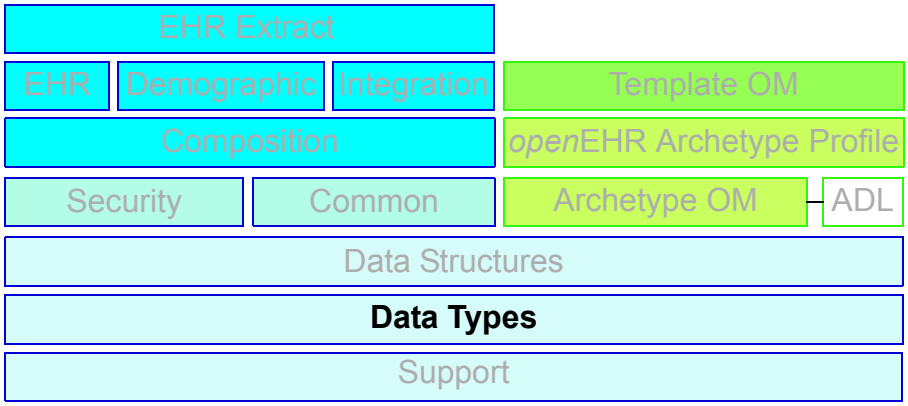
The *open*EHR Reference Model

Data Types Information Model

Issuer: openEHR Specification Program		
Revision: 2.1.1	*Pages:* 91	*Date of issue:* 20 Nov 2008
Status: STABLE		

Keywords: EHR, ADL, health records, modelling, constraints



Support **Issue tracker**: http://www.openehr.org/issues/
Web: http://www.openEHR.org

Amendment Record

Issue	Details	Raiser	Completed
	RELEASE 1.0.2		
2.1.1	**SPEC-257**: Correct minor typos and clarify text. Replace DV_EHR_URI with DV_URI in inheritance of class definition of DV_EHR_URI. Clarify the description of the *type* attribute in DV_PROPORTION to indicate how it is controlled by PROPORTION_KIND. Add *as_string*() function to DV_ENCAPSULATED class in UML diagram of encapsulated package. **SPEC-261**: Indicate how accuracy is treated over add/subtract operations in DV_QUANTIFIED types. Indicate special values for unknown accuracy.	T Cook C Ma, R Chen G Geurts, R Chen	20 Nov 2008
	RELEASE 1.0.1		
2.1.0	**CR-000144**: Add new type: DV_PROPORTION. **CR-000198**: Change DV_Date/Time/Duration to have value as attribute. **CR-000199**: Add *normal_range* attribute to DV_ORDERED. **CR-000200**. Correct Release 1.0 typographical errors. Correct DV_ENCAPSULATED.*size* to abstract in definition table. Correct DV_STATE.*value* in UML of basic package to be DV_CODED_TEXT. Correct DV_ORDINAL.*symbol* type to DV_CODED_TEXT in UML diagram for QUANTITY package. Add missing inheritance of Ordered to DV_ORDERED. **CR-000205**: Convert Date/time constants to a class. **CR-000211**: Add *magnitude_status* to DV_QUANTIFIED. **CR-000215**: Merge DV_PARTIAL_XX date/time classes and move ISO 8601 semantics to Support IM. Remove DV_WORLD_TIME class. **CR-000216**: Allow mixture of W, D etc in ISO8601 Duration (deviation from standard). **CR-000219**: Use constants instead of literals to refer to terminology in RM. **CR-000221**. Add *normal_status* to DV_ORDERED. Adjusted invariants. **CR-000227**: Remove DV_QUANTITY_RATIO. **CR-000230**: Change DV_DATE_TIME.*to_quantity* to seconds **CR-000236**: Change use of Character to Octet in DV_MULTIMEDIA. **CR-000237**: Correct semantics of Quantity and Date/Time types. **CR-000240**: Allow DV_ORDINAL values to be negative. **CR-000247**: Add DV_TEMPORAL class to Quantity package.	S Heard S Heard S Heard H Frankel S Heard G Grieve D Lloyd R Chen G Grieve D Lloyd S Heard T Beale S Heard R Chen H Frankel S Heard C Ma G Grieve T Beale G Grieve R Chen H Frankel	12 Apr 2007
	RELEASE 1.0		

Issue	Details	Raiser	Completed
2.0.0	**CR-000176**. Make DV_QUANTIFIED *accuracy* optional. **CR-000163**. Add identifiers to FEEDER_AUDIT for originating and gateway systems. Added *assigner* attribute to DV_IDENTIFIER. **CR-000121**. Improve DV_EHR_URI model to support Xpath style paths. **CR-000161**. Support distributed versioning. Remove functions from DV_EHR_URI.	S Heard H Frankel T Beale T Beale H Frankel	01 Feb 2006
RELEASE 0.96			
RELEASE 0.95			
1.9.1	Improve implementation guidance. DV_ORDINAL.*limits* type corrected to REFERENCE_RANGE<DV_ORDINAL>.	D Lloyd	22 Feb 2005
1.9	**CR-000126**. Correct details of partial date/time classes. **CR-000112**. Add DV_PARTIAL_DATE_TIME class **CR-000113**. Add DATA_VALUE subtype for identifying real-world entities **CR-000118**. Make package names lower case. **CR-000119**. Improve Data types documentation. **CR-000102**. Make DV_TEXT *language* and *charset* optional.	T Beale DSTC DSTC T Beale T Beale DSTC	09 Dec 2004
RELEASE 0.9			
1.8	**CR-000023**. TERM_MAPPING.*match* should be coded/enumerated. **CR-000069**. Correct date/time types statistical descriptions. **CR-000046**. Rename COORDINATED_TERM to CODE_PHRASE and DV_CODED_TEXT.*definition* to *defining_code*. **CR-000084**. Rename DV_COUNTABLE to DV_COUNT. **CR-000090**. Make TERM_MAPPING.*purpose* optional. **CR-000091**. Correct anomalies in use of CODE_PHRASE and DV_CODED_TEXT. **CR-000094**. Add lifecycle state attribute to VERSION; correct DV_STATE. **CR-000095**. Remove *property* attribute from Quantity package. **Formally validated using ISE Eiffel 5.4.**	G Grieve A Goodchild T Beale DSTC DSTC T Beale DSTC DSTC, S Heard	09 Mar 2004
1.7.9	**CR-000066**. Make DV_ORDERED.*normal_range* a function. Correct UML for DV_QUANTITY.	Z Tun	10 Nov 2003
1.7.8	**CR-000053**. Make DV_ORDINAL.*limits* a function. **CR-000054**. Move DV_QUANTIFIED.*is_normal* to DV_ORDERED **CR-000055**. Redefine DV_ORDERED.*less_than* as infix function '<'.	T Beale T Beale T Beale	02 Nov 2003
1.7.7	**CR-000041**. Visually differentiate primitive types in openEHR documents. **CR-000034**. State representation of date/time classes to be ISO8601. **CR-000052**. Change DV_DURATION.*sign* to prefix "-" operation. **CR-000042**. Make DV_ORDINAL.*rubric* a DV_CODED_TEXT; *type* attribute not needed.	D Lloyd, DSTC, T Beale	26 Oct 2003

Issue	Details	Raiser	Completed
1.7.6	**CR-000013**. Rename key classes, according to CEN ENV 13606. **CR-000026**. Rename DV_QUANTITY.*value* to *magnitude*. **CR-000031**. Change abstract NUMERIC to DOUBLE in DV_QUANTITY.*value*.	S Heard, D Kalra, T Beale, A Goodchild, Z Tun	01 Oct 2003
1.7.5	**CR-000022**. Code TERM_MAPPING.*purpose*.	G Grieve	20 Jun 2003
1.7.4	**CR-000020**. Move VERSION.*charset* to DV_TEXT, *territory* to TRANSACTION. Remove VERSION.*language*.	A Goodchild	10 Jun 2003
1.7.3	DV_INTERVAL now inherits from INTERVAL to avoid duplicating semantics. (Formally validated).	T Beale	25 Mar 2003
1.7.2	Minor corrections to diagrams in Text package. Improved heading structure, package naming. Corrected error in TEXT package diagram. Replaced TEXT_FORMAT_PROPERTY class with string attribute of same form. Made MULTIMEDIA.*media_type* mandatory. (Formally validated).	T Beale, Z Tun	21 Mar 2003
1.7.1	Moved *definitions* and *assumed types* to Support Reference Model. No semantic changes.	T Beale	25 Feb 2003
1.7	**Formally validated using ISE Eiffel 5.2.** **CR-000001**. Review of Data Types specification. Made pluralities of Terminology name definitions (sect 3.2.1) consistent. Corrected types of DV_ENCAPSULATED.*language*, *charset*, DV_MULTIMEDIA.*integrity_check_algorithm*, *compression_algorithm*, *media_type*. Corrected pluralities of Terminology name definitions (sect 3.2.1). Corrected invariants of DV_ENCAPSULATED, DV_MULTI_MEDIA, DV_QUANTITY, DV_CODED_TEXT, DV_TEXT, DV_INTERVAL, TERM_MAPPING. Corrected DV_TEXT.*formatting*; added TERM_MAPPING validity function. Made DV_ORDINAL.*limits* an attribute. Removed TERM_MAPPING.*source*; moved COORDINATED_TERM.*language* to DV_TEXT; changed type to COOORDINATED_TERM. Corrected time specification classes.	Z Tun, T Beale	17 Feb 2003
1.6.1	**Rome CEN TC 251 meeting.** Updates to HL7 comparison text. DV_DATE now inherits from DV_CUSTOMARY_QUANTITY.	S Heard, T Beale	27 Jan 2003
1.6	**Sam Heard complete review**. Changed constant terminology defs to runtime-evaluated set; removed DV_PHYSICAL_DATA. Added new chapter for generic implementation guidelines, and new section for assumed types. Post-conditions moved to invariants: DV_TEXT.*value*, DV_ORDERED.*is_simple*, DV_PARTIAL_DATE.*probable_date*, *possible_dates*, DV_PARTIAL_TIME.*probable_time*, *possible_times*. Minor updates to HL7 comparison text. Added explanation to HL7 section.	S Heard, T Beale	13 Dec 2002
1.5.9	Minor corrections: DV_ENCAPSULATED; DV_QUANTITY.*units* defined to be String; changed COORDINATED_TERM class (but semantically equivalent).	T Beale	10 Nov 2002

Issue	Details	Raiser	Completed
1.5.8	Changed name of LINK package to URI. Major update to Text cluster classes and explanation. Updated HL7 data type comparison.	T Beale, D Kalra, D Lloyd, M Darlison	1 Nov 2002
1.5.7	DV_TEXT_LIST reverted to TEXT_LIST. DV_LINK no longer a data types; renamed to LINK and moved to Common RM. Link package renamed to "URI".	S Heard, Z Tun, T Beale, D Kalra, M Darlison	18 Oct 2002
1.5.6	Rewrite of TIME_SPECIFICATION parse specs. Adjustments to DV_ORDINAL.	T Beale	16 Sep 2002
1.5.5	Timezone not allowed on pure DV_DATE in ISO8601.	T Beale, S Heard	2 Sep 2002
1.5.4	Moved DV_QUANTIFIED.*units* and *property* attributes to DV_QUANTITY. Introduced DV_WORLD_TIME.*to_quantity*. Added *fractional_second* to DV_TIME, DV_DATE_TIME, DV_DURATION.	T Beale, S Heard	29 Aug 2002
1.5.3	Further corrections - removed derived '/' markers; renamed TERM_MAPPING.*granularity* to *match*. Improved explanation of DV_ORDINAL. DV_QUANTIFIED.*units* is now a DV_PARSABLE. REFERENCE_RANGE.*meaning* is now a DV_TEXT. DV_ENCAPSULATED.*uri* is now a DV_URI. DV_LINK.*type* is now a DV_TEXT. Detailed review by Zar Zar Tun (DSTC).	T Beale, S Heard, P Schloeffel, D Lloyd, Z Tun	20 Aug 2002
1.5.2	Further corrections - removed derived '/' markers; renamed TERM_MAPPING.*granularity* to *match*.	T Beale, D Lloyd, S Heard	15 Aug 2002
1.5.1	Minor corrections.	T Beale, S Heard	15 Aug 2002
1.5	Rewrite of section describing text types; addition of new attribute DV_CODED_TEXT.*mappings*. Removal of TERM_REFERENCE.*concept_code*.	T Beale, S Heard	1 Aug 2002
1.4.3	Minor changes to text. Corrections to DV_CODED_TEXT relationships. Made DV_INTERVAL.*lower_unbounded* and DV_INTERVAL.*upper_unbounded* functions.	T Beale, Z Tun	16 Jul 2002
1.4.2	DV_LINK.*meaning* changed to DV_TEXT (typo in table). Added abstract class DV_WORLD_TIME.	T Beale, D Lloyd	14 Jul 2002
1.4.1	Changes to DV_ENCAPSULATED, DV_PARSABLE invariants.	T Beale Z Tun	10 Jul 2002
1.4	DV_ENCAPSULATED. *text_equivalent* renamed to DV_ENCAPSULATED.*alternate_text*. Added invariant for QUANTITY.*precision*.	T Beale, D Lloyd	1 Jul 2002
1.3	Added *timezone* to DV_TIME and DV_DATE_TIME and *sign* to DV_DURATION; added *linguistic_order* to TERM_RELATION; added *as_display_string* and *as_canonical_string* to all types. Added DV_STATE.*is_terminal*. Renamed TERM_TEXT as CODED_TEXT.	T Beale, D Lloyd	30 Jun 2002

Issue	Details	Raiser	Completed
1.2	Minor corrections to Text package.	T Beale	15 May 2002
1.1	Numerous small changes, including: term equivalents, relationships and quantity reference ranges.	T Beale, D Lloyd, D Kalra, S Heard	10 May 2002
1.0	Separated from the *open*EHR Reference Model.	T Beale	5 May 2002

Acknowledgements

The work reported in this paper has been funded by a number of organisations, including The University College, London; The Cooperative Research Centres Program through the Department of the Prime Minister and Cabinet of the Commonwealth Government of Australia; Ocean Informatics, Australia.

Table of Contents

1 Introduction

1.1 Purpose

This document defines the *open*EHR Data Types Information Model, used throughout the *open*EHR Reference Model. The intended audience includes:

- Standards bodies producing health informatics standards;
- Software development organisations developing EHR systems;
- Academic groups studying the EHR;
- The open source healthcare community;
- Medical informaticians and clinicians interested in health information.

1.2 Related Documents

Prerequisite documents for reading this document include:

- The *open*EHR Architecture Overview
- The *open*EHR Support Information Model

1.3 Status

The publication status of this specification is 'stable'. It is available at http://www.openehr.org/releases/trunk/architecture/rm/data_types_im.pdf.

The latest version of this document can be found at https://github.com/openEHR/specifications-RM/blob/master/publishing/data_types_im.pdf.

New versions are announced on the openehr-announce mailing list.

1.4 Peer review

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. Feedback should be provided either on the technical mailing list, or on the specifications issue tracker.

1.5 Conformance

Conformance of a data or software artifact to an *open*EHR Reference Model specification is determined by a formal test of that artifact against the relevant *open*EHR Implementation Technology Specification(s) (ITSs), such as an IDL interface or an XML-schema. Since ITSs are formal, automated derivations from the Reference Model, ITS conformance indicates RM conformance.

2 Background

2.1 Scope

The data type specification presented here defines the clinical/scientific data types which are used in other *open*EHR models. Harmonisation of data types between information models used by related services in a health information infrastructure is essential to reducing the conversion work and potential for errors between these services. Accordingly, the *open*EHR data type specification is intended to work not only for the EHR, but also for other models defined by *open*EHR, such as the *open*EHR demographic and terminological models.

The types described here have been derived from data types used in the GEHR [14], Synapses and SynEx [10], CEN 13606 [11], [13] and the HL7v3 [15] reference models.

2.2 Design Criteria

Over and above the need to satisfy the functional requirements of clinical data, three concerns have driven the design of the *open*EHR data types:

1. clarity of expression
2. ease of implementation
3. interoperability with data types from other standards

The first of these has led to models which try to clearly convey the semantics of types required by the clinical domain. The use of constraints (pre- and post-conditions and class invariants) and a comprehensible class structure ensures formal self-consistency, correct type-substitutability and implementability in object-oriented formalisms. Types have been designed so as not to clash with norms of object-oriented languages and libraries, in particular, class names and the inbuilt types. Accordingly, all types presented here have a logical name commencing with '`DV_`', ensuring that there is no clash with a type in the implementation formalism, hence the type `DV_DATE` presented here will not be confused with the type `DATE` which appears in many programming languages or libraries.

Object-oriented languages which have been considered include IDL, C++, Java, C#, Eiffel, Delphi and Python. Each of these languages obeys some variant of the well-known semantics of classes, encapsulation, typing and inheritance. The data types described here follow the tenets of object-orientation defined in UML most closely, while being careful not to invalidate their implementation in any language. The models have all been validated by implementation in the Eiffel language, the closest available semantic fit for UML, and currently the most powerful of mainstream object-oriented formalisms.

Implementability in XML-schema has also been an important design criterion, and the current data types remove many of the problems which the GEHR and CEN data types presented for XML-schema. There has been no attempt to support XML-DTD, since it has no type system, and cannot reliably be reasoned about in an object-oriented way.

To simplify implementation in all object-oriented formalisms, including IDL, programming languages and XML-schema, multiple inheritance has generally been avoided (where it is used, only onr branch corresponds to substitutability). Generic classes have been used, since they significantly clarify the model. Type genericity is available in Java, C#, Eiffel, C++, and some othre languages. For languages not having it, there is a well-known transformation from models containing generic classes to classes for non-generic types systems (see for example [3]).

Implementability in relational databases has also been considered, and appears relatively straightforward, since only the data view of the types needs to be represented. Many implementations are likely to use only a single String or XML string to represent each entire data instance, which significantly simplifies things.

2.3 Prior Work

Four other type systems for clinical data, namely the GEHR data types, the HL7 v3 data types, the CEN 13606 data item types, and the Corbamed data types were carefully scrutinised in order to ensure a) that all needed types were covered in the *open*EHR specification, and b) that data conversion will be possible. Concepts from all three are cross-referenced throughout this specification where possible.

Because the HL7v3 data type specification is a widely available and comprehensive specification for clinican data types, particular attention has been paid to incorporating its semantics, as well as fixing errors or shortcomings. While there are differences both in design approach and in detail, a significant debt must be recognised to the authors of this work, from which many ideas in the present specification were drawn. A detailed discussion is found under Comparison with HL7v3 Types on page 82.

3 Introduction

3.1 Overview

This specification describes a set of types suitable for use in scientific, clinical and related information structures. In order for such types to exist, a set of primitive types is assumed, namely `Integer`, `Real`, `Boolean`, `Character`, `Octet`, `String`, `List<T>`, `Set<T>`, and `Array<T>`. These have standard definitions in the OMG object model used in UML, OCL, and are available in almost all type systems. The exact assumptions are described in the *open*EHR Support Information Model. A number of symbolic definitions (similar to constants in programming) are also described in the Support IM.

The data types described here are named with the class prefix "`DV_`", and inherit from the class `DATA_VALUE`. They have two distinct uses in reference models. Firstly, they may be used as 'data values' in reference model structures wherever the `DATA_VALUE` class appears, for example, in the EHR Reference Model via the `ELEMENT`.*value* attribute. Additionally, specific subtypes of the data types described here can also be used as attribute types in other classes in reference models, such as date/times, coded terms and so on. The difference is that in the former case, *only* subtypes of `DATA_VALUE` may be used, whilst in the latter case, other types may be used as well, from the assumed set of basic types.

3.2 Package Structure

The package structure of the *open*EHR data types is illustrated in FIGURE 1.

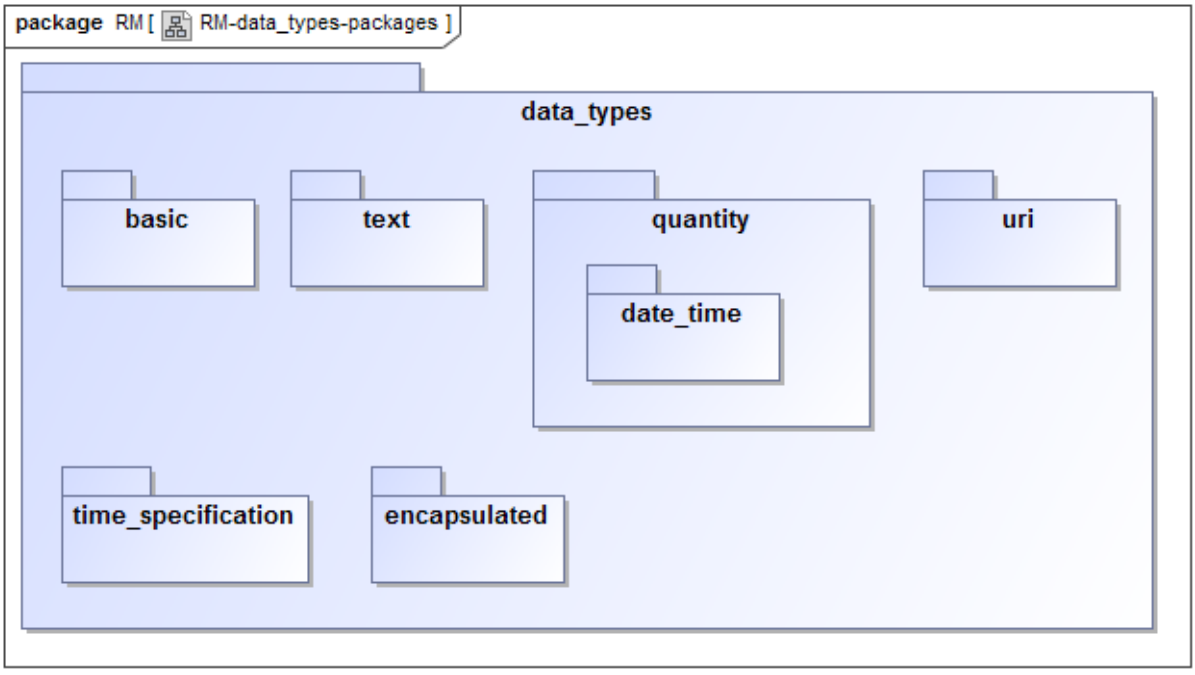


FIGURE 1 openehr.rm.data_types Package

4 Basic Package

4.1 Overview

The `data_types.basic` package, illustrated in FIGURE 2, contains types for the concepts of bi-state, state (in a state machine) and real-world entity identifiers (see the *open*EHR Common IM for a discussion on identifier types).

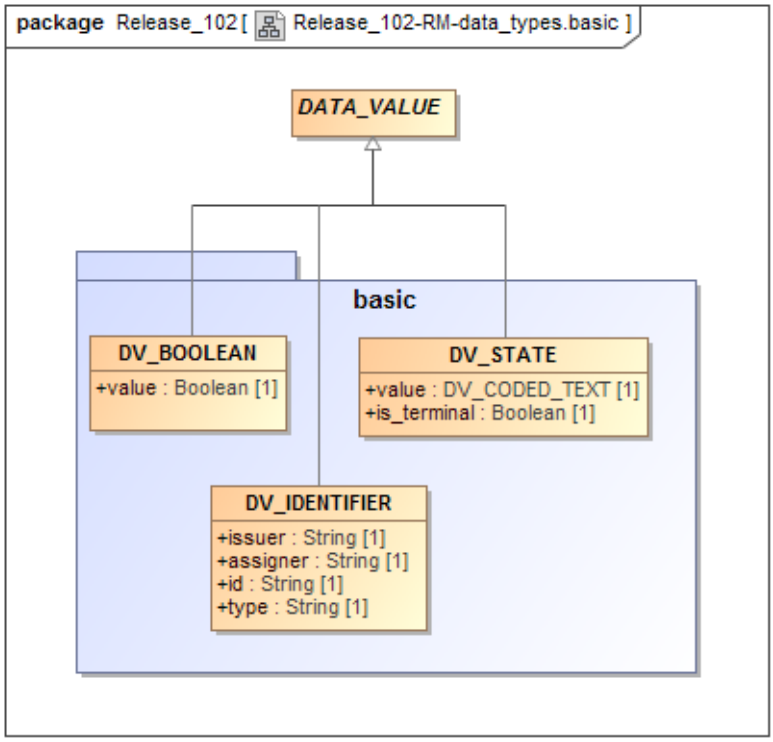


FIGURE 2 rm.data_types.basic Package

4.1.1 Requirements

Bi-state Values

One of the most basic types of data is boolean or bi-state data. The need here is for a type which both includes a boolean value, and which inherits from the type `DATA_VALUE`, enabling it to be used as an `ELEMENT`.*value*.

State Machine States

A type is required to represent state values of a state machine. In a simple system of data types, a simple integer would appear sufficient to perform this job. However, in an archetyped framework, a distinct type is required, so that it can be archetyped not by the constraints used for integers, but by a state machine definition instead. The type `DV_STATE` is provided for this purpose. An example of a state machine which models the lifecycle of a medication order is illustrated in FIGURE 3. This definition would appear in an archetype; the values of a `DV_STATE` object are then restricted to the values of the states in the definition.

Real-world Entity Identification

Real world entities (RWEs) such as people, car engines, invoices, and appointments may all be assigned identifiers. Although many of these are designed to be unique within a jurisdiction or issuing

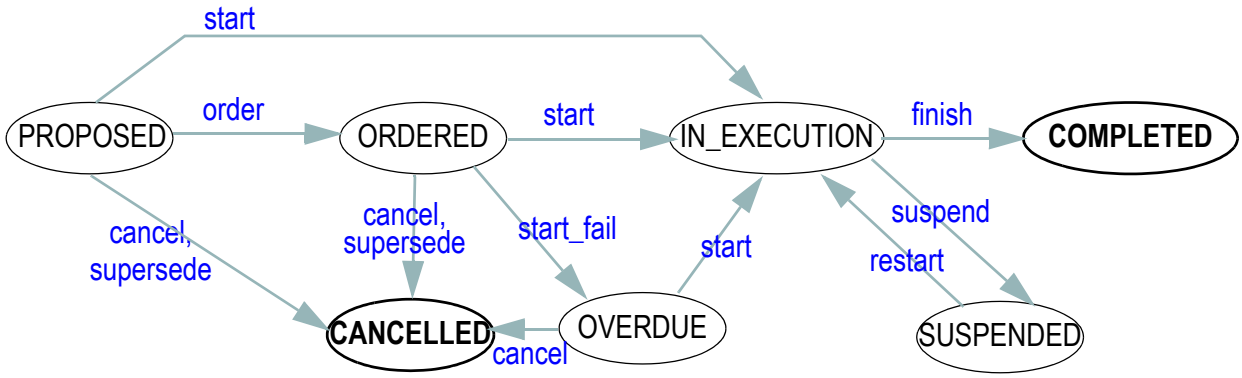


FIGURE 3 Example State Machine for Medication Orders

space, they are often not, due to data entry errors, bad design (ids which are too small or incorporate some non-unique characteristic of the identified entities), bad process (e.g. non-synchronised id issuing points); identity theft (e.g. via theft of documents of proof or hacking). In general, while some real world identifiers (RWIs) are "nearly unique", none can be guaranteed so. Therefore, from a strict computatoinal point of view, RWIs are not treated as reliable identifiers, but as attributes of their owning objects, in the same ways as names and addresses for example.

Examples of RWE identifiers which are intended to be unique within the space of the issuing authority or organisation include:

- driver's licence id
- social security number
- passport number
- prescription id

The defining logical characteristic of RWE ids is that they continue to identify the entities in question, regardless of how they change in time; for example a social security number does not change when someone changes their hair colour or even their gender (both of which attributes may be recorded in the database). In general it should be the case that if two RWE ids are equal, they refer to the same RWE.

At a practical level, RWE identifiers differ from information entity (IE) identifiers in that the former are generally not assigned by the computing infrastructure that uses them - that is to say, in the production computing system, such identifiers are no different from other characteristics of the entity, such as names or addresses.

4.1.2 Design

The model defined here in the `DV_IDENTIFIER` class allows the recording of three things as part of identifying an item of interest:

- the issuing authority of the kind of id used (e.g. it might be the federal department of health);
- the assigner of the id to the item being identified. This is usually the organisation which created the item being identified;
- the identifier given to the item of interest.

In addition, the type of item being identified can also be recorded.

See the Support IM specification for a further discussion of RWEs and IEs, and the definition of IEs in *open*EHR.

4.2 Class Descriptions

4.2.1 DATA_VALUE Class

CLASS	*DATA_VALUE (abstract)*
Purpose	Serves as a common ancestor of all data value types in *open*EHR models.
Invariants	

4.2.2 DV_BOOLEAN Class

CLASS	DV_BOOLEAN	
Purpose	Items which are truly boolean data, such as true/false or yes/no answers.	
Use	For such data, it is important to devise the meanings (usually questions in subjective data) carefully, so that the only allowed results are in fact true or false.	
MisUse	The `DV_BOOLEAN` class should not be used as a replacement for naively modelled enumerated types such as male/female etc. Such values should be coded, and in any case the enumeration often has more than two values.	
Inherit	`DATA_VALUE`	
Attributes	**Signature**	**Meaning**
1..1	**value**: `Boolean`	Boolean value of this item.
Invariants	***Value_exists***: value /= Void	

4.2.3 DV_STATE Class

CLASS	DV_STATE	
Purpose	For representing state values which obey a defined state machine, such as a variable representing the states of an instruction or care process.	
Use	`DV_STATE` is expressed as a `String` but its values are driven by archetype-defined state machines. This provides a powerful way of capturing stateful complex processes in simple data.	
Inherit	`DATA_VALUE`	
Attributes	**Signature**	**Meaning**

CLASS	DV_STATE	
1..1	**value**: DV_CODED_TEXT	The state name. State names are determined by a state/event table defined in archetypes, and coded using *open*EHR Terminology or local archetype terms, as specified by the archetype.
1..1	**is_terminal**: Boolean	Indicates whether this state is a terminal state, such as “aborted”, “completed” etc from which no further transitions are possible.
Invariants	***value_exists***: value /= Void ***Is_terminal_exists***: is_terminal /= Void	

4.2.4 DV_IDENTIFIER Class

CLASS	DV_IDENTIFIER	
Purpose	Type for representing identifiers of real-world entities. Typical identifiers include drivers licence number, social security number, vertans affairs number, prescription id, order id, and so on.	
Use	DV_IDENTIFIER is used to represent any identifier of a real thing, issued by some authority or agency.	
Misuse	DV_IDENTIFIER is not used to express identifiers generated by the infrastructure to refer to information items; the types OBJECT_ID and OBJECT_REF and subtypes are defined for this purpose.	
Inherit	DATA_VALUE	
Attributes	**Signature**	**Meaning**
1..1	**issuer**: String	Authority which issues the kind of id used in the *id* field of this object.
1..1	**assigner**: String	Organisation that assigned the id to the item being identified.
1..1	**id**: String	The identifier value. Often structured, according to the definition of the issuing authority’s rules.
1..1	**type**: String	The identifier type, such as “prescription”, or “SSN”. One day a controlled vocabulary might be possible for this.

CLASS	DV_IDENTIFIER
Invariants	***issuer_valid***: issuer /= Void **and then not** issuer.is_empty ***assigner_valid***: assigner /= Void **and then not** assigner.is_empty ***id_valid***: id /= Void **and then not** id.is_empty ***type_valid***: type /= Void **and then not** type.is_empty

5 Text Package

5.1 Overview

The `data_types.text` package contains classes for representing all textual values in the health record, including plain text, coded terms, and narrative text. It is illustrated in FIGURE 4.

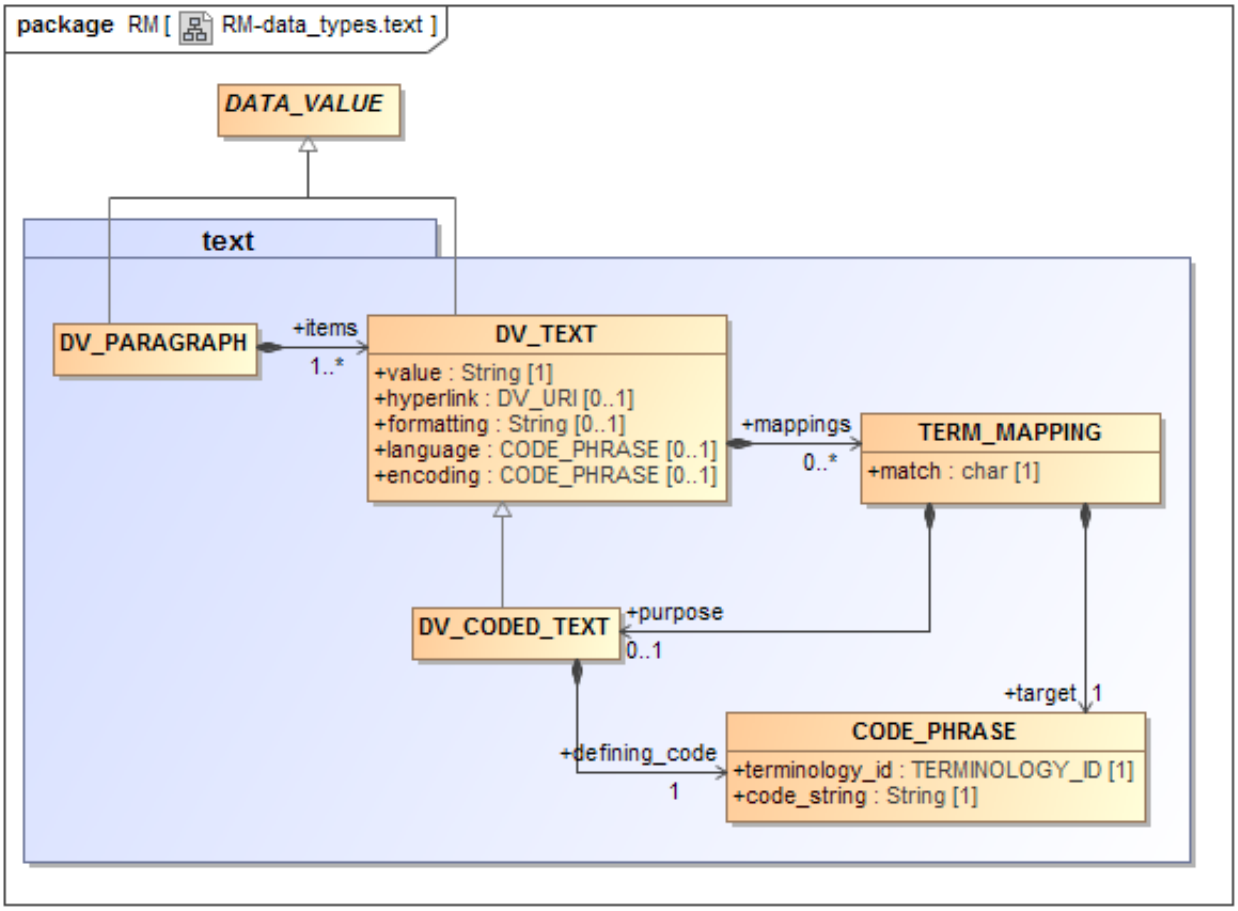


FIGURE 4 rm.data_types.text Package

5.1.1 Requirements

The sections below describe the requirements of text data types. Two overriding principles should be noted at the outset with regard to text.

1. Regardless of what terminologies are (or are not) available to the clinician and/or the software, the primary requirement is that in all cases clinicians are able to record exactly what they want to say. This means that if they want to record something very general, such as “cold” or a very specific term such as “Ross River virus infection” they should be able to, whether or not the appropriate coded terms are available. However, the facility should be available to additionally code any such textual item, at the time or indeed at some later time, so as to satisfy reporting or other needs.
2. It is assumed that any client of terminology, such as the EHR, uses a terminology service which provides a complete interface to the terminology. The design of the `DV_CODED_TEXT` type reflects this. Accordingly, there is no concept of

"post-coordination" outside the terminology environment allowed by the data types described here: the only thing that is available from the terminology service is a key which refers to a lexical entity, which may be a single term or a code phrase, and which may be part of a reference terminology and/or linked to element(s) of underlying ontologies. It is also assumed that there is no direct access to any particular terminology; access to all terminologies (whether simple coded lexicons or large semantic networks) is via the same abstract interface.

Terminology Ids are likely to be of various types.

1. Terminology_Id = "local": this constant value means that the origin of allowable values is described within the archetype. This is coded to allow translation. The local archetype then only needs the set of codes and the local translation. The archetype may contain a translation table if required.
2. Terminology_Id = "[authority]". This might be "openehr", "centc251", "hl7", etc;
3. The variant Terminology_Id = "[authority]:[Domain value set]" could also be supported, although it should not generally be necessary, since all codes should be unique within a given issuing authority. Examples might be "openehr:event math function", "hl7:gender";
4. Terminology_Id = "SNOMED-CT", "ICD10AM", etc. Idemtifiers of this kind must be unique values in an accepted set of terminologies from an authoritative source. These MUST be universally known. In *open*EHR, names from the US Natoinal Library of Medicine's UMLS terminology name list are used. See Support IM, `terminology` package for details.

5.1.1.1 Narrative Text

Narrative text items are used in the EHR in a number of cases, including:

- values of coded attributes in the reference model;
- recording of subjective or imprecise patient responses, particularly quantities or dates not deemed sufficiently precise to be represented using structured quantitative or date/time date types;
- recording of narrative statements, e.g. visual observations;
- recording of tracts of prose, e.g. overall findings and conclusions, prognoses;
- recording of values that would normally be coded, but for which no code and/or no terminology service is available.

While narrative text items themselves are not themselves coded, they may have code phrases associated with them, as described below under Mappings, and may be mixed within a paragraph with coded items.

5.1.1.2 Terminological Entities

Textual entities available in a terminology service are used in the health record to enable processing, from simple queries to decision support. Reasons for using terminology include the following.

- To guarantee interoperability of meaning. For instance, if the term "cold" is recorded in plain text, it could be interpreted as "feeling cold", "C.O.L.D" (chronic obstructive lung disease), "rhinorrhoea", "coryza" or "U.R.T.I. (upper respiriatory tract infection), among others. If, however, it is coded from a terminology such as ICD10 or SNOMED-CT, any party

reading the data (including software) knows the intention, since the meaning of the code in the terminology is unambiguous.

- To standardise textual renderings of terms and avoid informal shorthand. For example, practitioners wanting to write “systolic blood pressure” write things like “systolic BP”, “systolic bp”, “sys. BP.” and so on; use of coded terms ensures that such abbreviations are either avoided, or associated with an unambiguous meaning.
- For unambiguous naming of problems, medications or diagnoses for support of knowledge-based tools such as prescribing packages and other decision support applications.
- For standardised names of things in the record e.g. a heading of “Physical examination” or an entry such as “Differential diagnosis”.
- For finite sets of values (‘value sets’), e.g. Blood Group = ‘A|B|AB|O’.
- For classifying other data for the purpose of statistical studies, e.g. by putting ICD disease group classifiers on actual disease names entered in health records.

A basic requirement for interoperability of text items, coded using terms (i.e. where the text is the official rubric for the code), is that both the rubric and the code (or ‘code-phrase’) must be recorded, to ensure the originally intended text is retained for receivers of EHR information who do not have access to the terminologies used at the origin. However, where a terminology service *is* available, the key can be used to unambiguously locate the string value of the term, and can also be used to find translations in other languages. (Note that these comments do not apply to *mappings*, which are described below).

In some terminologies, there are semantic networks of links emanating from most coded terms, which classify them or relate them to other terms. Such links provide a means for decision support to make inferences about specific things found in the record. For instance if the term “leukaemia” is found, queries to the terminology service can be made in order to deduce that the patient has both a “cancer” and a “disease of the immune system” (assuming leukaemia is classified under these more general terms in the terminology).

This specification assumes the existence of a terminology service which is responsible for interrogating actual terminologies and performing *validated coordination* of terms, i.e. creating combinations deemed valid by the underlying source terminology, potentially without even assigning a new code to the result. All validated coordination is carried out *inside* the terminology service, and any “term” made available by the service is already ‘coordinated’. The difference between ‘pre-coordinated’ and ‘post-coordinated’ terms is that the former have a single code, whereas the latter have a code phrase, or expression that is interpretable by the terminology. For example, the coordination “foot, left” (a shorthand way of writing the relationship “foot has-laterality left”) could be created by the terminology service from the source terms “foot”, “left” and “has-laterality” from a terminology such as SNOMED. Any such coordination must be valid within the source terminology, i.e. correspond to valid relationships defined therein.

The class `DV_CODED_TEXT` described here captures the association of two things:

- the code phrase of a code phrase provided by the terminology service, recorded in the *defining_code* attribute.
- the text rubric of the code phrase, recorded in the *value* attribute (inherited from `DV_TEXT`);

The class `CODE_PHRASE`.*code_string* records a key, in the form of arguments to some retrieval function in the terminology service interface.

The semantics attached to coordinations of terms may differ. Two categories of coordination described in the literature are ‘qualification’ and ‘modification’. A common definition of the first is that “qualification narrows meaning” - i.e. creates a new term whose possible real world instances are within the set denoted by the original root term. Modification on the other hand changes the meaning of a root term. Various cases are described below under Meaning Modification. Both coordination types are assumed to be managed by the terminology service.

Coded terms may also be *mapped* to terms from other terminologies, which may be intended as *equivalents*, *classifiers*, or something in between. The section below on Mappings deals with these.

5.1.2 Design

All atomic text items are either instances of the type `DV_TEXT` or of `DV_CODED_TEXT`. The former allows the expression of text with optional formatting and hyperlinking. The latter additionally connects the text value to a key in the terminology service, with the implication that the key refers to a terminological entity lexically and semantically identical to the text value.

The model of `DV_CODED_TEXT` is designed to capture the actual coded term chosen by the user or software at runtime; it is implicitly assumed that this includes whichever synonym (term of equivalent meaning from the same terminology) was chosen, for terminologies supporting synonyms, and any coordination of underlying distinct terms. A `DV_CODED_TEXT` instance can only be used if the final textual value chosen by the user is lexically identical to the rubric returned by the terminology service for the key; if the user makes even the slightest change, the identity of rubric / key is lost, and a mapping (see Mappings on page 26) should be used instead.

The type `DV_TEXT` should be used wherever a coded or non-coded text item is allowed, while the type `DV_CODED_TEXT` should be used wherever a text item must be coded.

The type `DV_PARAGRAPH` allows larger tracts of text to be built up from lists of `DV_TEXT` instances, i.e. instances of `DV_TEXT` and `DV_CODED_TEXT`, as illustrated in FIGURE 5.

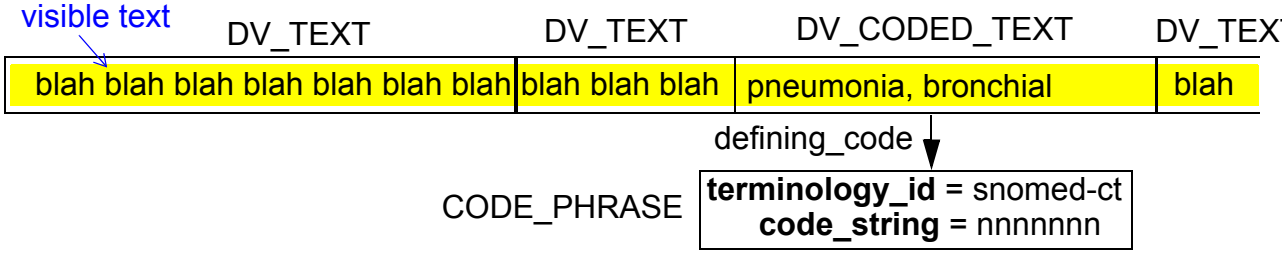


FIGURE 5 A DV_PARAGRAPH

5.1.3 Qualification

Qualification is the process of making a term more specific through the post-coordination of additional terms. It occurs when a terminology defines relationships between a primary term and other terms that qualify the primary. For example a coordination using the term “bronchitis” which creates a qualified term might be “acute bronchitis”; all real world instances of the latter are also instances of the former.

5.1.4 Meaning Modification

Terms that change the meaning of other terms are often known as “modifiers”. The difference between modification and qualification is that modifiers change the meaning so that the modifed term

as a whole does *not* refer to instances of the unmodified term. We describe below the particular types of modifiers and how they are represented using the text data types.

5.1.4.1 Mode-changing Terms

One class of modifers is exemplified by the addition of words like “risk of”, “fear of”, “history of” and so on. These are sometimes called *mode-changing* terms, since they change the “mode” of the root term from the present to the past (“history of”), a potential future (“risk of”) or some other alternate reality. Terms which are modified in this way should never be matched in queries searching for the root term; for example, a query for “coronary diease” (of the patient) should not match “family history of coronary disease”.

5.1.4.2 Context Sensitivity

There are many terms whose meaning is changed by the context in which they are stated, such as within a certain kind of note or test result. Consider the following:

- a blood sugar level after a 75gm oral loading has a different meaning than a fasting blood sugar;
- a systolic blood pressure in the pulmonary artery has a different meaning than a systemic arterial blood pressure;
- “total hip replacement” in the context of a “planned procedure";
- “meningitis” in the context of a “differential diagnosis”.

5.1.4.3 Negation

Negation is a special kind of mode change and has been a serious design challenge in the past, because modifiers like “not” or “no” only make sense when attached to some terms, and create nonsensical values or ambiguities by arbitrarily association with other terms.

5.1.4.4 Representation of Meaning-Modifying Terms

Rather than provide explicit features for representing modifier terms within `DV_CODED_TEXT`, the general principle underlying representation of all post-coordinations other than qualifications, is that a higher-level, archetyped structure such as an `ENTRY` (defined in the EHR RM), is a *minimal indivisible unit of information*. Such higher-level entities can have internal structure, and it is possible (and desirable) to achieve the effect of combinations of terms through this structure. In the case of `ENTRY`, it will be via structuring of `CLUSTER/ELEMENT` objects. The general rule is: to obtain the full meaning of any terms found in the record, all of the node names in any `ENTRY` (coded or not) must be considered from the root to the relevant leaf. Conversely, the “final” meaning of any term in the record cannot be known in isolation from the rest of the terms in the structure.

Accordingly, the concept “family history of coronary disease” is represented as an `ENTRY` whose root is named (for example) “subject family history”, and which includes further structure, which may be in greater of lesser detail; the coded term “coronary disease” would appear somewhere in this structure. The actual structure is completely defined by appropriate archetypes. Contrary to some perceptions, there is no general way to represent concepts such as “family history of coronary disease”, since it will vary depending on how much detail is recorded. Where some GPs routinely record just the simplest form, others may record the details of which family members had heart problems and exactly what they were.

The same approach is used for context-dependent terms. Archetypes defining contexts such as “planned procedures” or “differential diagnosis” will use these terms as their root nodes; as a result, the meaning of any term appearing below the root can only be understood by including the root. Once

again, the exact structures are completely dependent upon archetyping, and may be simple or quite sophisticated.

Negations are more complex than might first be apparent and are best handled by good archetype design. Terminologies might provide a term such as “No known allergies” which is helpful. But if someone has an allergy of some sort, the medicolegal requirement might be to record that the person has no known allergies to penicillin or another class of medication that is being prescribed. The often-proposed approach of using a generic negation ‘modifier’ to deal with such issues results in further problems. Consider the use of negation with liver - “no liver”, “no palpable liver”, “no liver disease”, “no history of liver disease”, “no liver function”, “no liver function tests”. The meaning of negated terms may be non-sensical and difficult to interpret.

A basic principle of dealing with negatives is to realise that most naïve suggested use cases are quite ambiguous as stated. Does “no allergies” mean “no reported episode of allergy”, “no allergic reactions ever”, “no known allergies to medication” or something else? Does it mean that these statements are taken as given by the patient, or determined by tests? Like all medical phenomena, allergies must be described in some detail for the EHR to be of any real use. Almost inevitably, this precludes the use of negated terms. Since the actual information structure will be determined in advance by archetype designers, clinicians will almost never be in the situation of having to negate a term. However, if the need does arise, it should be dealt with by a negative or quantitative *answer*, i.e. a value rather than a name. For example, in any `ENTRY` describing current problems, the clinician may record the name/value pair “allergies: NONE”. Here, “allergies” will be a `DV_CODED_TEXT`, and “NONE” will be either a `DV_CODED_TEXT` or a `DV_TEXT`; the two will be associated by a containing object, such as an instance of the `ELEMENT` class from the EHR RM. There is no explicit model of negation in *open*EHR.

5.1.5 Mappings

In a number of circumstances, both plain text and coded text items are mapped to terms from other terminologies. In theory, this should never occur, since it means that relationships between terms which should only be knowable in the knowledge base (in the form of the terminology service, or something else) are being created and transmitted as part of EHR information, potentially invalidating or overriding the knowledge base. Where mappings are required, the proper approach is to create thesauri within the knowledge environment, and map through them. Unfortunately, in some cases, activities in the real world do not respect the information/knowledge boundary, hence the model described here includes an explicit *mapping* concept, which itself includes a “purpose” and a “match” indicator. Matching corresponds to the categories described below.

5.1.5.1 Classification (Broader Terms)

Any text item, whether coded or not, may be *classified* with a coded term, for research, reporting and decision support purposes. For example, a GP working in tropical Australia may wish to write “Ross River infection”, and be working with ICD9, which does not contain this term (although ICD9-CM does). He or she will use a plain text item, but will still be able to map it to an ICD9 classifier, such as the code for “arbovirus infection NOS”. The same approach can be used for adding a classifying term to a coded text item. The utility of classifier terms is various: they allow decision support to make more powerful inferences; in situations where the available terminologies do not provide the classification inbuilt, and where it is known that not all users of EHR data will have terminologies available. In data terms, classification mapping can be visualised as illustrated in FIGURE 6.

Classifying mappings are represented by adding a term to the *mappings* list of the original term. Each mapping is explicitly represented with an instance of `TERM_MAPPING`, which indicates both the term being associated with the original text item, and a value of ‘>’ for the *match* attribute, which indicates

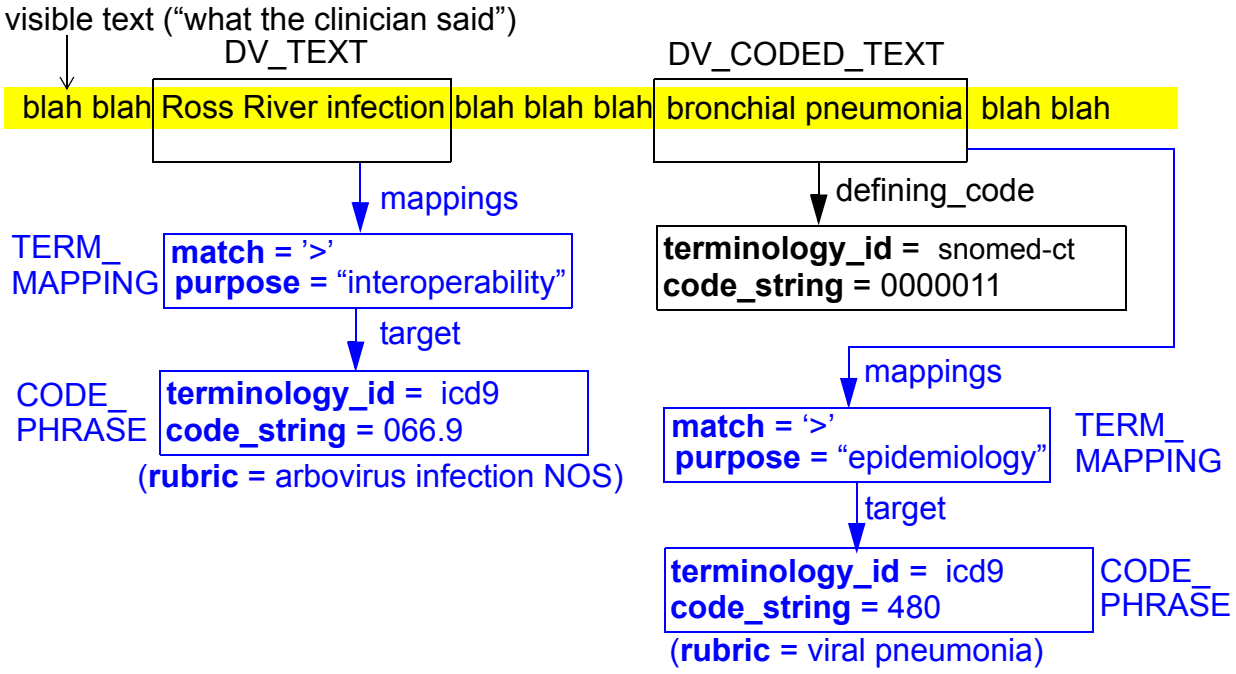


FIGURE 6 Plain Text and Coded Text with Classifier(s)

that the mapping is "broader". The possible values of the *match* attribute are '>' (broader), '<' (narrower), and '=' (equivalent); they are taken from the ISO standards 2788 ("Guide to Establishment and development of monolingual thesauri") and 5964 ("Guide to Establishment and development of multilingual thesauri").

5.1.5.2 Equivalent / Synonymous Terms

Data from pathology laboratories has often been coded using a terminology local to the laboratory, due to lack of or economic unfeasibility of using existing widespread terminologies for the job. However, some laboratories also supply a nearest *equivalent* code from a well-known terminology such as LOINC, to enable the receiver of the data to process it in a more standard fashion. Here, "equivalence" is taken to mean a term of the same meaning but from a different vocabulary.

Another instance where equivalent terms might be supplied is to effect the translation of terms across specialist vocabularies such as nursing vocabularies when sharing EHRs across jurisdictions.

In theory, the cleanest way for senders and receivers of data coded with both a local and a more standard equivalent to deal with the mapping problem is for the originator of the local terminology to provide a complete thesaurus of translations into one or more recognised terminologies. However, in practice, laboratories using the HL7 v2.x messaging standard usually encode a primary term and equivalents with the HL7 CE data type, meaning that equivalents are included only with the term they are used with. A similar pragmatic approach to mapping equivalent terms in the EHR is likely to be used with the data types described here, and can be effected with the same mapping approach as for classification.

A further situation in which text values - this time plain text - is mapped to equivalent terms is when natural language processing is used to generate coded terms for existing free-text prose. The aim of such processing is to detect word phrases and associate them with a coded term of the same meaning, without obliterating the original text. In terms of the model described here, a `CODE_PHRASE` is associated with a `DV_TEXT` instance via the *mappings* attribute.

In all cases with equivalents, the value of the *match* attribute is '=', indicating that the mapping is a synonym.

5.1.5.3 More Specific Mappings (Narrower Terms)

Occasionally, there is a need to create a mapping to a term of narrower meaning than the original text item. Circumstances in which this occurs include when a clinician wants to record a syndrome such as "croup" or "influenza", but the terminology does not contain these general terms, although it does contain more specific terms, e.g. "viral laryngo-tracheitis" or "influenza type A". Clearly the clinician should be allowed to record what he/she wants (as plain text if necessary), but it should also be possible to add a mapping to the more precise term. For mappings to narrower terms, the value of the *match* attribute is '<'.

5.1.5.4 The Unified Medical Language System (UMLS)

It has been argued in GEHR [14] that UMLS reference terms should also be supplied with occurrences of coded terms, in the form of the UMLS concept unique identifier, or "CUI". UMLS is a way of encoding terms developed at the National Library of Medicine in the United States, and consists of a meta-thesaurus, in which terms from any extant term set (such as ICD, SNOMED, READ) can be cross-referenced. UMLS CUIs could turn out to be extremely useful for decision support and reporting.

The proper use of UMLS is that terms from particular terminologies are passed to a UMLS interface and a CUI + rubric received in response. However, the mapping approach described above could also be used to map UMLS CUIs to existing text or terms in an EHR; in this case, a `DV_CODED_TEXT` is constructed for each UMLS "term", where the code is the CUI and the rubric is the text rendering of the CUI (guaranteed unique in UMLS). The same approach can be used for any other thesaurus which becomes available in the future.

5.1.5.5 Legacy Mapping Scenarios

In cases where legacy data has to be converted to *open*EHR-compliant data, and only codes are available, e.g. ICD or ICPC codes, the following approach is recommended:

- create a new `DV_TEXT` whose value is "(not available)"
- add a *mapping* to the `DV_TEXT`, with:
  - *purpose* = "legacy conversion"
  - *match* = "="
  - *target* = `CODE_PHRASE` object whose *code_string* and *terminology_id* are set to correspond to the available code in the legacy data.

This expresses the reality that no text was ever recorded in the legacy system; rather a code was recorded directly in the data field. In the converted data, this code is more correctly considered a mapping.

5.1.6 Language Translations

In most cases the natural language of a text object is known from the enclosing Entry (i.e. Observation) or other enclosing context. Where it is different (e.g. a german sentence within an English language diagnosis), or there is no enclosing context, the `DV_TEXT`.*language* attribute can be set to indicate the language of the text item.

5.2 Class Descriptions

5.2.1 DV_TEXT Class

CLASS	DV_TEXT	
Purpose	A text item, which may contain any amount of legal characters arranged as e.g. words, sentences etc (i.e. one DV_TEXT may be more than one word). Visual formatting and hyperlinks may be included. A DV_TEXT can be "coded" by adding mappings to it.	
Use	Fragments of text, whether coded or not are used on their own as values, or to make up larger tracts of text which may be marked up in some way, eventually going to make up paragraphs.	
Inherit	DATA_VALUE	
Attributes	**Signature**	**Meaning**
1..1	**value**: String	Displayable rendition of the item, regardless of its underlying structure. For DV_CODED_TEXT, this is the rubric of the complete term as provided by the terminology service. No carriage returns, line feeds, or other non-printing characters permitted.
0..1	**mappings**: List <TERM_MAPPING>	terms from other terminologies most closely matching this term, typically used where the originator (e.g. pathology lab) of information uses a local terminology but also supplies one or more equivalents from well-known terminologies (e.g. LOINC).
0..1	**formatting**: String	A format string of the form "name:value; name:value...", e.g. "font-weight : bold; font-family : Arial; font-size : 12pt;". Values taken from W3C CSS2 properties lists "background" and "font".
0..1	**hyperlink**: DV_URI	Optional link sitting behind a section of plain text or coded term item.
0..1	**language**: CODE_PHRASE	Optional indicator of the localised language in which the value is written. Coded from *open*EHR Code Set "languages". Only used when either the text object is in a different language from the enclosing ENTRY, or else the text object is being used outside of an ENTRY or other enclosing structure which indicates the language.

CLASS	DV_TEXT	
0..1	**encoding**: `CODE_PHRASE`	Name of character encoding scheme in which this value is encoded. Coded from *open*EHR Code Set "character sets". Unicode is the default assumption in *open*EHR, with UTF-8 being the assumed encoding. This attribute allows for variations from these assumptions.
Invariants	***Value_valid***: value /= void **and then not** value.is_empty **and then not** (value.has(CR) ***or*** value.has(LF)) ***Language_valid***: language /= Void **implies** code_set(Code_set_id_languages).has_code(language) ***Encoding_valid***: encoding /= Void **implies** code_set(Code_set_id_character_sets).has_code(encoding) ***Mappings_valid***: mappings /= void **implies not** mappings.is_empty ***Formatting_valid***: formatting /= void **implies not** formatting.is_empty	

5.2.2 TERM_MAPPING Class

CLASS	TERM_MAPPING	
Purpose	Represents a coded term mapped to a DV_TEXT, and the relative match of the target term with respect to the mapped item. Plain or coded text items may appear in the EHR for which one or mappings in alternative terminologies are required. Mappings are only used to enable computer processing, so they can only be instances of DV_CODED_TEXT.	
Use	Used for adding classification terms (e.g. adding ICD classifiers to SNOMED descriptive terms), or mapping into equivalents in other terminologies (e.g. across nursing vocabularies).	
Attributes	**Signature**	**Meaning**
1..1	**target**: CODE_PHRASE	The target term of the mapping.
1..1	**match**: Character	The relative match of the target term with respect to the mapped text item. Result meanings: • '>': the mapping is to a broader term e.g. orginal text = "arbovirus infection", target = "viral infection" • '=': the mapping is to a (supposedly) equivalent to the original item • '<': the mapping is to a narrower term. e.g. original text = "diabetes", mapping = "diabetes mellitus". • '?': the kind of mapping is unknown. The first three values are taken from the ISO standards 2788 ("Guide to Establishment and development of monolingual thesauri") and 5964 ("Guide to Establishment and development of multilingual thesauri").
0..1	**purpose**: DV_CODED_TEXT	Purpose of the mapping e.g. "automated data mining", "billing", "interoperability"
Functions	**Signature**	**Meaning**
	narrower: Boolean ***ensure*** match = '<' ***implies*** Result	The mapping is to a narrower term.
	equivalent: Boolean ***ensure*** match = '=' ***implies*** Result	The mapping is to an equivalent term.

CLASS	TERM_MAPPING	
	broader : `Boolean` ***ensure*** match = '>' ***implies*** Result	The mapping is to a broader term.
	unknown : `Boolean` ***ensure*** match = '?' ***implies*** Result	The kind of mapping is unknown.
	is_valid_match_code(c: Character) : `Boolean` ***ensure*** ***Result*** := c = '>' ***or*** c = '=' ***or*** c = '<' or c = '?'	True if match valid.
Invariants	***Target_exists***: target /= Void ***Purpose_valid***: purpose /= Void ***implies*** terminology(Terminology_id_openehr). has_code_for_group_id(Group_id_term_mapping_purpose, purpose.defining_code) ***Match_valid***: is_valid_match_code(match)	

5.2.3 CODE_PHRASE Class

CLASS	CODE_PHRASE	
Purpose	A fully coordinated (i.e. all "coordination" has been performed) term from a terminology service (as distinct from a particular terminology).	
Attributes	**Signature**	**Meaning**
1..1	**terminology_id**: `TERMINOLOGY_ID`	Identifier of the distinct terminology from which the code_string (or its elements) was extracted.
1..1	**code_string**: `String`	The key used by the terminology service to identify a concept or coordination of concepts. This string is most likely parsable inside the terminology service, but nothing can be assumed about its syntax outside that context.
Invariants	***Terminology_id_exists***: terminology_id /= Void ***Code_string_exists***: code_string /= Void **and then not** code_string.is_empty	

5.2.4 DV_CODED_TEXT Class

CLASS	DV_CODED_TEXT	
Purpose	A text item whose *value* must be the rubric from a controlled terminology, the key (i.e. the 'code') of which is the *defining_code* attribute. In other words: a `DV_CODED_TEXT` is a combination of a `CODE_PHRASE` (effectively a code) and the rubric of that term, from a terminology service, in the language in which the data was authored.	
Use	Since `DV_CODED_TEXT` is a subtype of `DV_TEXT`, it can be used in place of it, effectively allowing the type `DV_TEXT` to mean "a text item, which may optionally be coded".	
Misuse	If the intention is to represent a term code attached in some way to a fragment of plain text, `DV_CODED_TEXT` should not be used; instead use a `DV_TEXT` and a `TERM_MAPPING` to a `CODE_PHRASE`.	
Inherit	`DV_TEXT`	
Attributes	**Signature**	**Meaning**
1..1	**defining_code**: `CODE_PHRASE`	The term which the 'value' attribute is the textual rendition (i.e. rubric) of.
Invariants	***Definition_exists***: defining_code /= ***Void***	

5.2.5 DV_PARAGRAPH Class

CLASS	DV_PARAGRAPH	
Purpose	A logical composite text value consisting of a series of DV_TEXTs, i.e. plain text (optionally coded) potentially with simple formatting, to form a larger tract of prose, which may be interpreted for display purposes as a paragraph.	
Use	DV_PARAGRAPH is the standard way for constructing longer text items in summaries, reports and so on.	
Inherit	DATA_VALUE	
Attributes	**Signature**	**Meaning**
1..1	**items**: List<DV_TEXT>	Items making up the paragraph, each of which is a text item (which may have its own formatting, and/or have hyperlinks).
Invariants	***items_exists***: items /= void **and then not** items.is_empty	

FIGURE 7 illustrates the visual appearance of a typical DV_PARAGRAPH.

xxxxx *xxxx* *xxx* xxxxxxx xx xx **xxxxxxx** xxx *xxxxxxxxx* xxxx
xxxxxxxxxxx xxxx **xxx xxxxxx** xxxxxxxxx *xxxxxx xxxxx* xxxxx xxxx
xxxxxx a xxxxxxx xxxx xxxxx *xxxxx xxxxxxx* xxxxxx x xxx

FIGURE 7 PARAGRAPH visual structure

6 Quantity Package

6.1 Overview

The `data_types.quantity` package is illustrated in FIGURE 8. Dates and Times are found in the next section.

6.1.1 Requirements

Ordinal Values

Medicine is one domain in which symbols representing relative magnitudes are commonly used, without exact values being known. The main purpose is usually to classify patients into groups for which different decisions might be made. Thus, while approximate ranges (technically speaking - "fuzzy intervals") might be stated (such as for a urinalysis), concrete values are not of interest, only categories are. Take for example the characterisation of pain as being "mild", "medium", "severe", or the reflex response to tendon percussion as "-", "+/-", "+", "++", "+++", "++++". There may be no way to scientifically precisely quantify such values because they reflect a subjective experience of the patient or informal judgement by clinician. However, they are understood as being ordered, e.g. "++" is 'greater than' "+".

Similarly, even though the symbolic values for haemolysed blood in a urinalysis have approximate ranges stated for them, as shown below, these 'values' are not usable in the same way as true quantities.

- "neg", "trace" (10 cells/μl)
- "small" (<25 cells/μl)
- "moderate" (<80 cells/μl)
- "large" (>200 cells/μl)

A second requirement for ordinal values is that in many cases there is a need to associate integer values with the symbols, in order to facilitate ordered comparison, and also to enable longitudinal comparison across results of the same kind (e.g. pain, protein). Integer values may be negative, 0 and positive, typically to allow the 0 value to correspond to a neutral value in a range.

[Note: an argument sometimes put forward for recording all ordinals in a more precise way is that comparisons might want to be made between the values quoted by two laboratories for the same symbol (e.g. "moderate"). There are a number of counter-arguments. Firstly, such comparisons are a poor attempt at normalisation, an activity which is the business of pathologists, not EHR users. Secondly, the symbolic values are often arrived at by the tester making a judgement of colour on a strip, which while an adequate (and cost-effective) approach for *classifying*, is not a valid means of *quantifying* a value. Lastly, in most cases, if a quantified point value or range is desired, or available, then it will be used - meaning that the appropriate quantitative data type can be used, rather than an the ordinal type.]

Countable Things

An common kind of data value in medicine is the dimensionless countable quantity, e.g. "number of doses: 2", "number of previous pregnancies: 1", "number of tablets: 3". Values of this type are always integral. Countable values need to be convertible to real numbers for statistical purposes, for example for a study of average number of pregnancies per couple.

Some countable entities such as tablets are divisible into major fractions, typically halves and occasionally quarters.

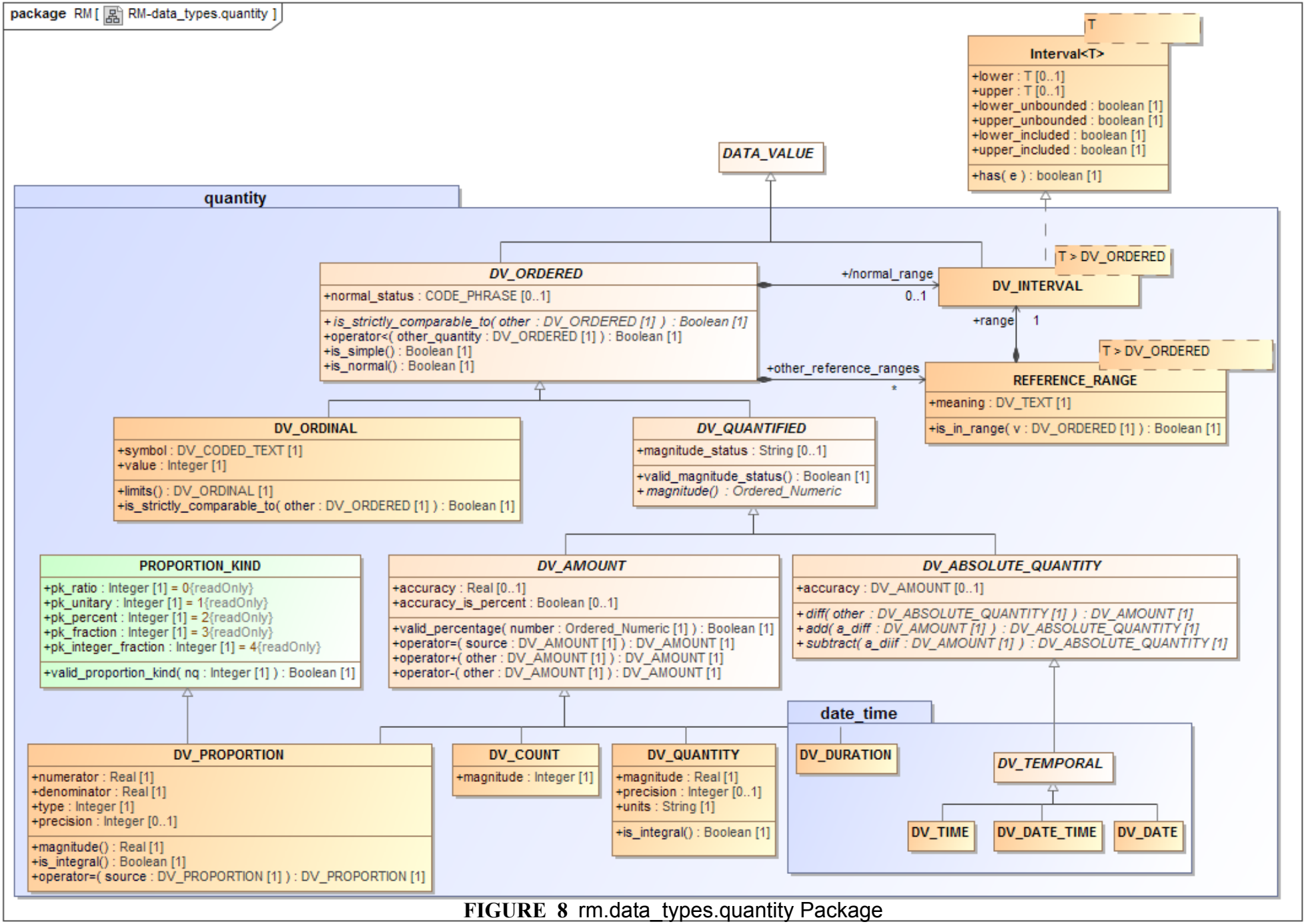


FIGURE 8 rm.data_types.quantity Package

Dimensioned Quantities

The most common kind of quantity is a measured, dimensioned quantity. Anything which is measurable (rather than countable) involves a number of data aspects, namely:

- a magnitude whose value is a real number;
- the physical property being measured, with the appropriate units;
- a concept of precision, i.e. to what number of decimal places the value is recorded;
- a concept of accuracy, i.e. the known or assumed error in the measurement due to instrumentation or human judgement.

Examples of dimensioned quantities include:

- systolic BP: 110 mmHg
- height: 178 cm
- rate of asthma attacks: 7 /week
- weight loss: 2.5 kg

Ratios and Proportions

A common quantitative type in science and medicine is the proportion, or ratio, which is used in situations like the following:

- 1:128 (a titer);
- Na:K concentration ratio (unitary denominator);
- albumin:creatinine ratio;
- % e.g. red cell distribution width (RDW) which is the width of a distribution of RBC widths.

In general ratios have real number values, even if many examples appear to be integer ratios. Proportions with unitary denominator and % (denominator = 100) are common.

Formulations

A concept superficially similar to proportions and ratios is formulations of materials, such as a solid in a liquid e.g.:

- 250 mg / 500 ml (solute/solvent)

Although a single solute/single solvent formulation appears to have the same form as a ratio, the general form is for any number of substances to be mixed together, usually according to a particular procedure. Formulations are therefore not candidates for direct modelling as fine-grained quantities, but instead are constructed by archetyping a higher-level structure, each leaf element of which contains the required kind of Quantity.

Quantity Ranges

Quantity ranges are ubiquitous in science and medicine, and may be defined for any kind of measured phenomenon. Examples include:

- healthy weight range, e.g. 48kg - 60kg
- normal range for urinalysis in pregnancy - protein, e.g. “nil” - “trace”

Reference Ranges

A reference range is a quantity range attached to a measured value, and is common for laboratory result values. The typical form of a reference range found in a pathology result indicates what is considered the ‘normal’ range for a measured value. Examples of reference ranges:

- normal range for serum Na is 135 - 145 mmol/L.
- desirable total cholesterol: < 5.5 mmol/L (strictly this probably should be 2.0 - 5.5 mmol/L, but is not usually quoted this way as low cholesterol is not considered a problem.)

Ranges can also be quoted for drug administrations, in which case they are usually thought of as the 'therapeutic' range. For example, the anticonvulsant drug Carbamazepine has a therapeutic range of 20 - 40 μMol/L. In some cases, there are multiple ranges associated with a drug, for example, Salicylate has a therapeutic range of 1.0 - 2.5 mmol/L and a toxic range > 3.6 mmol/L

Various examples occur in which multiple ranges may be stated, including the following.

- The administration recomendations for drugs which depend on the particular patient state. For example, the therapeutic range of Cyclosporin (an immunosuppresant) is a function of time post-transplant for the affected organ, e.g. kidney: < 6 months: 250 - 350 μg/L, > 6 months: 100 - 200 μg/L.
- Normal ranges for blood IgG, IgA, IgM which vary significantly with the age in months from birth.
- Progesterone and pituitary hormones have ranges which are different for different phases of the menstrual cycle and for menopause. This may result in 4 or 5 ranges given for one result. Only one will apply to any particular patient - but the exact phase of the cycle may be unknown - so the ranges may need to be associated with the value with no 'normal' range.

Where there are multiple ranges, the important question is: *which range information is relevant to the actual data being recorded for the patient*? In theory, only the range corresponding to the particular patient situation should be used, i.e. the range which applies after taking into account sex, age, smoking status, "professional athlete", organ transplanted, etc. In most cases, this is a single "normal" range, or a pair of ranges, typically "therapeutic" and "critical". However, practical factors complicate things. Firstly, data is sometimes supplied from pathology labs along with some or all of the applicable reference ranges, even though only some could possibly apply. This is particularly the case if the laboratory has no other data on the patient, and cannot evaluate which range applies. The requirement for faithfulness of recording might be extended to reference data supplied by laboratories, regardless of how irrelevant or arbitrarily chosen the reference data is, meaning that such data has to be stored in the record anyway. Secondly, there may be circumstances in which physicians want a number of reference ranges, even while knowing that only one range is applicable to the datum. Ranges above and below the relevant one might be useful to a physician wishing to determine how far out of range the datum is.

Normal Range and Status in Laboratory data

It is quite common for laboratories to include a normal range with each measured value, and/or a normal 'status', which indicates where the value lies with respect to the normal range. The latter will commonly take the form of markers like "HHH" (critically high), HH (abnormally high), H (borderline high), L, LL, LLL in HL7v2 messaging, although other schemes are undoubtedly used.

6.1.2 Design

Basic Semantics

In order to make sense of the requirements in a systematic way, a proper typology for quantities is needed. The most basic characteristic of all values typically called 'quantities' is that they are ordered, meaning that the operator "<" (less-than) is defined between any two values in the domain. An ancestor class for all quantities called `DV_ORDERED` is accordingly defined. This type is subtyped into ordinals and true quantities, represented by the classes `DV_ORDINAL` and `DV_QUANTIFIED`

respectively. DV_ORDINAL represents data values whose exact numeric values are not known, and which use symbolic renderings instead, such as “+”, “++”, “+++”, or “mild”, “medium”, “severe”. Each symbol can be assigned any integer value, providing a basis for computable comparison. In contrast, instances of DV_QUANTIFIED and all its subtypes have precise numeric magnitudes.

DV_QUANTIFIED itself introduces the concepts of *magnitude* and *magnitude_status*. The *magnitude* attribute is guaranteed to be available on any DV_QUANTIFIED, carrying the effective value, regardless of the particular subtype. The optional *magnitude_status* attribute can be used to provide a non-quantified indication of accuracy, and takes the following values:

- “=” : *magnitude* is a point value
- “<“ : value is < *magnitude*
- “>” : value is > *magnitude*
- “<=” : value is <= *magnitude*
- “>=” : value is >= *magnitude*
- “~” : value is approximately *magnitude*

If not present, meaning is “=”.

Logically, an *accuracy* attribute should also be included in DV_QUANTIFIED, but as its modelling is different in the subtypes in a way that does not easily lend itself to a common ancestor, it is only included in the subtypes.

The DV_QUANTIFIED class has two subtypes: DV_AMOUNT and DV_ABSOLUTE_QUANTITY. The former corresponds to relative ‘amounts’ of something, either a physical property(such as mass) or items (e.g. cigarettes). Mathematically, the ‘+’ and ‘-’ operators (as well as ‘*’ and ‘/’) are defined in the same way as for the real numbers (or any other mathematical ‘field’), with the semantics that adding two relative quantities measuring the same thing (i.e. with the same units) produces another relative quantity of the same kind; while the semantics of subtraction are that one relative quantity subtracted from another generates a third.

The second subtype of DV_QUANTIFIED, DV_ABSOLUTE_QUANTITY, models quantities whose values are absolute along a line having a defined origin. The main example of absolute quantities are the temporal concepts date, time and date/time. These are distinguished from relative quantities in that the normal addition and subtraction operations don’t apply. Instead, the semantics of such operators are based on the idea of the difference between absolute values being a relative amount. For example, two dates can be subtracted, but the result is a duration, not another date. For this reason, the operations *add*, *subtract* and *diff* are defined rather than ‘+’ or ‘-’. Date/time types, as well as the relative concept duration, are defined in the Date Time Package on page 58.

Subtypes of DV_AMOUNT are DV_PROPORTION, DV_QUANTITY, DV_COUNT, and DV_DURATION (see date_time package). The type DV_COUNT has an integer *magnitude* and is used to record naturally countable things such as number of previous pregnancies, number of steps taken by a recovering stroke victim and so on. There are no *units* or *precision* in a DV_COUNT. Countable quantities can be used to create instances of DV_QUANTITY, such as during a statistical study which average tobacco consumption over a time period. Such a computation might cause the creation of DV_QUANTITY objects representing values like {*magnitude* = 5.85, *units* = ‘/ week’}

DV_QUANTITY is used to represent amounts of measurable things, and has a real number *magnitude*, *precision*, *units* and *accuracy*. The *units* attribute contains the scientific unit in a parsable form defined by the Unified Code for Units of Measure (UCUM) [8]. A valid units string always implies a measured property, such as “force” or “pressure”. The property of a Quantity can conveniently con-

strained in archetypes, e.g. to “pressure”, which would allow any pressure unit. Unit strings can be compared to determine if they measure the same property (e.g. “bar” and “kPa” are both units corresponding to the property “pressure”), which enables the *is_strictly_comparable_to* function defined on DV_ORDERED to be properly specified on DV_QUANTITY.

It is **important to note** that while these semantics will allow comparison of e.g. two pressures recorded in mbar and mmHg, or even two accelerations whose units are “m.s^-2” and “m/s^2”, they provide no guarantee that this is a sensible thing to do in terms of domain semantics: comparing a blood pressure to an atmospheric pressure for example may or may not make any sense. It is not within the scope of the quantity package to express such semantics: this is up to application software which uses Quantities found in specific places in the data.

Accuracy and Uncertainty

Theoretically it might be argued that ‘accuracy’ should not be included in a model for quantified values, because it is an artifact of a measuring process and/or device, not of a quantity itself. For example, a weight of “82 kg +/-5%” can be represented in two parts. The “82 kg” is represented as a DV_QUANTITY, while the “+/-5%” could be included in the protocol description of the weighing instrument, since this is where the error comes from. For practical purposes however, (in)accuracy in a measured quantity corresponds to a range of possible values. In realistic computing in health, it is quite likely that the accuracy will be required in computations on quantities, especially for statistical population queries in which measurement error must be disambiguated from true correlation.

Accuracy is therefore introduced as the abstract feature *accuracy* of the DV_QUANTIFIED class. It is defined concretely in the two descendants, DV_AMOUNT, where it is of type Real, and DV_ABSOLUTE_QUANTITY, where it is of a differential type defined by subtypes. A value of 0 in either case indicates 100% accuracy, i.e. no error in measurement. Where accuracy is not recorded in a quantity, it is represented by a special value. In DV_AMOUNT, a value of -1 for the *accuracy* attribute is used for this purpose, and the constant *unknown_accuracy_value* = -1 is provided within the class to give a symbolic name for the special value. In the DV_ABSOLUTE_QUANTITY class, *accuracy_unknown* is represented by a Void (i.e. null) value for the *accuracy* attribute. An abstract Boolean feature *accuracy_unknown* is defined in the parent class DV_QUANTIFIED to provide a logical test of accuracy being absent, and is implemented in the respective descendants by concrete functions that check for the special values.

In addition, the class DV_AMOUNT, provides a feature *accuracy_is_percent*: Boolean to indicate if accuracy value is to be understood as a percentage, or an absolute value.

When two compatible quantities are added or subtracted using the + or - operators (DV_AMOUNT descendants) or add and substract (DV_ABSOLUTE_QUANTITY class), accuracy behaves in the following way:

- if accuracies are present in both quantities, they are added in the result, for both addition and subtraction operations;
- if either or both quantities has an unknown accuracy, the accuracy of the result is also unknown;
- if two DV_AMOUNT descendants are added or subtracted, and only one has *accuracy_is_percent* = True, accuracy is expressed in the result in the form used in the larger of the two quantities.

The related notion of ‘uncertainty’ is understood as a subjective judgement made by the clinician, indicating that he/she is not certain of a particular statement. It is not the same as accuracy: uncertainty may apply to non-quantified values, such as subjective statements, and it is not an aspect of

objective measurement processes, but of human confidence. Where the uncertainty is due to subjective memory e.g. “I think my grandfather was 56 when he died”, the uncertainty is simply recorded as another value, along with the main data item being recorded. Uncertainty is therefore not directly modelled in the *open*EHR data types, but appears instead in particular archetypes.

Quantity Ranges

Ranges are modelled by the generic type DV_INTERVAL<T:DV_ORDERED> which enables a range of any of the other quantity types (except ratio) to be constructed. This allows any subtype of DV_ORDERED to occur as a range as well.

Proportions

The DV_PROPORTION type is provided for representing true ratios, i.e. *relative* values, and consists of *numerator* and *denominator* Real values, and a *magnitude* function which is computed as the result of the numerator/denominator division. The *type* attribute is used to indicate the logical type of the proportion. Supported types include:

- *percent*: denominator is 100; usual presentation is “numerator %”
- *unitary*: denominator is 1; usual presentation is “numerator”
- *fraction*: numerator and denominator are both integer values; usual presentation is n/d, e.g. such as ½ or ¾, 1/2, 3/4 etc;
- *integer_fraction*: numerator and denominator are both integer values; usual presentation is n/d; if numerator > denominator, display as “a b/c”, i.e. the integer part followed by the remaining fraction part, e.g. 1½; this is the most likely form for expressing a number of tablets;
- *ratio*: numerator and denominator can take any value; usual presentation is “numerator:denominator”

Lastly, the *is_integral* function indicates that the numerator and denominator are both integer values; this is used for fractions (the *fraction* and *integer_fraction* types above) and other commonly occurring ratios where both parts are always integer values.

Normal and Reference Ranges

Normal range for any of the quantity types (i.e. any instance of a subtype of DV_ORDERED) can be included via the attribute DV_ORDERED.*normal_range*, of type REFERENCE_RANGE. Other reference ranges (e.g. sub-critical, critical etc) can be included via the attribute DV_ORDERED.*other_reference_ranges*. The separation of normal and other reference range attributes is used because the former constitute the vast majority of ranges quoted for quantitative data.

Normal status can be included via the attribute DV_ORDERED.*normal_status*, which takes the form of a DV_ORDINAL, whose *symbol* attribute is coded according to the *open*EHR terminology group “normal status”, and takes values “HHH” (critically high), “HH” (abnormally high), “H” (borderline high)”, “N” (normal), “L” ... “LLL”.

Recording Time

Time can be recorded in two ways. Absolute times in the social time domain, such as dates and time of day are recorded using the types in the date_time package. Fine-grained ‘time’, which is a duration rather than a time, is recorded using a DV_QUANTITY with units = ‘s’ or another temporal unit (‘h’, ‘ms’, ‘ns’ etc).

6.2 Class Descriptions

6.2.1 DV_ORDERED Class

CLASS	*DV_ORDERED (abstract)*	
Purpose	Abstract class defining the concept of ordered values, which includes ordinals as well as true quantities. It defines the functions '<' and *is_strictly_comparable_to*, the latter of which must evaluate to True for instances being compared with the '<' function, or used as limits in the `DV_INTERVAL<T>` class.	
Use	Data value types which are to be used as limits in the `DV_INTERVAL<T>` class must inherit from this class, and implement the function *is_strictly_comparable_to* to ensure that instances compare meaningfully. For example, instances of `DV_QUANTITY` can only be compared if they measure the same kind of physical quantity.	
Inherit	`DATA_VALUE, Ordered`	
Abstract	**Signature**	**Meaning**
	infix '<' **(**other**:** ***like*** Current**)**: `Boolean` ***require*** is_strictly_comparable_to(other)	Tests if this item is less than other, which must be of the same concrete type.
	is_strictly_comparable_to **(**other**:** ***like*** Current**)**: `Boolean`	Test if two instances are strictly comparable.
Attributes	**Signature**	**Meaning**
0..1	**normal_range**: `DV_INTERVAL<`***like*** Current>	Optional normal range.
0..1	**other_reference_ranges**: `List <REFERENCE_RANGE<`***like*** Current>>	Optional tagged other reference ranges for this value in its particular measurement context
0..1	**normal_status**: `CODE_PHRASE`	Optional normal status indicator of value with respect to normal range for this value. Often included by lab, even if the normal range itself is not included. Coded by ordinals in series HHH, HH, H, (nothing), L, LL, LLL; see *open*EHR terminology group "normal status".
Functions	**Signature**	**Meaning**

CLASS	*DV_ORDERED (abstract)*	
	is_normal: `Boolean` ***require*** normal_range /= Void or normal_status /= Void ***ensure*** normal_range /= Void **implies** Result = normal_range.has(Current) normal_status /= Void **implies** normal_status.code_string.is_equal(“N”)	Value is in the normal range, determined by comparison of the value to the *normal_range* if present, or by the *normal_status* marker if present.
	is_simple: `Boolean`	True if this quantity has no reference ranges.
Invariants	***Other_reference_ranges_validity***: other_reference_ranges /= Void **implies not** other_reference_ranges.is_empty ***Is_simple_validity***: (normal_range = Void **and** other_reference_ranges = Void) **implies** is_simple ***Normal_status_validity***: normal_status /= Void ***implies*** code_set(Code_set_id_normal_statuses).has_code(normal_status) ***Normal_range_and_status_consistency***: (normal_range /= Void **and** normal_status /= Void) **implies** (normal_status.code_string.is_equal(“N”) **xor not** normal_range.has(Current))	

6.2.2 DV_INTERVAL<T : DV_ORDERED> Class

CLASS	DV_INTERVAL<T : DV_ORDERED>
Purpose	Generic class defining an interval (i.e. range) of a comparable type. An interval is a contiguous subrange of a comparable base type.
Use	Used to define intervals of dates, times, quantities (whose units match) and so on. The type parameter, T, must be a descendant of the type `DV_ORDERED`, which is necessary (but not sufficient) for instances to be compared (*strictly_comparable* is also needed). Without the `DV_INTERVAL` class, quite a few more `DV_` classes would be needed to express logical intervals, namely interval versions of all the date/time classes, and of quantity classes. Further, it allows the semantics of intervals to be stated in one place unequivocally, including the conditions for strict comparison. The basic semantics are derived from the class `INTERVAL<T>`, described in the support RM.
Inherit	`DATA_VALUE`, `INTERVAL<T>`
Invariants	***Limits_consistent***: (not upper_unbounded ***and not*** lower_unbounded) ***implies*** (lower.is_strictly_comparable_to(upper) ***and*** lower <= upper)

6.2.3 REFERENCE_RANGE<T:DV_ORDERED> Class

CLASS	REFERENCE_RANGE<T:DV_ORDERED>	
Purpose	Defines a named range to be associated with any `ORDERED` datum. Each such range is particular to the patient and context, e.g. sex, age, and any other factor which affects ranges.	
Use	May be used to represent normal, therapeutic, dangerous, critical etc ranges.	
Attributes	**Signature**	**Meaning**
1..1	**meaning**: `DV_TEXT`	Term whose value indicates the meaning of this range, e.g. “normal”, “critical”, “therapeutic” etc.
1..1	**range**: `DV_INTERVAL<T>`	The data range for this meaning, e.g. “critical” etc.
Functions	**Signature**	**Meaning**
	is_in_range (val**:** `T`**)**: `Boolean`	Indicates if the value ‘val’ is inside the range

CLASS	REFERENCE_RANGE<T:DV_ORDERED>
Invariants	***Meaning_exists***: meaning /= Void ***Range_exists***: range /= Void ***Range_is_simple***: (range.lower_unbounded ***or else*** range.lower.is_simple) ***and*** (range.upper_unbounded ***or else*** range.upper.is_simple)

6.2.4 DV_ORDINAL Class

CLASS	DV_ORDINAL	
Purpose	Models rankings and scores, e.g. pain, Apgar values, etc, where there is a) implied ordering, b) no implication that the distance between each value is constant, and c) the total number of values is finite. Note that although the term 'ordinal' in mathematics means natural numbers only, here any integer is allowed, since negative and zero values are often used by medical professionals for values around a neutral point. Examples of sets of ordinal values: -3, -2, -1, 0, 1, 2, 3 -- reflex response values 0, 1, 2 -- Apgar values	
Use	Used for recording any clinical datum which is customarily recorded using symbolic values. Example: the results on a urinalysis strip, e.g. `{neg, trace, +, ++, +++}` are used for leucocytes, protein, nitrites etc; for non-haemolysed blood `{neg, trace, moderate}`; for haemolysed blood `{neg, trace, small, moderate, large}`.	
Inherit	`DV_ORDERED`	
Attributes	**Signature**	**Meaning**
1..1	**value**: `Integer`	Value in ordered enumeration of values. Any integer value can be used.
1..1	**symbol**: `DV_CODED_TEXT`	Coded textual representation of this value in the enumeration, which may be strings made from "+" symbols, or other enumerations of terms such as "mild", "moderate", "severe", or even the same number series as the values, e.g. "1", "2", "3". Codes come from archetype.
Functions	**Signature**	**Meaning**
	limits: `REFERENCE_RANGE <DV_ORDINAL>`	limits of the ordinal enumeration, to allow comparison of an ordinal value to its limits.
	infix '<' (other**:** ***like*** Current**)**: `Boolean` ***ensure*** value < other.value ***implies*** Result	True if types are the same and values compare
	is_strictly_comparable_to (other**:** ***like*** Current**)**: `Boolean` ***ensure*** symbol.is_comparable (other.symbol) ***implies*** Result	True if symbols come from same vocabulary, assuming the vocabulary is a subset or value range, e.g. "urine:protein".

CLASS	DV_ORDINAL
Invariants	***Symbol_exists***: symbol /= Void ***Limits_valid***: limits /= Void **and then** limits.meaning.is_equal("limits") ***Reference_range_valid***: other_reference_ranges /= Void **and then** other_reference_ranges.has(limits)

6.2.5 DV_QUANTIFIED Class

CLASS	*DV_QUANTIFIED (abstract)*	
Purpose	Abstract class defining the concept of true quantified values, i.e. values which are not only ordered, but which have a precise magnitude.	
Inherit	`DV_ORDERED`	
Abstract	**Signature**	**Meaning**
1..1	***magnitude***: *`Ordered_Numeric`*	Numeric value of the quantity in canonical (i.e. single value) form. Implemented as constant, function or attribute in subtypes as appropriate. The type Ordered_numeric is mapped to the available appropriate type in each implementation technology.
0..1	***accuracy***: *`Any`*	Accuracy of measurement, indicating error in the recording method or instrument. Implemented in subtypes. A logical value of 0 indicates 100% accuracy, i.e. no error.
1..1	***accuracy_unknown***: `Boolean`	True if accuracy is not known, e.g. due to not being recorded or discernable.
Attributes	**Signature**	**Meaning**
0..1	**magnitude_status**: `String`	Optional status of *magnitude* with values: • “=” : *magnitude* is a point value • “<“ : value is < *magnitude* • “>” : value is > *magnitude* • “<=” : value is <= *magnitude* • “>=” : value is >= *magnitude* • “~” : value is approximately *magnitude* If not present, meaning is “=”.
Functions	**Signature**	**Meaning**
	valid_magnitude_status (s: `String`): `Boolean` **ensure** Result = s.is_equal(“=”) **or** s.is_equal(“<”) **or** s.is_equal(“>”) **or** s.is_equal(“<=”) **or** s.is_equal(“>=”) **or** s.is_equal(“~”)	Test whether a string value is one of the valid values for the *magnitude_status* attribute.

CLASS	*DV_QUANTIFIED (abstract)*
Invariants	***Magnitude_exists***: magnitude /= Void ***Magnitude_status_valid***: magnitude_status /= Void **implies** valid_magnitude_status(magnitude_status)

6.2.6 DV_AMOUNT Class

CLASS	*DV_AMOUNT (abstract)*	
Purpose	Abstract class defining the concept of relative quantified 'amounts'. For relative quantities, the '+' and '-' operators are defined (unlike descendants of DV_ABSOLUTE_QUANTITY, such as the date/time types).	
Inherit	DV_QUANTIFIED	
Constants	**Signature**	**Meaning**
	unknown_accuracy_value: Real = -1.0	Special value for *accuracy* indicating no accuracy recorded.
Abstract	**Signature**	**Meaning**
	prefix '-': ***like*** Current	Negated version of current object, such as used for representing a difference, e.g. a weight loss.
	infix '+' (other**:** *like Current*)**:** *like Current*	Sum of this quantity and another quantity of the same concrete type. The value of *accuracy* in the result is either: • the sum of the accuracies of the operands, if both present, or; • *unknown_accuracy_value*, if either or both operand accuracies are *unknown_accuracy_value*. If the accuracy value is a percentage in one operand and not in the other, the form in the result is that of the larger operand.
	infix '-' (other**:** *like Current*)**:** *like Current*	Difference of this quantity and another quantity of the same concrete type. The value of *accuracy* in the result is either: • the sum of the accuracies of the operands, if both present, or; • unknown, if either or both operand accuracies are unknown. If the accuracy value is a percentage in one operand and not in the other, the form in the result is that of the larger operand.
Attributes	**Signature**	**Meaning**

CLASS	*DV_AMOUNT (abstract)*	
0..1	**accuracy**: `Real`	Accuracy of measurement, expressed either as a half-range percent value (accuracy_is_percent = True) or a half-range quantity. A value of 0 means that accuracy is 100%, i.e. no error. A value of *unknown_accuracy_value* means that accuracy was not recorded.
0..1	**accuracy_is_percent**: `Boolean`	If True, indicates that when this object was created, accuracy was recorded as a percent value; if False, as an absolute quantity value.
Functions	**Signature**	**Meaning**
	accuracy_unknown: `Boolean` ***ensure*** Result = (accuracy = unknown_accuracy_value)	True if accuracy is not known, e.g. due to not being recorded or discernable.
	valid_percentage (val: `Numeric`): `Boolean` ***ensure*** Result implies val >= 0.0 and val <= 100.0	Test whether a number is a valid percentage, i.e. between 0 and 100.
Invariants	***Accuracy_is_percent_validity***: accuracy = 0 implies **not** accuracy_is_percent ***Accuracy_validity***: accuracy_is_percent ***implies*** valid_percentage(accuracy)	

6.2.7 DV_QUANTITY Class

CLASS	DV_QUANTITY	
Purpose	Quantitified type representing “scientific” quantities, i.e. quantities expressed as a magnitude and units. Units were inspired by the Unified Code for Units of Measure (UCUM), developed by Gunther Schadow and Clement J. McDonald of The Regenstrief Institute [8].	
Use	Can also be used for time durations, where it is more convenient to treat these as simply a number of seconds rather than days, months, years.	
Inherit	`DV_AMOUNT`	
Attributes	**Signature**	**Meaning**
1..1	**magnitude**: `Double`	numeric magnitude of the quantity.
1..1	**units**: `String`	Stringified units, expressed in UCUM unit syntax, e.g. "kg/m2", “mm[Hg]", "ms-1", "km/h". Implemented accordingly in subtypes.
0..1	**precision**: `Integer`	Precision to which the value of the quantity is expressed, in terms of number of decimal places. The value 0 implies an integral quantity. The value -1 implies no limit, i.e. any number of decimal places.
Functions	**Signature**	**Meaning**
	is_integral: `Boolean`	True if precision = 0; quantity represents an integral number.
(effected)	**is_strictly_comparable_to (**other**: *like* Current**)**: `Boolean` ***require*** units_equivalent(units, other.units)	Test if two instances are strictly comparable by ensuring that the measured property is the same, achieved using the Measurement service function *units_equivalent*.
Invariants	***Precision_valid***: precision >= -1 ***Units_valid***: units /= void	

6.2.8 Units Syntax

The BNF syntax specification of the *units* string, adapted from [8] is as follows:

Parse Specification

```
units ::=     '/' exp_units
            | units '.' exp_units
            | units '/' exp_units
            | exp_units
```

```
exp_units ::= unit_group exponent | unit_group

unit_group ::= PREFIX annot_unit
             | annot_unit
             | '(' exp_units ')'
             | factor

annot_unit ::= unit_name
             | unit_name '{' ANNOTATION '}'
             | '{' ANNOTATION '}'

factor ::=   Integer

exponent ::= SIGN Integer | Integer
```

Lexical Specification

```
PREFIX ::=    'Y' |'Z' | 'E' | 'P' | 'T' | 'G' | 'M' | 'k' | 'h' | 'da'
             | 'd' | 'c' | 'm' | 'µ' | 'n' | 'p' | 'f' | 'a' | 'z' | 'y'
UNIT_NAME ::=      [a-zA-Z_%]+             ; from unit tables
ANNOTATION ::=     [a-zA-Z'.]+             ; from unit tables
SUFFIX ::=         [a-zA-Z0-9'_]+          ; from unit tables

SIGN ::=           '+' | '-'
Integer ::=        [0-9]+
```

This proposal is comprehensive, covering all useful unit systems, including SI, various imperial, customary mesaures, and some obscure measures, as well as clinically specific additions. Metric prefixes, meaning-changing textual suffixes (e.g. "`[Hg]`" in "`mm[Hg]`") and non-meaning-changing annotations (e.g. "`kg {total}`") are recognised. With this syntax, units can be simply expressed in strings such as:

```
"kg/m^2", "m.s^-1", "km/h", "mm[Hg]"
```

and so on.

6.2.9 DV_COUNT Class

CLASS	DV_COUNT	
Purpose	Countable quantities.	
Use	Used for countable types such as pregnancies and steps (taken by a physiotherapy patient), number of cigarettes smoked in a day.	
Misuse	Not used for amounts of physical entities (which all have units)	
Inherit	`DV_AMOUNT`	
Attributes	**Signature**	**Meaning**
1..1	**magnitude**: `Integer`	numeric magnitude of the quantity
Invariants		

6.2.10 DV_PROPORTION Class

CLASS	DV_PROPORTION	
Purpose	Models a ratio of values, i.e. where the numerator and denominator are both pure numbers. The *valid_proportion_kind* property of the `PROPORTION_KIND` class is used to control the *type* attribute to be one of a defined set.	
Use	Used for recording titers (e.g. 1:128), concentration ratios, e.g. Na:K (unitary denominator), albumin:creatinine ratio, and percentages, e.g. red cell distirbution width (RDW).	
MisUse	Should not be used to represent things like blood pressure which are often written using a '/' character, giving the misleading impression that the item is a ratio, when in fact it is a structured value. Similarly, visual acuity, often written as (e.g.) "6/24" in clinical notes is not a ratio but an ordinal (which includes non-numeric symbols like CF = count fingers etc). Should not be used for formulations.	
Inherit	`DV_AMOUNT, PROPORTION_KIND`	
Attributes	**Signature**	**Meaning**
1..1	**numerator**: `Real`	numerator of ratio
1..1	**denominator**: `Real`	denominator of ratio

CLASS	DV_PROPORTION	
1..1	**type**: `Integer`	Indicates semantic type of proportion, including percent, unitary etc. Values controlled by inherited properties from `PROPORTION_KIND`.
0..1	**precision**: `Integer`	Precision to which the numerator and denominator values of the proportion are expressed, in terms of number of decimal places. The value 0 implies an integral quantity. The value -1 implies no limit, i.e. any number of decimal places.
Functions	**Signature**	**Meaning**
	is_integral: `Boolean`	True if the numerator and denominator values are integers, i.e. if the precision is 0.
	magnitude: `Real` ***ensure*** Result = numerator / denominator	Effective magnitude represented by ratio.
(effected)	**is_strictly_comparable_to** (other**:** ***like*** Current): `Boolean` ***ensure*** type = other.type ***implies*** Result	True if *type* is the same.
Invariants	***Type_validity***: valid_proportion_kind(type) ***Precision_validity***: precision = 0 **implies** is_integral ***Is_integral_validity***: is_integral **implies** (numerator.floor = numerator **and** denominator.floor = denominator) ***Fraction_validity***: (type = pk_fraction **or** type = pk_integer_fraction) **implies** is_integral ***Unitary_validity***: type = pk_unitary **implies** denominator = 1 ***Percent_validity***: type = pk_percent **implies** denominator = 100	

6.2.11 PROPORTION_KIND Class

CLASS	PROPORTION_KIND	
Purpose	Class of enumeration constants defining types of proportion for the `DV_PROPORTION` class.	
Attributes	**Signature**	**Meaning**
const	**pk_ratio**: `Integer = 0`	Ratio type. Numerator and denominator may be any value
const	**pk_unitary**: `Integer = 1`	Denominator must be 1
const	**pk_percent**: `Integer = 2`	Denominator is 100, numerator is understood as a percentage value.
const	**pk_fraction**: `Integer = 3`	Numerator and denominator are integral, and the presentation method uses a slash, e.g. “1/2”.
const	**pk_integer_fraction**: `Integer = 4`	Numerator and denominator are integral, and the presentation method uses a slash, e.g. “1/2”; if the numerator is greater than the denominator, e.g. n=3, d=2, the presentation is “1 1/2”.
Functions	**Signature**	**Meaning**
	valid_proportion_kind (n: `Integer`): `Boolean`	True if n is one of the defined types.
Invariants		

6.2.12 DV_ABSOLUTE_QUANTITY Class

CLASS	*DV_ABSOLUTE_QUANTITY (abstract)*	
Purpose	Abstract class defining the concept of quantified entities whose values are absolute with respect to an origin. Dates and Times are the main example.	
Inherit	`DV_QUANTIFIED`	
Abstract	**Signature**	**Meaning**

CLASS	*DV_ABSOLUTE_QUANTITY (abstract)*	
	add **(a**_diff**:** *like diff***)**: *like Current*	Addition of a differential amount to this quantity. The value of *accuracy* in the result is either: • the sum of the accuracies of the operands, if both present, or; • unknown, if either or both operand accuracies are unknown.
	subtract **(**a_diff**:** *like diff***)**: *like Current*	Result of subtracting a differential amount from this quantity. The value of *accuracy* in the result is either: • the sum of the accuracies of the operands, if both present, or; • unknown, if either or both operand accuracies are unknown.
	diff **(**other**:** *like Current***)**: *DV_AMOUNT*	Difference of two quantities. The value of *accuracy* in the result is either: • the sum of the accuracies of the operands, if both present, or; • unknown, if either or both operand accuracies are unknown.
Attributes	**Signature**	**Meaning**
0..1	**accuracy**: *like* diff	Accuracy of measurement, expressed as a half-range value of the *diff* type for this quantity (i.e. an accuracy of x means +/–x). A Void (i.e. null) value means accuracy not known.
Functions	**Signature**	**Meaning**
	accuracy_unknown: Boolean ***ensure*** Result = (accuracy = Void)	True if accuracy is not known, e.g. due to not being recorded or discernable.
Invariants		

7 Date Time Package

7.1 Overview

The data_types.quantity.date_time package includes three absolute date/time concepts: DV_DATE, DV_TIME, DV_DATE_TIME, and a relative concept: DV_DURATION. The representations of all of these are ISO8601:2004-compatible date/time strings. They also include the ISO8601 semantics for partial dates and times. The date_time package is illustrated in FIGURE 9.

7.1.1 Requirements

Standard Date/Times

The basic requirement is for types which represent the following concepts:

- *Date*: a type which records year, month and day in month. Examples include date of birth, date of onset of a problem
- *Time*: a type which records hour, minute, second, and timezone. Examples include time of meal, time of day when a problem recurs. Timezone is required in a shared EHR repository so that times of clinical events which occurred in different timezones are comparable; this includes specialised pathology tests which might be done in another country.
- *Date_time*: a type which records year, month, day, hour, minute, second, and timezone. Examples include date & time of death, timestamp of any observation. Timezone required for the same reason as in Time.
- *Duration*: a type which records duration of an event or (in)activity, as days, hours, minutes, and seconds.

Partial Date/Times

Partial or uncertain date/times have to be supported in clinical medicine. It is common for patients to be unsure about dates and durations. Requirements for partial date/times include the following.

- For dates, one of the following rules applies to any instance:
  - only the year is known
  - only the year and month are known

  If not even the year is known, then the date is obviously extremely approximate and it would probably be unsafe to represent it computationally. However, if computatable representation was needed in this case, a date interval can be used. A pedantic example which breaks these rules is someone who claims to be born on “a Monday at the start of May in 1934” (i.e. day but not date unknown). Either the clinician determines what date the first Monday in May 1934 actually was and record that (assuming the patient’s way of accurately remembering just happens to be via day rather than date), or else records a partial date of the form “May 1934” (in ISO 8601 form, “1934-05”) if they determine that the patient really is unsure.
- Sometimes incomplete times are recorded, which follow the same rule that either the hours or both the hours and minutes are present. Examples:
  - recordings by instruments which only generate hh:mm values (i.e. no seconds);
  - recordings by patients who report approximate times of events;
  - recordings by clinicians who use approximate times in administrations, e.g. “take insulin at 8am” really means something like 8am +/- 30 mins.
- Imprecise durations such as “2 - 3 hrs” need to be recordable in a computable form.

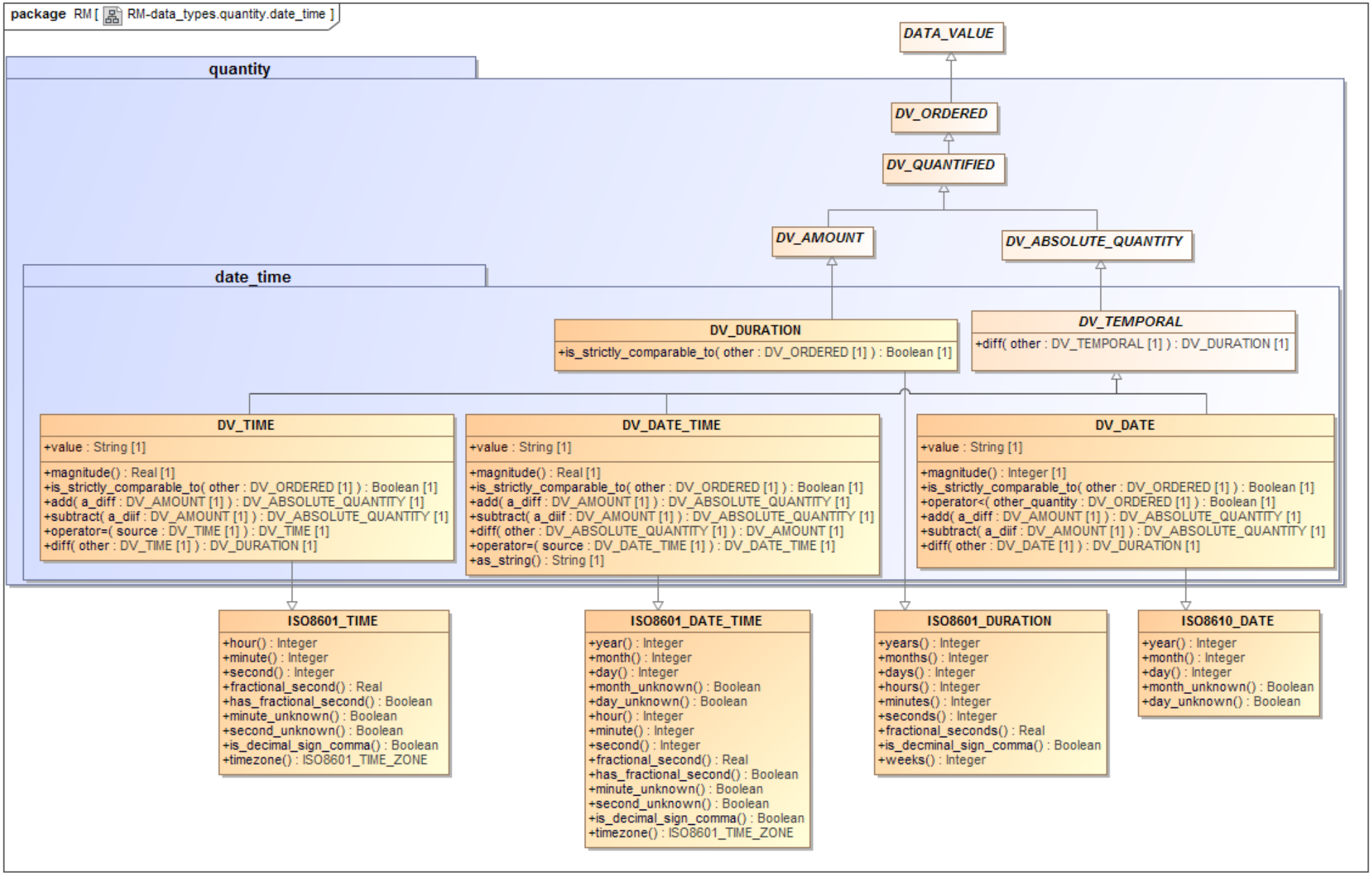


FIGURE 9 rm.data_types.quantity.date_time Package

To satisfy the faithfulness requirement for health record recording it should always be possible to record the narrative form of the datum provided by the patient as well as the formal form.

7.1.2 Design

General Approach

Date/time values are somewhat special in the realm of data types in that they are expressed in their "customary" form, in which the standard structure of {value, unit} and metric relationships between orders of magnitude do not hold. The customary form is what we are used to using in the social time domain, such as births, deaths, ages, and times and durations of events which we remember. In all these cases it is expressed using the familiar year/month/date/hour/minute/second system, in which the relationships between each successive unit of time is non-metric. The customary form can be converted to a magnitude since an origin point, and many date/time libraries do this in order to implement various operations, particularly comparison.

The date/time types fall into two categories: *absolute* and *relative*. The absolute category comprises the Date, Date_time and Time concepts, each of which measure time from a known origin. Date and Date_time measure calendrical time from the date 0001-01-01, while Time measures clock time from midnight. Both Date_time and Time can include timezone information, ensuring that their instances are correctly situated on the same timeline. All absolute time types inherit from the `DV_TEMPORAL` type, which provides the appropriate signature for the *diff()* function.

The relative category contains only the Duration concept, which expresses elapsed time between two time points. The `DV_DURATION` class is used for expressing durations of clinical phenomena and differences between absolute times.

All four concepts are defined in the ISO 8601:2004 standard, which is accordingly used as the definitional basis of the *open*EHR date/time types.

Partial Date/Times

The types defined in this specification support the notion of partially specified dates and times. The modelling approach used here takes into account the known needs for representing partial date/time data, while balancing that with the need to avoid incomprehensibly complex types whose generality would only apply to a tiny percent of difficult cases. Thus, the basis for modelling incomplete date/times is as follows.

- The modelling problem relates only to date/time quantities that need to be computable. For extremely imprecise date/times, if the clinician feels the need, she can record it as narrative text.
- For imprecise durations, an interval should be used, i.e. `DV_INTERVAL<DV_DURATION>`. In this way durations like "2 - 3 hrs" can be represented, and still be computable.

Based on the above considerations, the requirements for partial types are satisfied by the semantics of ISO8601:2004 for "reduced accuracy" date/times, in which parts of a date, time or date/time can be missing from the right hand end of the string. This models the reasonable situation where e.g. day may be unknown in a date, but a date cannot have month unknown and day known.

Calendars

A comment on calendars is in order. In this specification, all date/time types currently modelled are Gregorian calendar based. This is the same assumption made by by ISO 8601, and most technical computing systems today in many parts of the world. At first glance this may seem like a culturally insensitive approach, but in fact it makes sense in computational terms, for both users of the Grego-

rian and other calendars, e.g. Julian, Islamic, Baha'i, etc. Arguments against trying to use the date/time classes defined here to represent date/times from any calendar include the following:

- Almost all dates on computer systems, including in regions such as the Indian sub-continent, Turkey and the middle east, where alternate calendars are in use, are in the Gregorian system. This is likely to be the case for some time, and may always be the case, regardless of the continued use of other calendars for religious or other purposes (outside of health);
- If a calendar indicator were used in date quantities, all software, to be correct, would have to check the value to verify that it is in the expected calendar system, and to do something special if it is not - an added cost which is a possible source of bugs and which would rarely be used. The reality is that most software produced in the western world, India etc (possibly excepting open source software) would automatically assume the Gregorian calendar, and would be in error if ever it did receive EHR data containing dates from alternate calendars.
- If/when other calendars are used in EHR or related systems, the users of those calendars will be aware of it, and include the appropriate conversion logic between Gregorian dates and their own, limiting the extra software work and quality issues to those users who actually need alternate calendars. If EHRs from such places are sent to a health care facility where Gregorian is the default, nothing special is needed to ensure that those records will contain dates comprehensible to the receiver.
- The detailed model of date/times in some other calendars is not the same as in the Gregorian calendar, so they would require different classes anyway - the classes defined here would not necessarily function correctly simply by adding a calendar field.

For users requiring non-Gregorian dates in EHR and other health systems there are two approaches. One is to treat non-Gregorian dates as a localisation issue, to be handled inside the application and GUI evnvironment. The other is to actually add further sibling packages to the *open*EHR date/time package, for each new calendar or calendar group required. Conversion algorithms would most likely be needed in and out of the Gregorian types to enable interoperability of information drawn from different applications or sources. This approach may require a substantial modelling effort.

Algorithms for conversion between the Egyptian, Armenian, Khwarizmian, Persian, Ethiopian, Coptic, Republican, Macedonian, Syrian, Julian Roman, Gregorian, Islamic A, Islamic B, Baha'i and Saka calendars are described by Richards [7] and are based on the work of D. A. Hatcher (1986).

Representation

All of the date/time classes described here are defined so as to have an attribute called *value* of type String, in the form of an ISO 8601:2004 string. ISO 8601 is convenient for this purpose, as it is a simple syntax, and covers not only all four variants of fully-specified date/time described here, but also the partial varieties. Using a single string attribute significantly simplifies persistence as well as mapping to XML-based formalisms, which use a mostly ISO 8601 compliant date/time representation. The ISO 8601 semantics assumed by EHR are defined in classes found in the classes `ISO8601_DATE`, `ISO8601_TIME`, `ISO8601_DATE_TIME`, `ISO8601_DURATION`, from the `rm.support.assumed_types` package. These classes are inherited into the corresponding classes defined below.

7.2 Class Descriptions

7.2.1 DV_TEMPORAL Class

CLASS	*DV_TEMPORAL (abstract)*	
Purpose	Specialised temporal variant of `DV_ABSOLUTE_QUANTITY` whose *diff* type is `DV_DURATION`.	
Inherit	`DV_ABSOLUTE_QUANTITY`	
Abstract	**Signature**	**Meaning**
(redefined)	***diff*** **(**other**:** *`like Current`***)**: `DV_DURATION`	Difference of two temporal quantities.
Invariants		

7.2.2 DV_DATE Class

CLASS	DV_DATE	
Purpose	Represents an absolute point in time, as measured on the Gregorian calendar, and specified only to the day. Semantics defined by ISO 8601.	
Use	Used for recording dates in real world time. The partial form is used for approximate birth dates, dates of death, etc.	
Inherit	`DV_TEMPORAL, ISO8601_DATE`	
Functions	**Signature**	**Meaning**
(effected)	**diff (**other**:** *`like Current`***)**: `DV_DURATION`	Difference of two dates.
Attributes	**Signature**	**Meaning**
	value: `String`	ISO8601 date string
Functions	**Signature**	**Meaning**
(effected)	**magnitude**: `Integer` ***ensure*** Result >= 0	Numeric value of the date as days since the calendar origin point 1/1/0000
Invariants	***Value_valid***: valid_iso8601_date(value)	

7.2.3 DV_TIME Class

CLASS	DV_TIME	
Purpose	Represents an absolute point in time from an origin usually interpreted as meaning the start of the current day, specified to the second. Semantics defined by ISO 8601.	
Use	Used for recording real world times, rather than scientifically measured fine amounts of time. The partial form is used for approximate times of events and substance administrations.	
Inherit	`DV_TEMPORAL, ISO8601_TIME`	
Functions	**Signature**	**Meaning**
(effected)	**diff (**other**:** *`like Current`***)**: `DV_DURATION`	Difference of two times.
Attributes	**Signature**	**Meaning**
	value: `String`	ISO8601 time string
Functions	**Signature**	**Meaning**
(effected)	**magnitude**: `Double` ***ensure*** Result >= 0.0	Numeric value of the time as seconds since the start of day.
Invariants	***Value_valid***: valid_iso8601_time(value)	

7.2.4 DV_DATE_TIME Class

CLASS	DV_DATE_TIME	
Purpose	Represents an absolute point in time, specified to the second. Semantics defined by ISO 8601.	
Use	Used for recording a precise point in real world time, and for approximate time stamps, e.g. the origin of a `HISTORY` in an `OBSERVATION` which is only partially known.	
Inherit	`DV_TEMPORAL, ISO8601_DATE_TIME`	
Functions	**Signature**	**Meaning**
(effected)	**diff (**other**:** *`like Current`***)**: `DV_DURATION`	Difference of two date/times.
Attributes	**Signature**	**Meaning**
	value: `String`	ISO8601 date/time string

CLASS	DV_DATE_TIME	
Functions	**Signature**	**Meaning**
(effected)	**magnitude**: `Double` ***ensure*** Result >= 0.0	numeric value of the date/time as seconds since the calendar origin point.
Invariants	***Value_valid***: valid_iso8601_date_time(value)	

7.2.5 DV_DURATION Class

CLASS	DV_DURATION	
Purpose	Represents a period of time with respect to a notional point in time, which is not specified. A sign may be used to indicate the duration is "backwards" in time rather than forwards. Note that a deviation from ISO8601 is supported, allowing the 'W' designator to be mixed with other designators. See assumed types section in the Support IM.	
Use	Used for recording the duration of something in the real world, particularly when there is a need a) to represent the duration in customary format, i.e. days, hours, minutes etc, and b) if it will be used in computational operations with date/time quantities, i.e. additions, subtractions etc.	
MisUse	Durations cannot be used to represent points in time, or intervals of time.	
Inherit	`DV_AMOUNT, ISO8601_DURATION`	
Attributes	**Signature**	**Meaning**
	value: `String`	ISO8601 duration
Functions	**Signature**	**Meaning**
	prefix '-': ***`like`*** `Current`	Negated copy of current object.
(effected)	**magnitude**: `Double`	Numeric value of the duration in seconds.
Invariants	***Value_valid***: valid_iso8601_duration(value)	

8 Time_specification Package

8.1 Overview

Time specification is about potentiality rather than actuality, and needs its own types. The *open*EHR `data_types.time_specification` package provides such types, based on the HL7 types of the same names, and is illustrated in FIGURE 10.

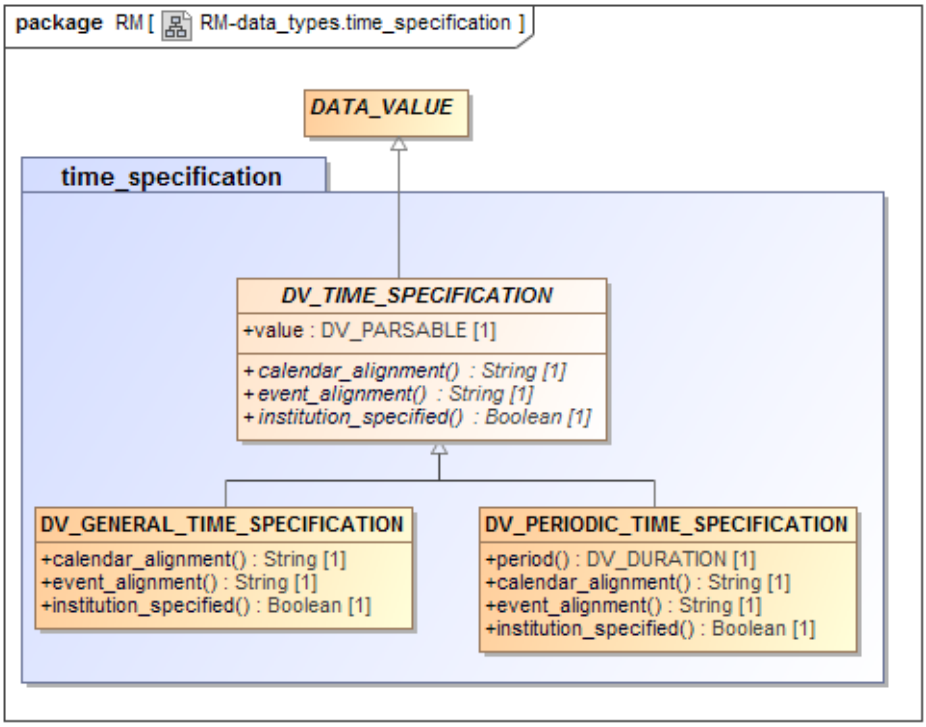


FIGURE 10 rm.data_types.time_specification Package

8.1.1 Requirements

One of the difficulties with time is expressing future times, since potential occurrences, durations, repetitions cannot be expressed in the same way as actual time. Complicating the problem is the fact that humans tend to use very customary (i.e. calandar-anchored) ways of specifying time, such as “every second Tuesday”, or “ the first Sunday of the month”. In clinical medicine, future time is most commonly used to express when medications or other therapies are intended to take place. They are often anchored to the calendar, and can easily include repetitions.

As with other time types, there are both simple and complex cases to consider. One of the most common examples of time in the future is the timing for drug administrations, e.g. “once every four hours”. This could be represented as a simple periodic specification, consisting of a start point in time, a period, and a number of repetitions. The specification for taking blood sugar levels during a glucose test could be represented as a simple aperiodic series, e.g. “.5hr, 1hr, 2hr”. However, even common specifications for prescriptions e.g. “three times a day for seven days” start to become quite complex, for example, because “three times a day” might not mean literally 8 hours apart.

Some of the factors to consider in timing specifications are:

- period of repetition
- duration of activity being specified

- possible alignment to the calendar, e.g. “every 5th of the month”
- possible alignment to real world events e.g. “after meals”
- fuzziness

Because time is inherently “messy” (months do not all have the same number of days, leap years change the number of days in some years etc), and because the relationship we have with time can also be arbitrary (e.g. anchored to mealtimes etc), specifying linguistically obvious specifications formally is quite challenging.

8.1.2 Design

The HL7 version 3 data types for time specification appear to allow for all of the required possibilities. The syntax is based on the ISO 8601 standard [9]. It provides types which express:

- Periodic intervals (HL7v3 - `PIVL<T:TS>`) - allows period, duration, and calendar linking to be specified.
- Event-linked periodic intervals (HL7v3 - `EIVL<T:TS>`) - allows `PIVLs` to be linked to real-world events like meals.
- General timing specification (HL7v3 - `GTS`) - allows any time specification to be expressed, using a syntax which is equivalet to a series of `IVL<TS>` (i.e. intervals of `DATE_TIME`).

The HL7 syntax for time specification is encapsulated in equivalent *open*EHR types described here.

8.2 Class Descriptions

8.2.1 DV_TIME_SPECIFICATION Class

CLASS	*DV_TIME_SPECIFICATION (abstract)*	
Purpose	This is an abstract class of which all timing specifications are specialisations. Specifies points in time, possibly linked to the calendar, or a real world repeating event, such as “breakfast”.	
Inherit	DATA_VALUE	
Attributes	**Signature**	**Meaning**
1..1	**value**: DV_PARSABLE	the specification, in the HL7v3 syntax for PIVL or EIVL types. See below.
Abstract	**Signature**	**Meaning**
	calendar_alignment: String	Indicates what prototypical point in the calendar the specification is aligned to, e.g. “5th of the month”. Empty if not aligned. Extracted from the ‘value’ attribute.
	event_alignment: String	Indicates what real-world event the specification is aligned to if any. Extracted from the ‘value’ attribute.
	institution_specified: Boolean	Indicates if the specification is aligned with institution schedules, e.g. a hospital nursing changeover or meal serving times. Extracted from the ‘value’ attribute.
Invariant	***Value_valid***: value /= Void	

8.2.2 DV_PERIODIC_TIME_SPECIFICATION Class

CLASS	DV_PERIODIC_TIME_SPECIFICATION	
Purpose	Specifies periodic points in time, linked to the calendar (phase-linked), or a real world repeating event, such as “breakfast” (event-linked). Based on the HL7v3 data types PIVL<T> and EIVL<T>.	
Use	Used in therapeutic prescriptions, expressed as INSTRUCTIONs in the *open*EHR model.	
Inherit	DV_TIME_SPECIFICATION	
Functions	**Signature**	**Meaning**

CLASS	DV_PERIODIC_TIME_SPECIFICATION	
	period: DV_DURATION ***ensure*** Result /= Void	The period of the repetition, computationally derived from the syntax representation. Extracted from the 'value' attribute.
	calendar_alignment: String	Calendar alignment extracted from value.
	event_alignment: String	Event alignment extracted from value.
	institution_specified: Boolean	Extracted from value.
Invariant	***Value_valid***: value.formalism.is_equal("HL7:PIVL") ***or*** value.formalism.is_equal("HL7:EIVL")	

8.2.2.1 Phase-linked Time Specification Syntax

The syntactic form of phase-linked periodic time specifications (derived from the PIVL<T> spec HL7v3 ballot) is as follows.

```
"[" interval "]" "/" "(" difference ")" [ "@" alignment ] [ "IST" ]
```

Examples include:

- [200004181100;200004181110]/(7d)@DW = every Tuesday from 11:00 to 11:10 AM.
- [200004181100;200004181110]/(1mo)@DM" = every 18th of the month 11:00 to 11:10 AM.

A parse specification is as follows:

```
phase_linked_time_spec: pure_phase_linked_time_spec |
                  pure_phase_linked_time_spec "IST"

pure_phase_linked_time_spec: phase |
                  phase "@" alignment

phase: interval "/" "(" difference ")"

alignment: "DW" | etc /* terms from "HL7::CalendarCycle" domain */

difference: /* ISO 8601 for time difference */

interval: "[" interval_spec "]"

interval_spec: ";" |
                  ";" date_time |
                  date_time ";" date_time |
                  date_time ";"

date_time: /* ISO 8601 for date/time string yyyymmdd[hh[mm[ss]]] */
```

8.2.2.2 Event-linked Periodic Time Specification Syntax

Examples of event-linked periodic time specifications include:

- "PC+[1h;1h]" = one hour after meal

- "HS-[50min;1h]" = one hour before bedtime for 10 minutes

The following parse specification defines the syntax for event-related periodic time specifications.

```
event_linked_time_spec: event |
            event offset

event: "AC" | "ACD" | etc /* HL7 domain "HL7::TimingEvent" */

offset: "+" dur_interval |
       "-" dur_interval

dur_interval: /* ISO 8601 for duration interval */
```

8.2.3 DV_GENERAL_TIME_SPECIFICATION Class

CLASS	DV_GENERAL_TIME_SPECIFICATION	
Purpose	Specifies points in time in a general syntax. Based on the HL7v3 GTS data type.	
Use		
Inherit	DV_TIME_SPECIFICATION	
Functions	**Signature**	**Meaning**
	calendar_alignment: String	Calendar alignment extracted from value.
	event_alignment: String	Event alignment extracted from value.
	institution_specified: Boolean	Extracted from value.
Invariant	***Value_valid***: value.formalism.is_equal("HL7:GTS")	

8.2.3.1 General Time Specification Syntax

The class is the same structurally as the DV_TIME_SPECIFICATION parent. The syntax is the HL7 GTS syntax, defined by the following parse specification:

```
general_time_spec: symbol |
       union |
       exclusion

union: intersection ";" union |
       intersection

exclusion: exclusion "\" intersection

intersection: factor intersection |
       factor

hull:  factor ".." hull |
       factor

factor:
```

```
            interval |
            phase_linked_time_spec |
            event_linked_time_spec |
            "(" general_time_spec ")"
```

9 Encapsulated Package

9.1 Overview

The `data_types.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 11.

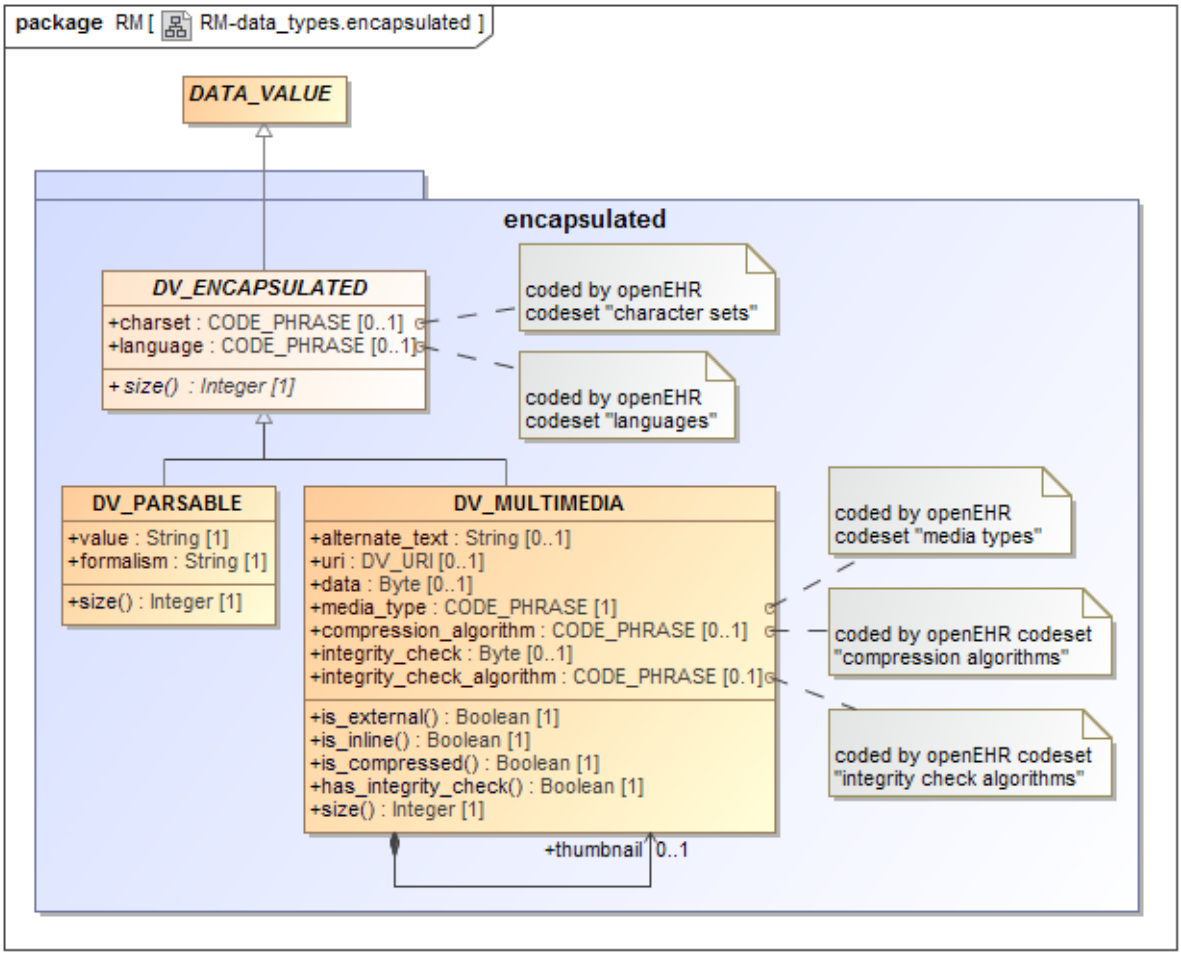


FIGURE 11 rm.data_types.encapsulated Package

9.1.1 Requirements

There is a need to be able to include content in the EHR whose interior structure is not modelled in the EHR reference model, but instead documented by sufficient meta-data attributes for specific tools to process the data. Types of content in this category are as follows.

- Images, including images which are themselves a compressed version of one image from a high-resolution image set stored elsewhere. Such images may be in any of the well-known compressed or uncompressed formats, and may have their own thumbnail image attached, to facilitate web-viewing.
- Bio-signal data series, such as a set of values representing a diagnostic part of an ECG trace. This might be represented as DICOM content.
- Content which is textual (or nearly so) which is essentially a parsable language file of some kind. This includes all XML instance, HTML, and any other EHR content which happens to

be represented in syntax form - such as the unit strings used in quantities. The name of the formalism should be stored as meta-data.

- Binary content which is processed by a work processor or other dedicated tool.
- Digital signatures.

Sufficient meta-data must be included with all of these types to enable a way for the content to be processed, typically by indicating either its type (e.g. “jpeg”, “word document”) or the name of a tool which can be used to process it. Important meta-data include:

- size of the content;
- natural language, if any.

Any encapsulated data item may be a summary, “thumbnail” or otherwise reduced form of an original content item found outside the EHR, in some other system or file-system.

Checksums must be expressible for those items for which a checksum is available, or for which the system generates checksums to improve the quality of its internal data transmissions.

9.1.2 Design

The design approach used here is based on the following analysis.

1. Any encapsulated data item may be in some particular language, even if it is an image or other graphic form such as a biosignal with axis markings in a particular language;
2. The general structure of encapsulated content data items includes a block of bytes or characters representing the content, and various meta-data as appropriate, including:

- size
- character encoding
- compression type/algorithm
- name of formalism for parsable content

3. For encapsulated items that have a counterpart in another system, the standard means of portable address is the W3C URI;
4. For items may that have an associated integrity checksum, the checksum is itself a series of bytes, and the type of checksum must also be specified, e.g. “md5”.

These observations lead naturally to an abstract `DV_ENCAPSULATED` class, with two subtypes, `DV_PARSABLE`, for all content which is syntactic in nature, and `DV_MULTIMEDIA` for everything else. Note that it is possible to imagine parsable content items which are large, stored in compressed form, and are themselves a summary of another item elsewhere on the web; such items can for practical purposes be represented as instances of `DV_MULTIMEDIA`, rather than `DV_PARSABLE`. The vast majority of parsable encapsulated data are expected to be short and stored in native textual form, e.g. fragments of XML or HTML.

The formal model of the classes `DV_ENCAPSULATED` and `DV_MULTIMEDIA` are closely based on the ED type from the HL7v3 data types specification.

9.2 Class Descriptions

9.2.1 DV_ENCAPSULATED Class

CLASS	*DV_ENCAPSULATED (abstract)*	
Purpose	Abstract class defining the common meta-data of all types of encapsulated data.	
Inherit	DATA_VALUE	
Abstract	**Signature**	**Meaning**
1..1	***size***: Integer	Original size in bytes of unencoded encapsulated data. I.e. encodings such as base64, hexadecimal etc do not change the value of this attribute.
Attributes	**Signature**	**Meaning**
0..1	**charset**: CODE_PHRASE	Name of character encoding scheme in which this value is encoded. Coded from *open*EHR Code Set "character sets". Unicode is the default assumption in *open*EHR, with UTF-8 being the assumed encoding. This attribute allows for variations from these assumptions.
0..1	**language**: CODE_PHRASE	Optional indicator of the localised language in which the data is written, if relevant. Coded from *open*EHR Code Set "languages".
Functions	**Signature**	**Meaning**
	as_string: String	Result = alternate_text [(uri)]
Invariant	***Size_positive***: size >= 0 ***Language_valid***: language /= Void **implies** code_set(Code_set_id_languages).has_code(language) ***Charset_valid***: charset /= Void **implies** code_set(Code_set_id_character_sets).has_code(charset)	

9.2.2 DV_MULTIMEDIA Class

CLASS	DV_MULTIMEDIA
Purpose	A specialisation of DV_ENCAPSULATED for audiovisual and biosignal types. Includes further metadata relating to multimedia types which are not applicable to other subtypes of DV_ENCAPSULATED.
Use	

CLASS	DV_MULTIMEDIA	
Inherit	DV_ENCAPSULATED	
Attributes	**Signature**	**Meaning**
0..1	**alternate_text**: String	Text to display in lieu of multimedia display/replay
1..1	**media_type**: CODE_PHRASE	Data media type coded from *open*EHR code set "media types" (interface for the IANA MIME types code set).
0..1	**compression_algorithm**: CODE_PHRASE	Compression type, a coded value from the *open*EHR "Integrity check" code set. Void means no compression.
0..1	**integrity_check**: Array <Octet>	Binary cryptographic integrity checksum
0..1 (cond)	**integrity_check_algorithm**: CODE_PHRASE	Type of integrity check, a coded value from the *open*EHR "Integrity check" code set.
0..1	**thumbnail**: DV_MULTIMEDIA	The thumbnail for this item, if one exists; mainly for graphics formats.
0..1 (cond)	**uri**: DV_URI	URI reference to electronic information stored outside the record as a file, database entry etc, if supplied as a reference.
0..1 (cond)	**data**: Array <Octet>	The actual data found at *uri*, if supplied inline
1..1 (effected)	**size**: Integer	Size in bytes of *data*, if present, or else of the object referred to by *uri*.
Functions	**Signature**	**Meaning**
	is_external: Boolean ***ensure*** uri /= Void ***implies*** Result	Computed from the value of the *uri* attribute: True if the data is stored externally to the record, as indicated by `uri'. A copy may also be stored internally, in which case `is_expanded' is also true.
	is_inline: Boolean ***ensure*** data /= Void ***implies*** Result	Computed from the value of the *data* attribute: True if the data is stored in expanded form, ie within the EHR itself.
	is_compressed: Boolean ***ensure*** compression_algorithm /= Void ***implies*** Result	Computed from the value of the *compression_algorithm* attribute: True if the data is stored in compressed form.

CLASS	DV_MULTIMEDIA	
	has_integrity_check: `Boolean` ***ensure*** integrity_check_algorithm /= Void ***implies*** Result	Computed from the value of the *integrity_check_algorithm* attribute: True if an integrity check has been computed.
Invariant	***Not_empty***: is_inline ***or*** is_external ***Media_type_validity***: media_type /= Void ***and then*** code_set(Code_set_id_media_types).has_code(media_type) ***Compression_algorithm_validity***: compression_algorithm /= Void ***implies*** code_set(Code_set_id_compression_algorithms).has_code(compression_algorithm) ***Integrity_check_validity***: integrity_check /= Void ***implies*** integrity_check_algorithm /= Void ***Integrity_check_algorithm_validity***: integrity_check_algorithm /= Void ***implies*** code_set(Code_set_id_integrity_check_algorithms).has_code(integrity_check_algorithm)	

9.2.3 DV_PARSABLE Class

CLASS	DV_PARSABLE	
Purpose	Encapsulated data expressed as a parsable `String`. The internal model of the data item is not described in the *open*EHR model in common with other encapsulated types, but in this case, the form of the data is assumed to be plaintext, rather than compressed or other types of large binary data.	
Use	Used for representing values which are formal textual representations, e.g. guidelines.	
Inherit	`DV_ENCAPSULATED`	
Functions	**Signature**	**Meaning**
1..1 (effected)	**size**: `Integer`	Size in bytes of *value*.
Attributes	**Signature**	**Meaning**
1..1	**value**: `String`	The string, which may validly be empty in some syntaxes
1..1	**formalism**: `String`	name of the formalism, e.g. “GLIF 1.0”, “proforma” etc.
Invariant	***value_valid***: value /= Void ***formalism_validity***: formalism /= Void ***and then not*** formalism.is_empty	

10 Uri Package

10.1 Overview

The `data_types.uri` package includes two types used for referring to information resources. The `DV_URI` type allows data values which are references to objects on the world wide web to be created. Its specialisation, `DV_EHR_URI`, enables any element in an *open*EHR record to be identified in the same way as other objects on the web. The `DV_EHR_URI` type is convenient, because it is a string, like any other URI, and is therefore easily transportable and processable. Because it has its own scheme space, "ehr", instances can be globally unique, as long as EHR identification is globally unique. `DV_EHR_URIs` are used to express all runtime paths in the EHR. The `uri` Package is illustrated in FIGURE 12.

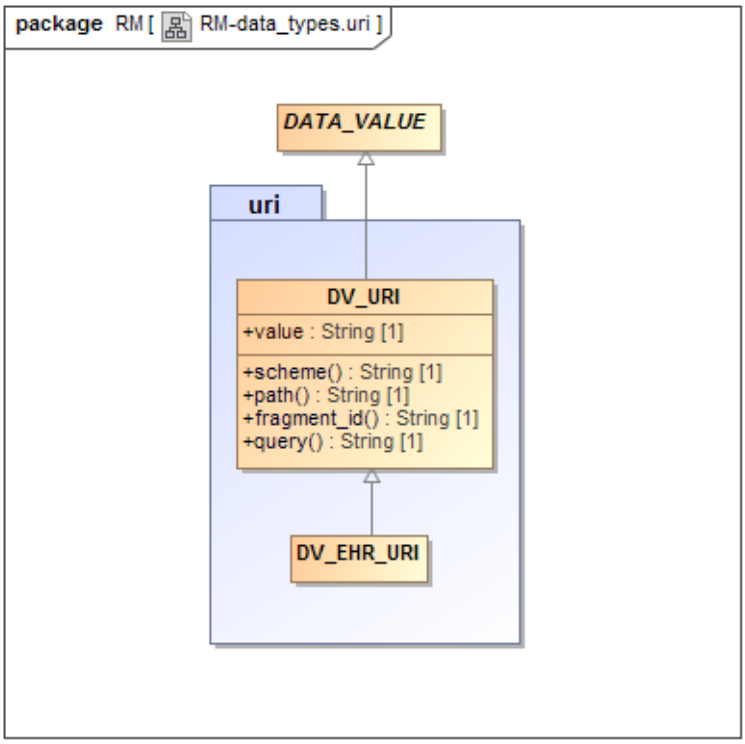


FIGURE 12 rm.data_types.uri Package

10.1.1 Requirements

This package meets the requirement for a `DATA_VALUE` subtype which represents a W3C Uniform Resource Identifier (URI). A common example of where this might be used is to represent a reference to a clinical guideline or other justifying document associated with an intervention or treatment plan recorded in the EHR.

URIs are a superset of Uniform Resource Locators (URLs) (although the two are often confused, even within the W3C), and can be used to specify the location of any information item, regardless of its type, location or storage method, as long as a URI "scheme" exists for that type of information.

There is an additional requirement for a kind of URI that can point at an EHR data item, either inside the same EHR containing the link, or in another EHR. This is the basis of implementing the `LINK` type.

10.1.2 Design

A simple design approach is used whereby a URI is represented as a String, and appropriate functions are defined to extract the various parts according to the syntax of URIs defined by Tim Berners-Lee at http://www.ietf.org/rfc/rfc2396.txt. An EHR specific subtype is defined, whose scheme is "ehr", and which contains further attributes enabling the instances of the type to record what kind of object they are referring to.

10.2 Definitions

The following symbolic definitions are used in the classes below.

- **Ehr_scheme**: String is "ehr"

10.3 Class Descriptions

10.3.1 DV_URI Class

CLASS	DV_URI	
Purpose	A reference to an object which conforms to the Universal Resource Identifier (URI) standard, as defined by W3C RFC 2936. See "Universal Resource Identifiers in WWW" by Tim Berners-Lee at http://www.ietf.org/rfc/rfc2396.txt. This is a World-Wide Web RFC for global identification of resources. See http://www.w3.org/Addressing for a starting point on URIs. See http://www.ietf.org/rfc/rfc2806.txt for new URI types like telephone, fax and modem numbers.	
Use	Enables external resources to be referenced from within the content of the EHR. A number of functions return the logical subparts of the URI string.	
Inherit	DATA_VALUE	
Attributes	**Signature**	**Meaning**
1..1	**value**: String	Value of URI as a String.
Functions	**Signature**	**Meaning**
	scheme: String	A distributed information "space" in which information objects exist. The scheme simultaneously specifies an information space and a mechanism for accessing objects in that space. For example if scheme = "ftp", it identifies the information space in which all ftpable objects exist, and also the application - ftp - which can be used to access them. Values may include: "ftp", "telnet", "mailto", "gopher" and many others. Refer to WWW URI RFC for a full list. New information spaces can be accommodated within the URI specification.
	path: String	A string whose format is a function of the scheme. Identifies the location in <scheme>-space of an information entity. Typical values include hierarchical directory paths for any machine. For example, with scheme = "ftp", path might be /pub/images/image_01. The strings "." and ".." are reserved for use in the path. Paths may include internet/intranet location identifiers of the form: sub_domain...domain, e.g. "info.cern.ch"

CLASS	DV_URI	
	fragment_id: `String`	A part of, a fragment or a sub-function within an object. Allows references to sub-parts of objects, such as a certain line and character position in a text object. The syntax and semantics are defined by the application responsible for the object.
	query: `String`	Query string to send to application implied by scheme and path Enables queries to applications, including databases to be included in the URI Any query meaningful to the server, including SQL.
Invariant	***value_exists***: value /= Void ***and then not*** value.is_empty	

10.3.2 DV_EHR_URI Class

CLASS	DV_EHR_URI	
Purpose	A `DV_EHR_URI` is a `DV_URI` which has the scheme name "ehr", and which can only reference elements in EHRs. The syntax is described below.	
Use	Used to reference elements in an EHR, which may be the current one, or another.	
Inherit	`DV_URI`	
Functions	**Signature**	**Meaning**
Invariant	***Scheme_is_ehr***: scheme.is_equal(Ehr_scheme)	

10.3.2.1 DV_EHR_URI Syntax

The syntax of a `DV_EHR_URI` is an *open*EHR path, inside the "ehr" URI scheme-space, and is of the form:

"ehr://" ehr_path

The syntax of ehr_path is described in the section on Paths in The *open*EHR Architecture Overview document. `DV_EHR_URIs` are used as a mechanism for referencing in the EHR, ensuring readability by humans, as well as validity when extracts are transmitted elsewhere: even if the target of a path is not present, the path can be used to locate the missing item on demand.

11 Implementation Strategies

11.1 Overview

This section notes a few of the general challenges for mapping the *open*EHR data types to implementation technologies such as programming languages and XML. For specific guidelines, Implementation Technology Specification (ITS) document for each target formalism should be consulted.

11.2 Quantities and Ordered_numeric

In the `quantity` package, the type `DV_QUANTIFIED` is shown having an abstract property of type `Ordered_numeric`. This is intended to indicate that the type `DV_QUANTIFIED` is distinguished by the *magnitude* property (compared to say `DV_ORDERED`, which describes ordered things without having magnitudes). The type `Ordered_numeric` be mapped to various types in implementation technologies as follows:

- Java: `java.lang.Number`
- C#: `System.IComparable`
- Eiffel: `NUMERIC`

All of these type systems currently suffer from not having a single type whose meaning is both "ordered" (having the function '<') and "numeric" (having the functions '+', '-', '*', '/') but in practice it does not matter much. For type systems with no convenient supertype of the numeric concrete types `Real`, `Integer`, `Double`, the magnitude property can safely be left out of `DV_QUANTIFIED`; the only drawback is that code cannot call `DV_QUANTIFIED`.*magnitude* polymmorphically, e.g. in a statistical application processing `DV_QUANTITY` and `DV_COUNT` objects.

11.3 Unicode

Unicode is supported in various ways in different languages. In Java, since JDK 1.1, unicode support is implicit in the base classes. From the documentation:

> the classes `java.io.InputStreamReader`, `java.io.OutputStreamWriter`, and `java.lang.String` can convert between Unicode and a number of other character encodings. More information is available at: http://java.sun.com/j2se/1.3/docs/guide/intl/encoding.doc.html.

In the C# language, conversion can be done between Unicode and other codepages using the `System.Text.UnicodeEncoding` (for UTF-16) and `System.Text.UTF8Encoding` (for UTF-8) classes.

In XML unicode is handled by specifying the encoding of the document in the XML declaration, e.g. `<?xml version="1.0" encoding="UTF-16"?>`.

In the Eiffel language, unicode is available in the Gobo public domain library (see http://www.gobosoft.com), in the `UC_STRING` class, which inherits from the `String` class.

The support in other languages varies, and may require a special type like the `UC_STRING` used in Eiffel.

11.4 Dates and Times

In some formalisms, dates and times are represented using a single calendar-like class. This is not considered to be good practice from the point of specification, since it is more difficult to state proper invariants for such a class used to represent a particular logical type such as a `DATE` or `TIME`, however, its utility in implementation is recognised.

Where implementors want to use such a class (call it `CALENDAR` here for the sake of discussion) the recommended approach is to wrap the class `CALENDAR` with classes representing the types described in this specification, i.e. `DATE` etc. This enables the addition of any necessary functionality in the wrapper for example, for serialising and deserialising in and out of XML.

12 Comparison with HL7v3 Types

12.1 Scope

Some HL7v3 types are not modelled in *open*EHR. HL7v3 V3DT types which are assumed by *open*EHR to exist in the underlying type system of any implementation technology include:

- Integer (`INT`)
- Real (`REAL`)
- Set (`SET`)
- List (`LIST`)
- Bag (`BAG`)

HL7v3 types which are not modelled here because they are almost always too volatile for concrete modelling, and can be created with archetyped generic information structures are as follows (even in HL7 they are really data structures rather than data types):

- Postal address (`AD`)
- Entity name (`EN`)
- Person name (`PN`)
- Organisation name (`ON`)
- Trivial name (`TN`)

These types are all modelled by archetyped spatial data structures in *open*EHR (equivalent to sub-types of `Structure` in the CDA specification).

HL7v3 types which may need to be modelled in the future include:

- Uncertain value probabilistic (`UVP`)
- Non-parametric probability distribution (`NPPD`)
- Parametric probability distribution (`PPD`)

Types which are provided by *open*EHR but not supported directly by HL7 include:

- state variable (`DV_STATE`);
- ordinal values (`DV_ORDINAL`);
- explicit partial date and time types (`DV_DATE`, `DV_TIME`);
- explicit time duration (`DV_DURATION`).

Types in the latter two categories may be implementable with the TS (timestamp) type.

12.2 Design Differences

There are some significant differences in design approach between the *open*EHR data types and the HL7v3 data types, described in the following sections.

12.2.1 Naming

All types in the HL7 specification have two names, one short and one long. For example the type representing physical quantities is known both as “PhysicalQuantity” and “PQ”. While short names may be reasonable for often-used types, someshort names are not obvious, e.g. “EN”, “ON”, “TN”, “NPPD” etc. Short names certainly have benefits for drawing tools such as Rational Rose or other

UML tools, however, it is questionable whether a formal model should include concept names chosen on the basis of visual appearance in such tools (one might argue that such tools should provide aliases for visual purposes, without changing the actual model). Another problem is that UML does not include the concept of class name aliases, nor do most programming languages.

The *open*EHR model uses one name only for each class.

12.2.2 Identification

The HL7 V3DT includes the types II, UID, OID and UUID. The II type is claimed to be for identifying all kinds of entities, which we here classify as real-world entities ("RWEs") (such as people, vehicle registrations, invoices) and informational entities ("IEs") - which in general are snapshots of data representing an RWE in a computer system. One problem with RWE identification schemes is that some are known (e.g. social security number) to produce fallible identifiers or situations where multiple RWEs have the same identifier, or no identifier at all. Conversely, with well-controlled and internationally agreed ways of issuing/generating information system identifiers (e.g. GUID, ISO OID) it is thought that such identifiers can be made reliable, and indeed correspond 1:1 with their intended IEs. However, a problem with IEs is that there are often duplicates and also multiple versions in time, each intended to represent the same RWE (such as a particular person, observation or composition).

As far as can be ascertained currently, there is no standard analysis taking into account the existence of IEs and RWEs, and recognising the fact that multiple versions and/or duplicates may refer to the same RWE.

The approach taken in *open*EHR with respect to identifiers is currently as follows.

- RWE identifiers such as social security numbers, licence numbers, etc are modelled with the type `DV_IDENTIFIER`, which has the attributes:
  - issuer: String
  - id: String
  - type: String

  The attributes listed above are nearly the same as for the HL7 II type, indicating that the two types may be compatible.
- Identification of IEs is done using the type `OBJECT_ID`, which is not a data type, and is documented in the Support Information Model. The `OBJECT_ID` type takes into account the fact that there may be multiple IEs referring to the same underlying RWE by adding a version identifier (assumed to be a timestamp).

12.2.3 Archetyping

The *open*EHR data types are defined on the assumption of archetype-based systems. While they will work perfectly well in systems which know nothing about archetyping, some types are not defined because archetypable structures built from more basic entities are assumed instead, rather than concretely modelled data types. These include "types" for address and person name which are found in HL7v3 and CEN 13606.

12.2.4 Treatment of Inbuilt Types

The HL7v3 data types do not make any assumptions about the existence of types typically built-in to most object and relational formalisms, such as the basic types `String`, `Integer`, `Boolean`, `Real`, `Double`, and the generic types `Set<T>`, `Bag<T>` and `Array<T>`. Hence, the types `ST`, `INT`, `REAL`, `BL`, `SET<T>`, `BAG<T>` and so on are redefined by HL7. The supposed advantage of this approach is that the semantics of all types in the HL7 system, right down to atomic data items are self-contained

in the definition, and do not rely on external semantics. Possible problems with this approach include the following.

- The HL7 definitions diverge from the OMG IDL and ISO 11404 definitions of the basic data types, which could cause unexpected problems in software development and data processing which is done in typical development technologies (object-oriented and relational).
- The HL7 types `INT` and `REAL` are defined as subtypes of the `QTY` type, a relationship that does not exist in any object-oriented formalism for these types (in particular, there is no substitutability of a type called Integer or Real for a type called Qty built in to any object language). The definitions of `INT` and `REAL` are also different from those found in most object formalisms. This might cause some difficulty in implementation.
- The binary data type `BIN` is represented as a `List<BL>` (where each item can be True, False, null), whereas it would normally be expected to be something like `Array<Octet>` (i.e. an array of bytes) in most software environments. There does not appear to be any utility in defining it as `List<BL>`, since binary data is almost without exception represented and processed as contiguous arrays of machine bytes.
- The string type `ST` inherits from the encapsulated data type `ED`, which in turn inherits from the binary data type `BIN`. The result of this is that an instance of `ST` contains numerous data attributes relating to multi-media data, and the content is presumably represented as a `List<BL>`. This is a major departure from the standard understanding of a string in computer sciences, which is usually simply an array of characters.
- The HL7 boolean type `BL` is a three-valued logic type due to the null marker approach (see below), not the usual two-valued type found in the Boolean concept in programming languages. The same is true of `INT` and `REAL`: due to the null marker design, "null" is a possible return value of an integer or real as well as true integer and real values.

In general, where differences exist between same-named types in HL7 and an underlying formalism such as a programming language, there is likely to be some confusion in implementation. Further, there is likely to be confusion in how to process instances of basic types which contain numerous (and sometimes recursive) fields which are not used in the standard specifications of basic types.

The *open*EHR approach with respect to inbuilt types is to assume only those types found in the mainstream object-oriented programming languages, and in particular, definitive formalisms like OMG IDL and XML. While this means there there is in theory less control over these types than in the HL7 approach, the number of types involved is quite small, and the problem of bindings to the basic types of object formalisms is well understood. Additionally, since it is recognised that some data types defined by *open*EHR could clash with types found in some languages and libraries, all data type class names are prefaced with "DV_" to avoid naming confusion, and to allow implementations of *open*EHR types to co-exist with existing types in implementation formalisms.

12.2.5 Use of Null Markers

All HL7 data types inherit from the `ANY` class (equivalent to the `DATA_VALUE` class in *open*EHR) which contains the attributes:

```
BL nonNull;
CS nullFlavor;
BL isNull;
```

The purpose of these attributes is to indicate whether a datum is Null, and for what reason. Since some data type classes also appear as the attributes of other data types, the Null markers also indicate

whether any part of a datum is null. Thus, in the class `Interval<T>` shown below, all attributes have the possibility of containing a Null marker.

```
type Interval<T> alias IVL<T> extends Set<T>
{
        T low;
        BL lowClosed;
        T high;
        BL highClosed;
        T.diff width;
        T center;
        IVL<T> hull(IVL<T> x);
literal ST;
promotion IVL<T> (T x);
demotion T;
};
```

For example, this allows an interval with missing ends and width to exist as a structured type. The consequence of the approach is that the entire model is essentially a model of "partial" data types; *any attribute and any function call may return a Null value, as well as the true values of its type* (in fact, in the specification, Null values are defined to be valid values of all data types). This design decision was taken in HL7 so that any datum, no matter how unknown, would be structurally representable in the same way as completely known data, enabling it to be processed in the same way as all other instances of the same type.

However, an important object-oriented design principle has been ignored in this approach. In the proper design of classes, properties and *class invariants* are stated. Invariants are statements which describe the correctness conditions of instances of the class; the general rule is that the post-condition of a creation routine (constructor) of a class must be that the invariants are satisfied. For example, an invariant of the HL7 `IVL<T>` class could be:

```
(exists(low) and exists(high)) or else
(exists(low) and exists(width)) or else
(exists(width) and exists(high))
```

When an instance of this class is created, this condition should be satisfied, and remain satisfied for the life of the instance. To do otherwise is to create instances of data which other software can make no assumptions about, and is forced to check every single field, and then determine what to do in an *ad hoc* way. (See [6] p366, [4] p43, [5] p29 for detailed explanations of the invariant concept).

Possible consequences of the built-in Null marker design approach include:

- since even HL7's basic types `ST`, `INT`, `REAL`, `LIST<>`, `SET<>` include null markers, processing of null values will be pervasive at the lowest level;
- software will be more complex, both implementations of the data types, and of software which handle them. This is because the software always has to deal with the possibility of calls to routines and attributes returning Null values. Most clinical information systems to date have taken the approach that a datum is either represented as an instance of a formal type if fully known, or else as narrative text if only partial;
- data may not be always be safely processable, since some software may not properly handle the null values associated with attributes of partially known data items. Essentially, all software which processes the data has to be "null-value aware", and make no assumptions at all about whether a particular data instance is valid or not.

The HL7 data type model is in contrast with simpler approaches such as used in CEN, GEHR, and *open*EHR, where data types are formal models of types such as `Coded_term`, `Quantity` and so on.

Rather than build the possibility of null markers into every attribute and class in the data types, a single null marker is defined in relevant containing classes. This decision is based on the principle that data types should be defined independently of their context of use. Hence, where data types are used as data values, such as in the *value* attribute of the class `ELEMENT` from the *open*EHR EHR reference model, the parallel features *is_null* and *null_flavour* are also defined. However, where data types appear as attributes elsewhere in the model and there is no possibility of them being null, no null markers are used. FIGURE 13 shows visually the difference between the two approaches.

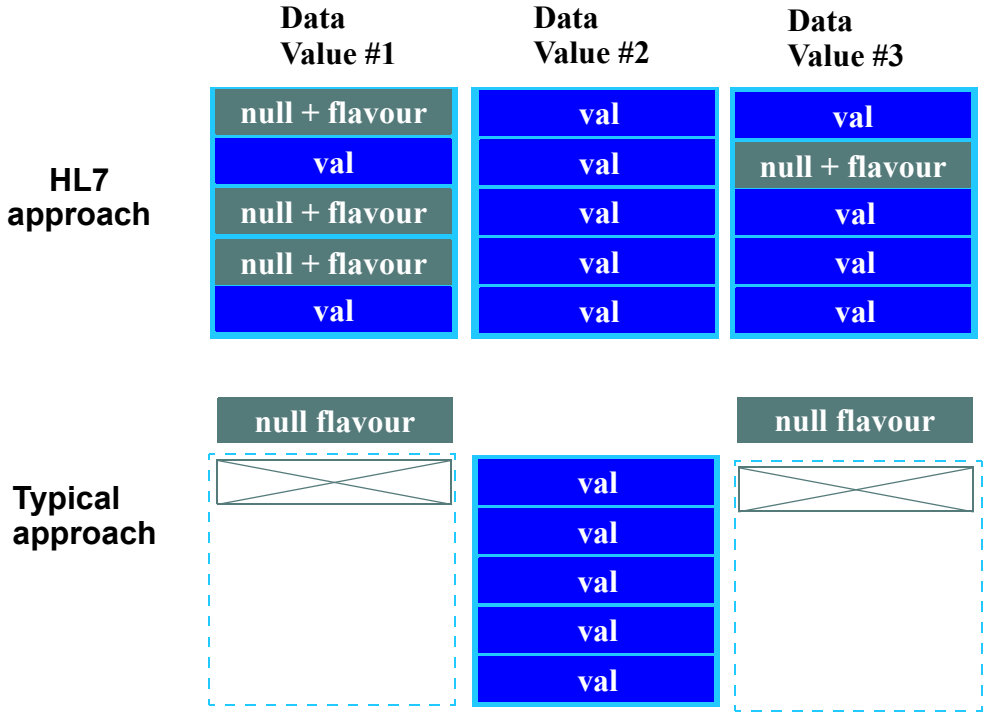


FIGURE 13 HL7 and Typical Null value approaches.

The consequences of the standard software-engineering approach include:

- data types can be more easily formally specified, since the semantics of invariants, attributes and operations do not need to include the possibility of null values;
- software implementations are simpler;
- data are always guaranteed to be safely processable for decision support and general querying, since no instance of a formal type will be created in the first place if the datum is very unreliable;
- null markers only appear in models where they are relevent, rather than everywhere data types are used;
- however, the *open*EHR data types do not automatically deal with missing or unknown internal attribute values (such as missing high *and* low values for an interval, partial dates etc).

In order to deal with the last possibility, various approaches are used in *open*EHR:

- for most data which is not fully known, no data type instance is created, and a null marker is created. Depending on the design of the revelant archetypes, there will usually be the possibility of recording the datum in narrative form;
- `ENTRYs` in the *open*EHR EHR reference model include a *certainty*:`Boolean` attribute, for recording a level of doubt;

- for particular data types which are often partial, special features are defined. The main types affected are DV_DATE, DV_TIME, and DV_DATE_TIME; the properties *month_unknown*, *day_unknown*, *minute_unknown*, and *second_unknown* (based on ISO 8601 semantics) are used to define explicitly the semantics of dates with a missing day, times with missing seconds and so on;
- Intervals of date/time types include types generated when the parameter type is one of the partial classes, thus, types DV_INTERVAL<DV_DATE> (where one or both ends has an unknown part) are possible. This covers the need for intervals in which some date is missing from the end date/times, while not allowing intervals with completely missing items to be created;
- for expressing uncertainy more precisely, probability distribution data types (based on the types defined in HL7) can be used.

A consequence of the HL7 model is that data instances represented in XML or another structured text format will be structurally the same regardless of whether there are Null values or not. A structured form for partially known data (which would normally break the invariants of its class) may well be useful for representing the data as part of a text field, making it easier to use for whatever processing is possible later on.

12.2.6 Terminology Approach

The approach in *open*EHR is to assume the existence of a Terminology Server which is the sole authoritative interface with terminologies of any kind, and is the only entity which can assume responsibility for querying, post-coordination or other manipulations of terms. No allowance is made for coordination of "modifiers", "qualifiers" or any other terms outside the service. As a consequence, there are no coordination facilities in the type DV_CODED_TEXT, a departure from earlier versions of the specification - any term provided from the terminology service must already be "coordinated", either by the terminology service, or by one of the terminologies it accesses. This places the responsibility of combining terms firmly in the knowledge part of the system, and prevents unsanctioned, unvalidated combinations being created elsewhere.

12.2.7 Date/Time Approach

The HL7 specification uses a single TS type to represent all logical dates, times, date/times, and partial versions thereof. The *open*EHR specification defines distinct types for each, since these are the types which occur in the real world, and it is easier to specify correctness constraints with this approach. It is recognised that a single type may be used by some implementors (depending on what is available in the language being used), however, the recommended practice is to wrap any such types with the logical types described in this specification. This approach reduces the possibility for any errors in transmitted data (since no strange combinations of year, ... , second can occur not explicitly described in the type definitions).

12.2.8 Time Specification Types

The HL7 approach for time specification appears to cover all reasonable requirements, but has some minor problems, including:

- the types PIVL and EIVL are declared as being generic types (i.e. PIVL<T:TS>, EIVL<T:TS>), when there appears to be no reason for this;
- the PIVL.*phase* attribute is used to represent an interval during which a activity occurs, example given is "2 minutes every 8 hours". However, the "2 mins" is almost always part of a therapeutic prescription of some kind, not part of the timing specification as such. Thera-

peutic prescriptions have the form "do X every Y time", where the X describes what to do, and how long to do it for (e.g. 40 mins massage, administer a drug slowly over 10 mins). In fact, what we are really interested in with a timing specification is the specification of the *starting points in time* of some activity, not a time-based graph of on/off points, whch is effectively what the `PIVL` type is now.

12.2.9 Type Conversions

The HL7v3 data types specification allows various type conversions, as follows:

> Three kinds of type conversions are defined: promotion, demotion, and character string literals. Type conversions can be implicit or explicit. Implicit type conversion occurs when a certain type is expected (e.g. as an argument to a statement) but a different type is actually provided.

One notable kind of conversion possible in HL7 is of a value of any type `T` into an instance of `Set<T>`, `List<T>`, `Bag<T>` or `IVL<T>` containing the value.

The *open*EHR model does not provide for any type conversions other than those automatically available between inbuilt basic numeric types such as Integer, Float and Double, and between types related by inheritance, as supported by all object-oriented languages.

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 Booch G. *Object-Oriented Analysis and Design with applications*. 2nd ed. Benjamin/Cummings 1994.

5 Kilov H. *Information Modelling - an object-oriented approach*. Prentice Hall 1994.

6 Meyer B. *Object-oriented Software Construction*, 2nd Ed. Prentice Hall 1997.

7 Richards E G. *Mapping Time - The Calendar and its History*. Oxford University Press 1998.

8 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

9 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

10 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

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

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

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

A.4 GEHR Australia

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

A.5 HL7

15 Schadow G, Biron P. HL7 version 3 deliverable: *Version 3 Data Types*. (2nd ballot 2002 version).

END OF DOCUMENT