2 Overview

2.1 Design Approach

2.1.1 Overview

As described in [2], an archetype model is created for each reference model for which it is desired to create archetypes, which are both domain concept models, and structural constraint definitions. An archetype model is usually nearly isomorphic to the reference model from which it is derived - that is, it's structural form is similar. Each key class in a reference model has a correspondent in the archetype model. The relationship between other semantic constructs in a reference model and its archetype model, including relationships, are described in detail in [2].

The data types are the lowest level construct in any reference model. Accordingly, the data types archetype model defines types which are mostly custom-designed. This is not surprising; consider that types such as `DV_CODED_TEXT` and `DV_QUANTITY`: these embody quite complex semantics. In a imilar way, their archetype counterparts `C_DV_CODED_TEXT` and `C_DV_QUANTITY` express complex semantics which have been discovered largely by experience.

This is in contrast with higher levels of constructs, such as `ENTRY`, `ORGANISER` etc in the EHR reference model, whose semantics are essentially that of containment. Consequently, the archetype classes for these constructs are largely automatically derivable from their reference model counterparts.

2.1.2 Naming

Classes with a correspondent in the Data Types RM are named with the same name preceded by "C_" indicating "constraint for". Class features which are direct homologues for features in the reference model classes are named similarly, with a preceding "c_". This is done so that other features added to archetype classes are clearly distinct from those features required to express constraints. All such features carry data describing a constraint. Where archetypes are persisted, all "c_" features should be persisted.

2.1.3 Independence

`To Be Continued:    of particular terminological models`

2.1.4 Design of Archetype Editors

2.2 Definitions

Various terminologies are assumed in this specification.

2.3 Package Structure

The package structure is illustrated in FIGURE 1.

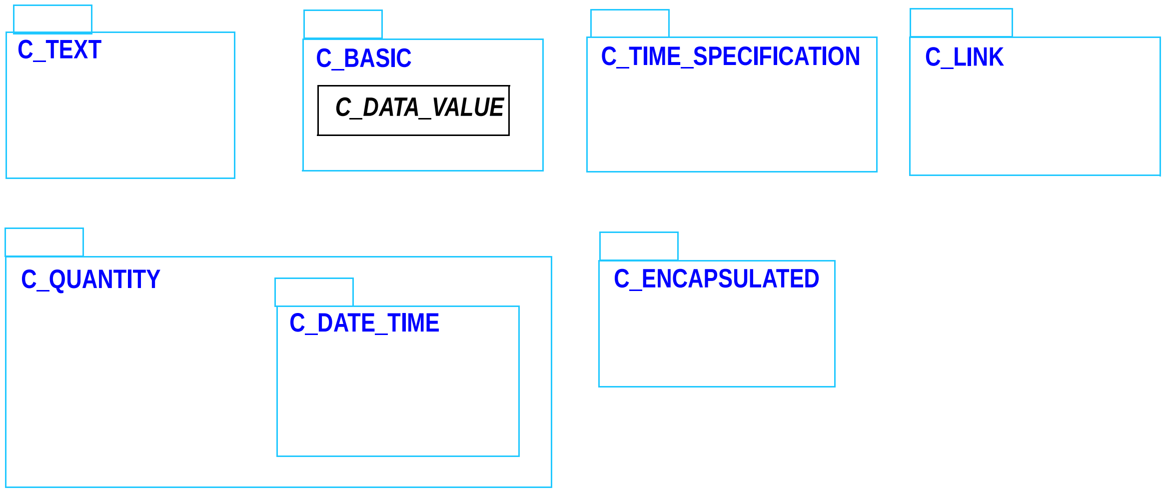

FIGURE 1 C_DATA Package

3 C_BASIC Package

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

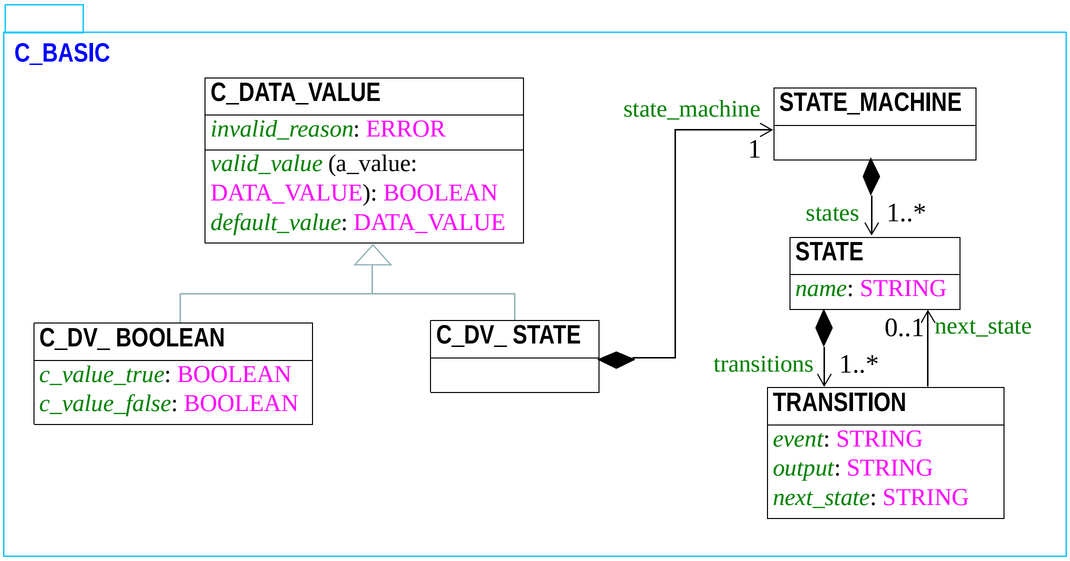

FIGURE 2 C_BASIC Package

3.1 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	
Abstract	**Signature**	**Meaning**
	valid_value(a_value: like ***Current***): BOOLEAN ***require*** value_exists: a_value /= void Result ***xor*** last_validation_failed	
	default_value: DATA_VALUE ***ensure*** result_valid: Result /= void ***and then*** valid_value (Result)	
	last_validation_failed: BOOLEAN ***ensure*** result_valid: Result invalid_reason /= Void	

CLASS	C_DATA_VALUE (abstract)	
	as_display_string: STRING ***ensure*** Result_exists: Result /= Void	String form of data item, i.e. a short, human-readable form of the data item. Not guaranteed to contain all attributes.
	as_canonical_string: STRING ***ensure*** Result_exists: Result /= Void	Standardised string form of data item, in tagged XML format, including all attributes.
Invariants		

3.2 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:	
Synapses		
Attributes	**Signature**	**Meaning**
	c_value_true: BOOLEAN	The value True of the constrained datum is valid
	c_value_false: BOOLEAN	The value True of the constrained datum is valid
Functions	**Signature**	**Meaning**
	as_display_string: STRING	Result = “valid values:” + {“True”, “False”}
	as_canonical_string: STRING	Result = “<c_value_true>” + c_value_true.out + </c_value_true>” + “<c_value_false>” + c_value_false.out + </c_value_false>”
Invariants		

3.3 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.	
Use		
Attributes	**Signature**	**Meaning**
	c_value: `STATE_MACHINE`	
Functions	**Signature**	**Meaning**
	as_display_string: `STRING` ***ensure*** Result.is_equal(c_value.out)	Result = value
	as_canonical_string: `STRING`	Result = “<c_value>” + c_value.out + “</c_value>”
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`.

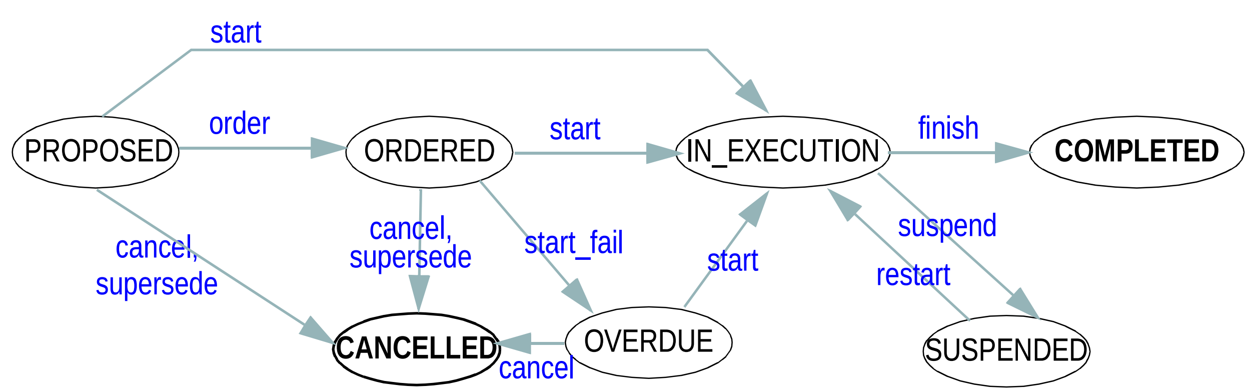

FIGURE 3 Example State Machine for Medication Orders

3.3.1 STATE_MACHINE Class

CLASS	**STATE_MACHINE**
Purpose	Definition of a state machine in terms of states, transition events and outputs, and next states.
Use	

CLASS	STATE_MACHINE	
Attributes	Signature	Meaning
	states: SET <STATE>	
Invariants	**states_valid**: states /= Void ***and then not*** states.empty	

3.3.2 STATE Class

CLASS	STATE	
Purpose	Definition of one state in a state machine.	
Use		
Attributes	Signature	Meaning
	states: SET <STATE>	
Invariants	**states_valid**: states /= Void ***and then not*** states.empty	

3.3.3 TRANSITION Class

CLASS	TRANSITION	
Purpose	Definition of a state machine transition.	
Attributes	Signature	Meaning
	event: STRING	
	guard: STRING	
	action: STRING	
	next_state: STATE	
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 C_TEXT Package

4.1 Overview

The C_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.

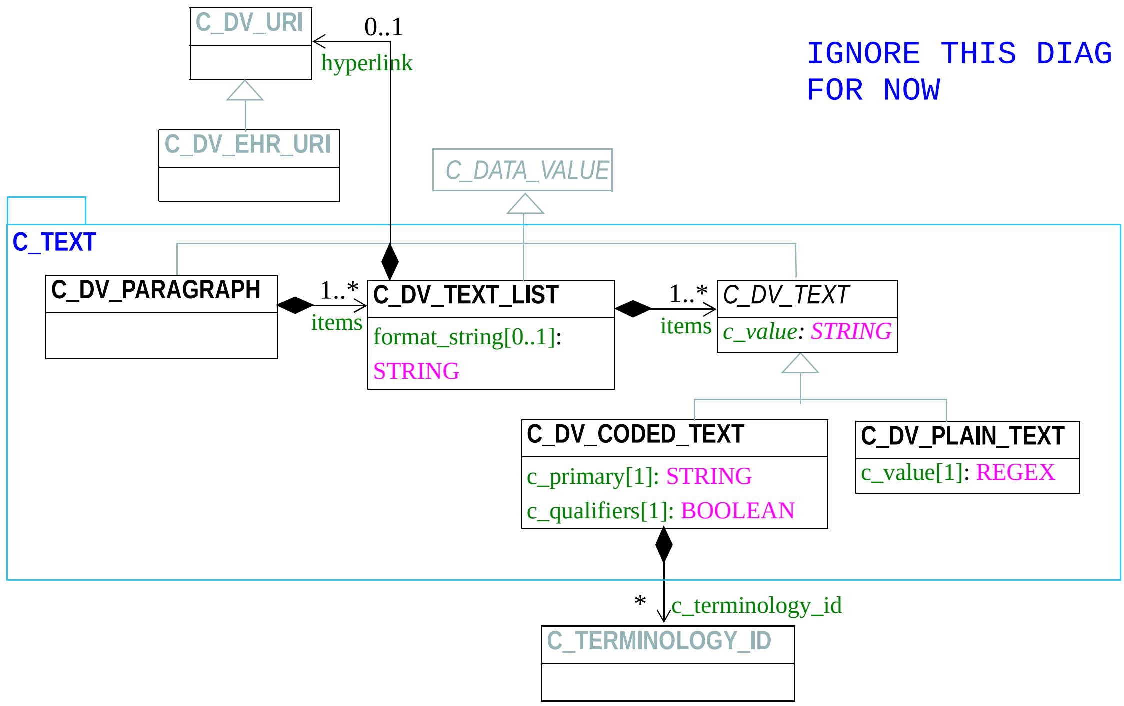

FIGURE 4 C_TEXT Package

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].*"` - 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), unvailability (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 a number of things which could possibly be constrained, in order to constrain instances of this class, namely the attributes *primary*, *qualifiers*, *terminology_id*, and *equivalents*. The last of these, *equivalents*, cannot meaningfully be constrained in an archetype, since the use of equivalents is a local affair. The following subsections describe types of constraint on the remaining attributes.

4.3.1 Constraints on Names

There are two broad types of constraints on meaning. The first is where the intention is to require a coded term instance, typically representing the name of something, e.g. “heart rate”. The two possible types of constraints here are:

- require the chosen to term to be a particular one from a particular terminology, e.g. the ICD10 term “diabetes mellitus”;
- require the chosen term to be one from any terminology, as long as it has a particular meaning.

The latter can 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. However, there is no guarantee that lexically matched concepts (i.e. concepts with the same name) always mean the same thing in different terminologies. Accordingly, at this stage, it would appear that the use of CUIs must be done on an informed case-by-case basis.

4.3.2 Constraints on Values

The second kind of constraint on coded terms is usually used where terms appear as values. In this case, the intention is to specify a set of allowed terms, for example blood groups, types of coronary disease, or characterisations of a lump. More complex constraints require any term which is a member of more than one group, or is in one of a number of groups, or 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.

One way to obtain a candidate set of terms is to use *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, usually in the form of coded terms.

Another way is to identify terms which belong in some kind of group, which may not be explicitly modelled in the terminology, or if it is, not as a relationship, but a category, grouping or subset of some kind. Consider a constraint such as "X has-category palpable-body-part" or equivalently, "X is-a-kind-of body-part AND has-category palpable", which uses both a relationship and a category. 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.

In some cases, more complex constraints are needed, e.g. the following:

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

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

- a particular term
- a relationship
- a category

This is the basis for the model of the class `C_DV_CODED_TEXT`.

For such expressions to be safe, all terms, relationships and categories must come from the same version of the same terminology, or be 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 Other Constraints

Other types of constraints which are possible are statements about term qualifiers (`DV_CODED_TEXT`.*qualifiers*) and terminology identifiers (`DV_CODED_TEXT`.*terminology_id*).

Qualifiers, such as laterality, or more specific additions to core terms, such as "acute myeloid" added to "leukaemia", or "mild" added to "rash" are available to be added to the primary term, if the terminology permits it, as modelled by the `TERM_RELATION` class in the data types reference model. It is up to the terminology to govern the use of qualifiers, consequently, constraints should not be used to replace or override this function.

The only sensible constraint on qualifiers seems to be whether in a given case, they are allowed or not. For example, it would be reasonable to force the use of unqualified disease terms in a problem list - e.g. "glaucoma" is more sensible than "glaucoma, right eye", even though the latter may appear in a diagnosis recorded in a patient contact elsewhere in the record.

Lastly, terminology identifiers can sensibly be constrained by name, version, and variant, where relevant. Since terminology identifiers are expressed as strings in the `TERMINOLOGY_ID` class, such constraints need to be in the form of regular expressions, such as "ICD*-CM" (any version of ICD clinical modifications), "ICD9|ICD10" (ICD9 or ICD10) and so on.

TBD_2: maybe terminology_ids need to be transparent - not opaque which would allow constraints like name="ICD" version>=10 but version ids are not guaranteed to be numerics anyway.

As shown above, there can only be one terminology_id for a whole boolean term expression, in order to guarantee meaningful semantics of the constraint expression, and a sensible candidate value set.

4.3.4 Constraint Representation

The representation of constraints on terms used in the `C_DV_CODED_TEXT` class 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.

A `C_DV_CODED_TEXT` accordingly consists of the following items:

- c_terminology_id: `REGULAR_EXPRESSION` -- constraint on terminology id
- c_primary: `STRING` -- constraint on primary - see syntax below
- c_qualifiers: `BOOLEAN` -- constraint on qualifiers (allowed/not allowed)

Syntax Definition

The syntax of c_primary is as follows:

To Be Continued: this is just a very rough first cut

```
expression: term_constraint |
             '(' expression ')' |
             expression BINARY_BOOL_OP term_constraint
term_constraint: UNARY_BOOL_OP term_constraint |
             'T' constraint_relation term_reference
constraint_relation: '=' |
                   'has-relation' term_reference |
                   'has-category' category_name
term_reference: term_code '(' NAME ')'
category_name: '"' NAME '"'

BINARY_BOOL_OP: 'AND' | 'OR' | 'XOR'
UNARY_BOOL_OP: 'NOT'
NAME: '[a-zA-Z_- ][a-zA-Z0-9_- ]*'
```

Syntax Examples

To Be Continued:

```
T = 12345(some term)
T has-relation 11111(is-a) 22222(some term)
T has-category "some category"
T has-relation 11111(is-a) OR (T has-relation 11111(is-a) AND T has-category
"some category")
```

4.3.5 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 C_DV_TEXT Class

CLASS	*C_DV_TEXT (abstract)*	
Purpose	Abstract parent of concrete text constraint classes.	
Use		
Abstract	Signature	Meaning
Functions	Signature	Meaning
	as_display_string: STRING	Result = value
Invariants		

TBD_3: Should C_DV_TEXT.value be Unicode rather than plain string

4.5 C_DV_PLAIN_TEXT Class

CLASS	C_DV_PLAIN_TEXT	
Purpose	C_DV_PLAIN_TEXT constrains instances of DV_PLAIN_TEXT. The constraint is expressed as a regular expression string.	
GEHR	A_PLAIN_TEXT	
Attributes	Signature	Meaning
	value: REGULAR_EXPRESSION	Regular expression used to constrain textual items.
Functions	Signature	Meaning
	as_canonical_string: STRING	Result =
Invariants		

4.6 C_DV_CODED_TEXT Class

CLASS	C_DV_CODED_TEXT
Purpose	Express constraints on instances of DV_CODED_TEXT.
Use	

CLASS	C_DV_CODED_TEXT	
GEHR	A_TERM_TEXT	
Attributes	**Signature**	**Meaning**
	c_primary: STRING	Syntax string expressing constraint on allowed primary terms
	c_qualifiers: BOOLEAN	True if qualifiers allowed, False if not.
Invariants	***minimal_term***: c_primary /= void ***and then not*** c_primary.empty ***terminology_id_exists***: terminology_id /= void	

5 QUANTITY Package

5.1 Overview

The Quantity package is illustrated in FIGURE 5.

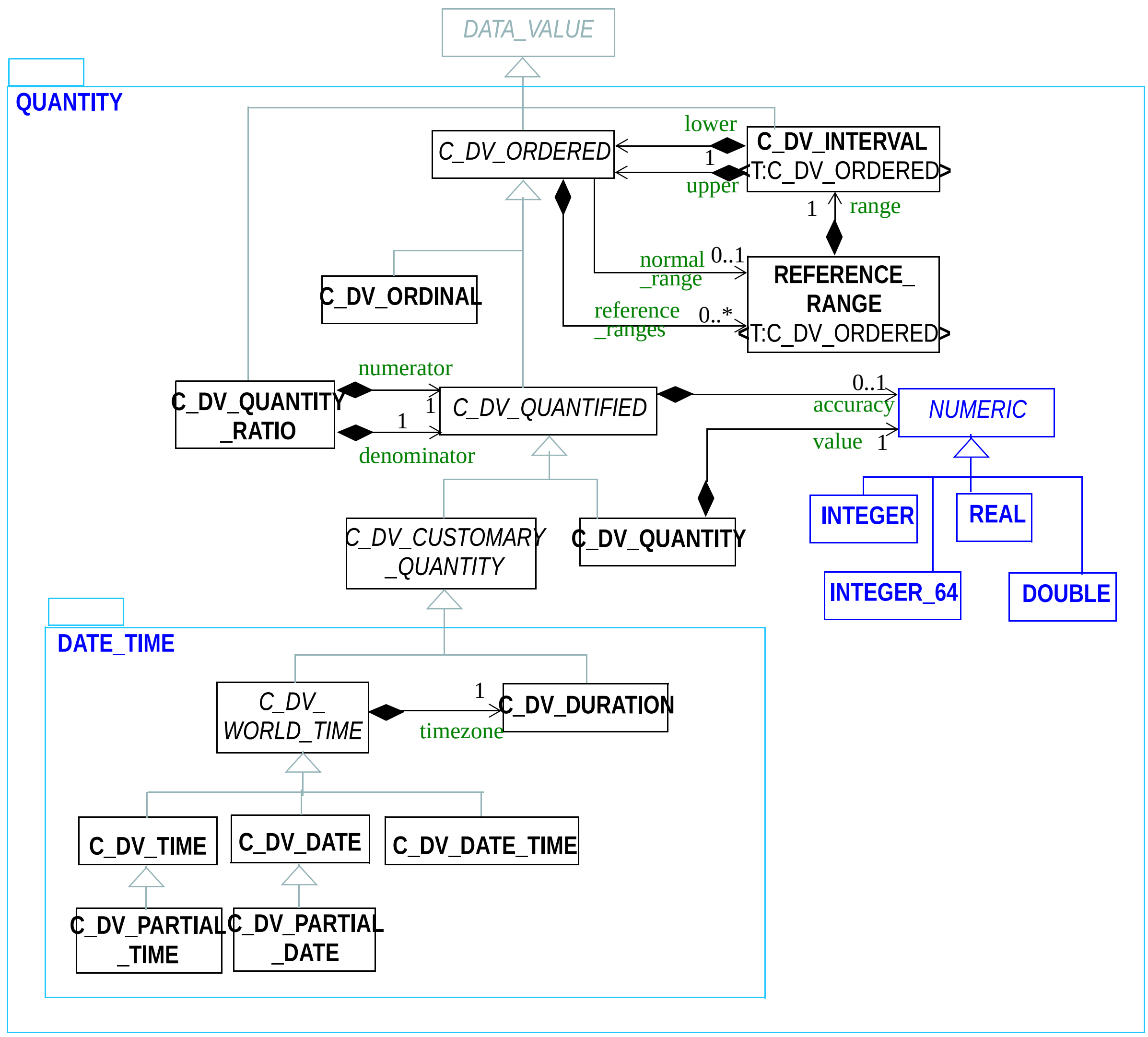

FIGURE 5 Quantity Package

5.1.1 Requirements

5.1.2 General Design

5.2 C_DV_ORDERED Class

CLASS	*C_DV_ORDERED (abstract)*
Purpose	

CLASS	*C_DV_ORDERED (abstract)*	
Attributes	**Signature**	**Meaning**
	reference_ranges: LIST<REFERENCE_RANGE<***like*** Current>>	
	normal_range: REFERENCE_RANGE<***like*** Current>	
Abstract	**Signature**	**Meaning**
	less_than **(**other**:** ***like*** Current**)**: BOOLEAN ***require*** **is_strictly_comparable_to**(other)	
	is_strictly_comparable_to **(**other**:** ***like*** Current**)**: BOOLEAN	
Functions	**Signature**	**Meaning**
	is_simple: BOOLEAN ***ensure*** reference_ranges = Void ***implies*** is_simple	
Invariants	***Reference_range_validity***: reference_ranges /= Void ***implies not*** reference_ranges.is_empty ***Normal_range_validity***: normal_range /= Void ***implies (***reference_ranges /= Void ***and then*** reference_ranges.has(normal_range))	

5.3 C_DV_INTERVAL<T : C_DV_ORDERED> Class

CLASS	C_DV_INTERVAL<T : C_DV_ORDERED>	
Purpose		
Attributes	**Signature**	**Meaning**
	lower: T	
	upper: T	
	lower_unbounded: BOOLEAN	
	upper_unbounded: BOOLEAN	
Functions	**Signature**	**Meaning**

CLASS	C_DV_INTERVAL<T : C_DV_ORDERED>
Invariants	***Lower_exists***: lower /= Void ***xor*** lower_unbounded ***Upper_exists***: upper /= Void ***xor*** lower_unbounded ***Limits_consistent***: (***not*** upper_unbounded ***and not*** lower_unbounded) ***implies*** lower <= upper ***Limits_comparable***: (***not*** upper_unbounded ***and not*** lower_unbounded) ***implies*** lower.strictly_comparable_to(upper)

5.4 REFERENCE_RANGE<T:C_DV_ORDERED> Class

CLASS	REFERENCE_RANGE<T:C_DV_ORDERED>	
Purpose		
Use		
Functions	Signature	Meaning
	meaning: C_DV_CODED_TEXT	
	range: C_DV_INTERVAL<T>	
Invariants	***Meaning_exists***: meaning /= Void ***Range_exists***: range /= Void ***Range_is_simple***: range.lower.is_simple ***and*** range.upper.is_simple	

5.5 C_DV_ORDINAL Class

CLASS	C_DV_ORDINAL	
Purpose		
Attributes	Signature	Meaning
	value: INTEGER	
	rubric: STRING	
	type: STRING	
Invariants	***Value_validity***: value > 0 ***Type_exists***: type /= Void ***and then not*** type.empty	

5.6 C_DV_QUANTIFIED Class

CLASS	*C_DV_QUANTIFIED (abstract)*	
Purpose		
Abstract	Signature	Meaning
	units: STRING	
	property: C_DV_CODED_TEXT	
	accuracy: NUMERIC	
	accuracy_is_percent: BOOLEAN	
Invariants	***units_validity***: units /= void ***implies*** property /= void ***accuracy_validity***: accuracy_is_percent ***implies*** accuracy.is_valid_percentage ***property_validity***: property /= Void ***implies*** prop-erty.terminology_id.value.is_equal("UnitsOfMeasureProperties")	

5.7 C_DV_QUANTITY Class

CLASS	C_DV_QUANTITY	
Purpose		
GEHR	A_QUANTITY	
Attributes	Signature	Meaning
	value: NUMERIC	
	precision: INTEGER	
Invariants	***Precision_valid***: precision >= 0	

5.8 C_DV_QUANTITY_RATIO Class

CLASS	C_DV_QUANTITY_RATIO	
Purpose		
GEHR	A_QUANTITY_RATIO	
Attributes	Signature	Meaning
	numerator: C_DV_QUANTIFIED	

CLASS	C_DV_QUANTITY_RATIO	
	denominator: `C_DV_QUANTIFIED`	
Invariants	***Numerator_exists***: numerator /= Void ***Denominator_exists***: denominator /= Void	

6 DATE_TIME Package

6.1 Overview

6.1.1 Design Basis

6.2 C_DV_WORLD_TIME Class

CLASS	*C_DV_WORLD_TIME (abstract)*	
Purpose		
Use		
Attributes	Signature	Meaning
	timezone: `C_DV_DURATION`	
Invariant	***timezone_valid***: timezone /= Void ***and then*** (timezone >= Min_timezone ***and*** timezone <= Max_timezone)	

6.3 C_DV_DATE Class

CLASS	C_DV_DATE	
Purpose	Represents an absolute point in time, as measured on the Gregorian calendar, and specified only to the day.	
GEHR	`A_DATE`	
Attributes	Signature	Meaning
	year: `INTEGER`	
	month: `INTEGER`	
	day: `INTEGER`	
Invariant	***Validity***: is_valid_date(year, month, day)	

6.4 C_DV_TIME Class

CLASS	C_DV_TIME
Purpose	

email: info@openEHR.org web: www.openEHR.org

CLASS	C_DV_TIME	
GEHR	A_TIME	
Attributes	Signature	Meaning
	hour: INTEGER	
	minute: INTEGER	
	second: INTEGER	
Invariant	***validity***: is_valid_time(hour, minute, second)	

TBD_4: fine seconds needs to be supported (ISO 18308)

6.5 C_DV_DATE_TIME Class

CLASS	C_DV_DATE_TIME	
Purpose		
GEHR	A_DATE_TIME	
Attributes	Signature	Meaning
	year: INTEGER	
	month: INTEGER	
	day: INTEGER	
	hour: INTEGER	
	minute: INTEGER	
	second: INTEGER	
Invariant	***validity***: is_valid_date_time(year, month, day, hour, minute, second)	

6.6 C_DV_DURATION Class

CLASS	C_DV_DURATION
Purpose	
GEHR	A_DATE_TIME_DURATION

CLASS	C_DV_DURATION	
Attributes	**Signature**	**Meaning**
	sign: `INTEGER`	
	days: `INTEGER`	
	hours: `INTEGER`	
	minutes: `INTEGER`	
	seconds: `INTEGER`	
Invariant	***validity***: is_valid_duration(days, hours, minutes, seconds) ***Sign_valid***: sign = -1 ***or else*** sign = 1	

6.7 C_DV_PARTIAL_DATE

CLASS	C_DV_PARTIAL_DATE	
Purpose		
Attributes	**Signature**	**Meaning**
	month_known: `BOOLEAN`	
Invariant		

6.8 C_DV_PARTIAL_TIME

CLASS	C_DV_PARTIAL_TIME	
Purpose		
Attributes	**Signature**	**Meaning**
	minute_known: `BOOLEAN`	
Invariant		

7 TIME_SPECIFICATION Package

7.1 Overview

These are illustrated in FIGURE 6.

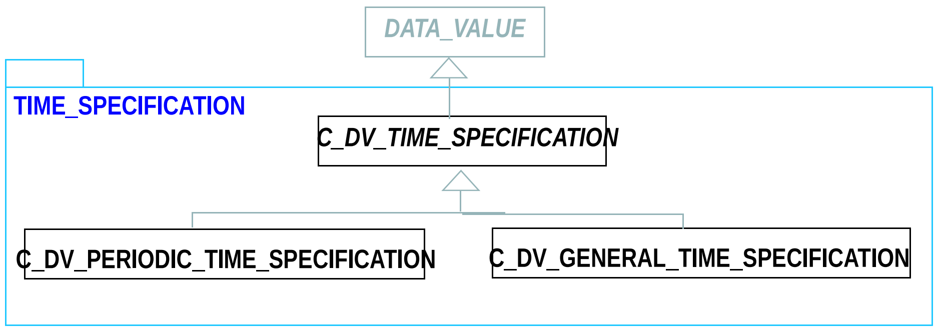

FIGURE 6 Time Specification Package

7.2 C_DV_TIME_SPECIFICATION

CLASS	***C_DV_TIME_SPECIFICATION (abstract)***	
Purpose		
Attributes	**Signature**	**Meaning**
	value: C_DV_PARSABLE	
Invariant	***Value_valid***: value /= Void ***and then*** value.formalism = "HL7:GTS"	

8 ENCAPSULATED Package

8.1 Overview

The Encapsulated package contains classes representing data values whose internal structure is defined outside the EHR model, such as multimedia and parsable data. It is illustrated in FIGURE 7.

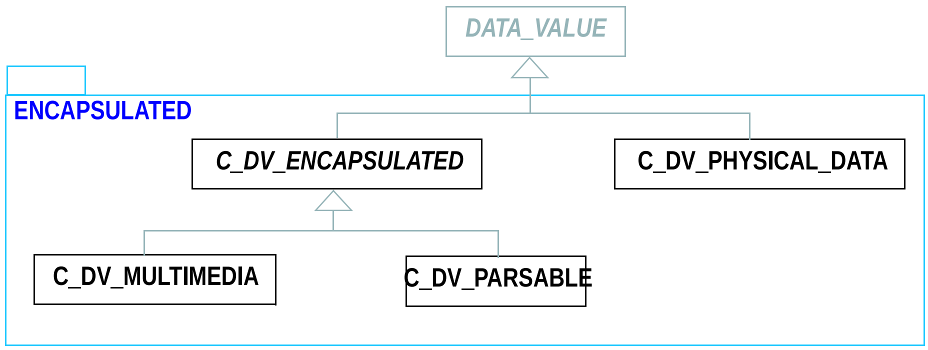

FIGURE 7 Encapsulated Package

8.2 C_DV_ENCAPSULATED Class

CLASS	*C_DV_ENCAPSULATED (abstract)*	
Purpose		
Attributes	**Signature**	**Meaning**
	alternate_text: STRING	
	uri: STRING	
	charset: C_DV_CODED_TEXT	
	language: C_DV_CODED_TEXT	
	size: INTEGER	
	data: ARRAY <CHARACTER>	
Invariant	***Not_empty***: is_expanded ***or*** is_external ***Size_positive***: size >= 0 ***language_terminology***: language /= Void ***implies*** language.terminology_id.is_equal (Terminology_id_ISO_639) ***charset_terminology***: charset /= Void ***implies*** charset.terminology_id.is_equal(Terminology_id_IANA_charsets)	

TBD_5: C_DV_ENCAPSULATED - as_display_string should display a text equivalent + URI, if relevant

8.3 C_DV_MULTIMEDIA Class

CLASS	**C_DV_MULTIMEDIA**	
Purpose		
Attributes	**Signature**	**Meaning**
	media_type: `C_DV_CODED_TEXT`	
	compression_algorithm: `C_DV_CODED_TEXT`	
	integrity_check: `ARRAY<CHARACTER>`	
	integrity_check_algorithm: `C_DV_CODED_TEXT`	
	thumbnail: `C_DV_MULTIMEDIA`	
Invariant	***Media_type_terminology***: media_type.terminology_id.is_equal (Terminology_id_HL7_MediaType) ***Compression_algorithm_terminology***: compression_algorithm /= Void ***implies*** compression_algorithm.terminology_id.is_equal (Terminology_id_HL7_CompressionAlgorithm) ***Integrity_check_validity***: integrity_check /= Void ***implies*** integrity_check_algorithm /= Void ***Integrity_check_algorithm_terminology***: integrity_check_algorithm /= Void ***implies*** integrity_check_algorithm.terminology_id.is_equal (Terminology_id_HL7_IntegrityCheckAlgorithm)	

8.4 C_DV_PARSABLE Class

CLASS	**C_DV_PARSABLE**	
Purpose		
Use		
Attributes	**Signature**	**Meaning**
	value: `STRING`	
	formalism: `STRING`	
Functions	**Signature**	**Meaning**
Invariant	***value_valid***: value /= Void ***formalism_exists***: formalism /= Void ***and then not*** formalism.is_empty	

9 LINK Package

9.1 Overview

The Link Package is illustrated in FIGURE 8.

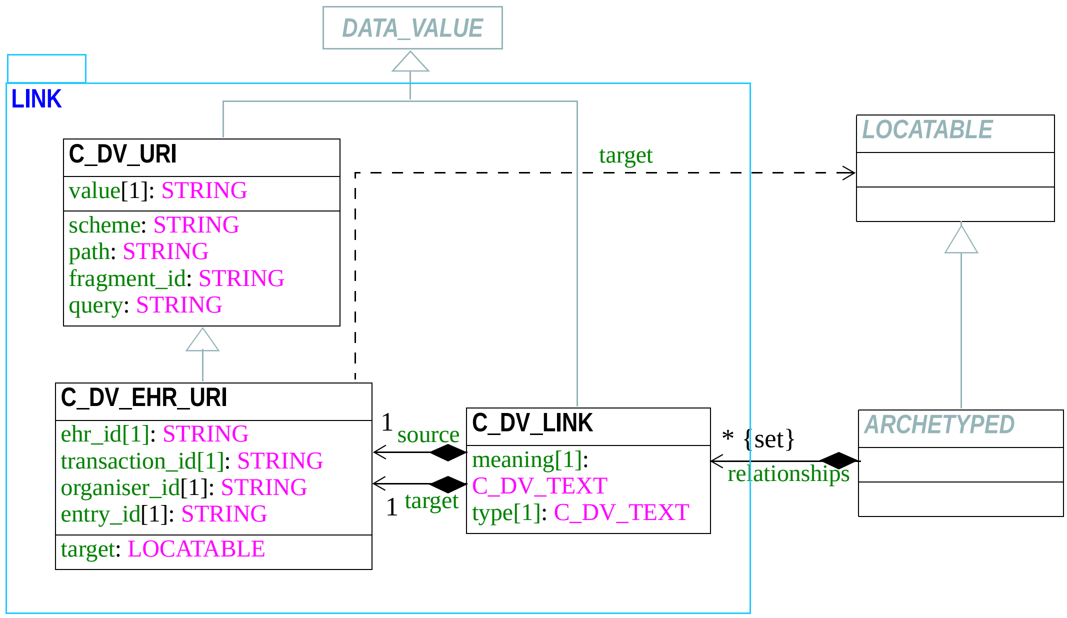

FIGURE 8 Link Package

9.2 C_DV_LINK Class

CLASS	**C_DV_LINK**	
Purpose		
Use		
Attributes	**Signature**	**Meaning**
	meaning: C_DV_TEXT	
	type: STRING	
	source: C_DV_EHR_URI	
	target: C_DV_EHR_URI	
Invariant	***meaning_exists***: meaning /= Void and then not meaning.is_empty ***type_exists***: type /= Void and then not type.is_empty ***source_exists***: source /= Void ***target_exists***: target /= Void	

9.3 C_DV_URI Class

CLASS	C_DV_URI	
Purpose		
Use		
Attributes	Signature	Meaning
	value: `STRING`	Value of URI as a String.
Invariant	***value_exists***: value /= Void ***and then not*** value.is_empty	

9.4 C_DV_EHR_URI Class

CLASS	C_DV_EHR_URI	
Purpose		
Use		
Attributes	Signature	Meaning
	ehr_id: `STRING`	
	transaction_id: `STRING`	
	organiser_id: `STRING`	
	entry_id:`STRING`	
Invariant	***Scheme_is_ehr***: scheme.is_equal("ehr")	