENCAPSULATED Package

8.1 Overview

The ENCAPSULATED package contains classes which constrain instances of classes defined in the package Encapsulated. It is illustrated in FIGURE 8.

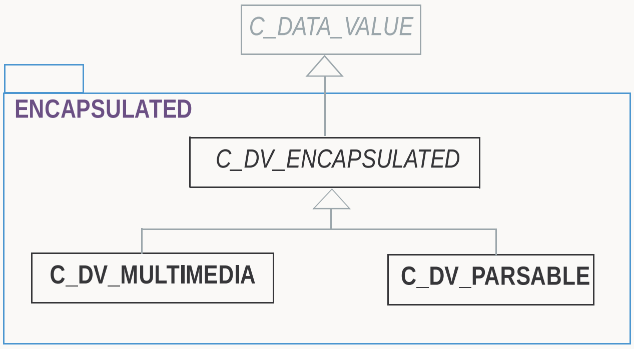

FIGURE 8 AM.DATA_TYPES.ENCAPSULATED Package

8.2 Class Descriptions

8.2.1 C_DV_ENCAPSULATED Class

CLASS	*C_DV_ENCAPSULATED (abstract)*	
Purpose	Abstract parent of types which constrain instances of `DV_ENCAPSULATED` subtypes.	
Inherit	`C_DATA_VALUE`	
Attributes	**Signature**	**Meaning**
Invariant		

8.2.2 C_DV_MULTIMEDIA Class

CLASS	C_DV_MULTIMEDIA	
Purpose	Constrain instances of `DV_MULTIMEDIA`	
Inherit	`C_DV_ENCAPSULATED`	
Attributes	**Signature**	**Meaning**
	c_uri: `C_DV_URI`	Constraint on allowable URIs.
	c_media_type: `C_COORDINATED_TERM`	Constraint on allowable media types..

CLASS	C_DV_MULTIMEDIA
Invariant	

8.2.3 C_DV_PARSABLE Class

CLASS	C_DV_PARSABLE	
Purpose	Constrain instances of DV_PARSABLE.	
Inherit	C_DV_ENCAPSULATED	
Attributes	Signature	Meaning
	c_formalism: C_STRING	Force formalism to match some expression.
Functions	Signature	Meaning
Invariant	***c_formalism_exists***: c_formalism /= Void	

9 AM.DATA_TYPES.URI Package

9.1 Overview

The URI Package is illustrated in FIGURE 9.

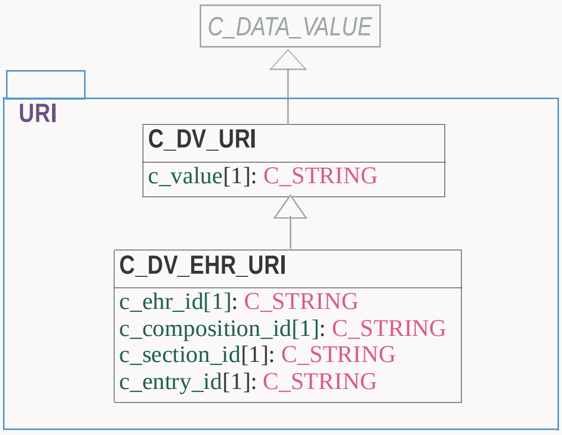

FIGURE 9 AM.DATA_TYPES.URI Package

9.2 Class Descriptions

9.2.1 C_DV_URI Class

CLASS	C_DV_URI	
Purpose	Constrain instances of `C_DV_URI`.	
Inherit	`C_DATA_VALUE`	
Attributes	Signature	Meaning
	c_value: `C_STRING`	Pattern for value to match.
Invariant		

9.2.2 C_DV_EHR_URI Class

CLASS	C_DV_EHR_URI	
Purpose	Constrain instances of DV_EHR_URI	
Use		
Inherit	`C_DV_URI`	
Attributes	Signature	Meaning

CLASS	**C_DV_EHR_URI**	
	c_ehr_id: `C_STRING`	
	c_composition_id: `C_STRING`	
	c_section_id: `C_STRING`	
	c_entry_id: `C_STRING`	
Invariant		

A References

A.1 General

1 Berners-Lee T. "Universal Resource Identifiers in WWW". Available at http://www.ietf.org/rfc/rfc2396.txt. This is a World-Wide Web RFC for global identification of resources. In current use on the web, e.g. by Mosaic, Netscape and similar tools. See http://www.w3.org/Addressing for a starting point on URIs.

2 Beale T. *Archetypes: Constraint-based Domain Models for Future-proof Information Systems*. See http://www.deepthought.com.au/it/archetypes.html.

3 Beale T *et al*. *Design Principles for the EHR*. See http://www.deepthought.com.au/openEHR.

4 Regular expressions - SOME REF NEEDED

5 Schadow G, McDonald C J. *The Unified Code for Units of Measure*, Version 1.4, April 27, 2000. Regenstrief Institute for Health Care, Indianapolis. See http://aurora.rg.iupui.edu/UCUM

6 ISO 8601 standard describing formats for representing times, dates, and durations. See e.g. http://www.mcs.vuw.ac.nz/technical/software/SGML/doc/iso8601/ISO8601.html and http://www.cl.cam.ac.uk/~mgk25/iso-time.html.

A.2 European Projects

7 Dixon R, Grubb P, Lloyd D. *EHCR Support Action Deliverable 3.5: "Final Recommendations to CEN for future work"*. Oct 2000. Available at http://www.chime.ucl.ac.uk/HealthI/EHCR-SupA/documents.htm.

A.3 CEN

8 ENV 13606-1 - *Electronic healthcare record communication - Part 1: Extended architecture*. CEN/ TC 251 Health Informatics Technical Committee.

9 ENV 13606-2 - *Electronic healthcare record communication - Part 2: Domain term list*. CEN/ TC 251 Health Informatics Technical Committee.

10 ENV 13606-3 - *Electronic healthcare record communication - Part 3: Distribution rules*. CEN/ TC 251 Health Informatics Technical Committee.

A.4 GEHR Australia

11 Beale T, Heard S. *GEHR Technical Requirements*. See http://www.gehr.org/technical/requirements/gehr_requirements.html.

A.5 HL7

12 Schadow G, Biron P. HL7 version 3 deliverable: *Version 3 Data Types*. (DRAFT Revision 1.0).

A.6 *open*EHR

13 Beale T, Heard S, Kalra D, Lloyd D. *openEHR Data Types Reference Model*. See http://www.openehr.org/productDT.htm.

A.7 Resources

14 Galen. See http://www.openGalen.org.

15 College of American Pathologists. Systematized Nomenclature of Medicine (SNOMED). http://www.snomed.org/.

16 UMLS (Unified Medical Language System). http://www.nlm.nih.gov/research/umls/.

17 WONCA. International Classification of Primary Care (ICPC) (second revision). http://www.ulb.ac.be/esp/wicc/icpc2.html

18 World Health Organisation. International Classification of Disease (ICD). http://www.who.int/whosis/icd10/.

END OF DOCUMENT