Release 1

The *open*EHR Reference Model

Support Information Model

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

Revision: 1.6

Pages: 43

1. Ocean Informatics Australia

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

The *open*EHR foundation

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

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

Copyright Notice

Amendment Record

Issue	Details	Raiser	Completed
RELEASE 1.0.1			
1.6	**CR-000209**: Minor changes to correctly define AUTHORED_RESOURCE.*current_revision*. Add minimal definition for List<T> class. **CR-000xxx**. Move INTERVAL class definition to correct section. Add two invariants in INTERVAL. **CR-000202**: Correct minor errors in VERSION.*preceding_version_id*. Added *is_first* function and invariant to VERSION_TREE_ID class. **CR-000204**: Add generic id subtype of OBJECT_ID.	Y S Lim S Heard S Heard, H Frankel H Frankel	21 Apr 2006
RELEASE 1.0			
1.5	**CR-000162**. Allow party identifiers when no demographic data. Relax invariant on PARTY_REF. **CR-000184**. Separate out terminology from Support IM. **CR-000188**: Add *generating_type* function to ANY for use in invariants **CR-000161**. Support distributed versioning. Move OBJECT_ID.*version* to subtypes. Add OBJECT_VERSION_ID, VERSION_TREE_ID and LOCATABLE_REF types.	S Heard H Frankel T Beale T Beale T Beale H Frankel	06 Feb 2006
RELEASE 0.96			
1.3	**CR-000135**: Minor corrections to rm.support.terminology package. **CR-000145**: Add class for access to external environment. **CR-000137**: Add definitions class to support.definition package.	D Lloyd D Lloyd D Lloyd	25 Jun 2005
RELEASE 0.95			
1.2.1	**CR-000129**. Fix errors in UML & specs of Identification package. Adjust invariants & postcondition of OBJECT_ID, HIER_OBJECT_ID, ARCHETYPE_ID and TERMINOLOGY_ID. Improve text to do with assumed abstract types Any and Ordered_numeric.	D Lloyd	25 Feb 2005
1.2	**CR-000128**. Update Support assumed types to ISO 11404:2003. **CR-000107**. Add support for exclusion and inclusion of Interval limits. **CR-000116**. Add PARTICIPATION.*function* vocabulary and invariant. **CR-000122**. Fix UML in Terminology_access classes in Support model. **CR-000118**. Make package names lower case. **CR-000111**. Move Identification Package to Support. **CR-000064**. Re-evaluate COMPOSITION.*is_persistent* attribute. Add "composition category" vocabulary. Re-ordered vocabularies alphabetically.	T Beale A Goodchild T Beale D Lloyd T Beale DSTC D Kalra	10 Feb 2005
RELEASE 0.9			
1.1	**CR-000047**. Improve handling of codes for structural attributes. Populated Terminology and code_set codes.	S Heard	11 Mar 2004

Issue	Details	Raiser	Completed
1.0	**CR-000091**. Correct anomalies in use of CODE_PHRASE and DV_CODED_TEXT. Add simple terminology service interface. **CR-000095**. Remove *property* attribute from Quantity package. Add simple measurement interface. **Formally validated using ISE Eiffel 5.4.**	T Beale DSTC, S Heard	09 Mar 2004
0.9.9	**CR-000063**. ATTESTATION should have a status attribute.	D Kalra	13 Feb 2004
0.9.8	**CR-000068**. Correct errors in INTERVAL class.	T Beale	20 Dec 2003
0.9.7	**CR-000032**. Basic numeric type assumptions need to be stated **CR-000041**. Visually differentiate primitive types in openEHR documents. **CR-000043**. Move External package to Common RM and rename to Identification (incorporates CR-000036 - Add HIER_OBJECT_ID class, make OBJECT_ID class abstract.)	DSTC, D Lloyd, T Beale	09 Oct 2003
0.9.6	**CR-000013**. Rename key classes. Based on CEN ENV13606. **CR-000038**. Remove *archetype_originator* from multi-axial archetype id. **CR-000039**. Change archetype_id section separator from ':' to '-'.	T Beale	18 Sep 2003
0.9.5	**CR-000036**. Add HIER_OBJECT_ID class, make OBJECT_ID class abstract.	T Beale	16 Aug 2003
0.9.4	**CR-000022**. Code TERM_MAPPING.*purpose*.	G Grieve	20 Jun 2003
0.9.3	**CR-000007**. Added forgotten terminologies for Subject_relationships and Provider_functions.	T Beale	11 Apr 2003
0.9.2	Detailed review by Ocean, DSTC, Grahame Grieve. Updated valid characters in OBJECT_ID.*namespace*.	G Grieve	25 Mar 2003
0.9.1	Added specification for BOOLEAN type. Corrected minor error in ISO 639 standard strings - now conformant to TERMINOLOGY_ID. OBJECT_ID.*version_id* now optional. Improved document structure.	T Beale	18 Mar 2003
0.9	Initial Writing. Taken from Data types and Common Reference Models. **Formally validated using ISE Eiffel 5.2.**	T Beale	25 Feb 2003

Acknowledgements

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

Table of Contents

1 Introduction

1.1 Purpose

This document describes the *open*EHR Support Reference Model, whose semantics are used by all *open*EHR Reference Models. 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.

1.2 Related Documents

Prerequisite documents for reading this document include:

- The *open*EHR Modelling Guide

1.3 Status

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

This document is available at `http://svn.openehr.org/specification/TAGS/Release-1.0/publishing/architecture/rm/support_im.pdf`.

The latest version of this document can be found at `http://svn.openehr.org/specification/TRUNK/publishing/architecture/rm/support_im.pdf`.

Blue text indicates sections under active development.

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. Please send requests for information to `info@openEHR.org`. Feedback should preferably be provided on the mailing list `openehr-technical@openehr.org`, or by private email.

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 Support Package

2.1 Overview

The Support Reference Model comprises types which are used throughout other *open*EHR models, but are defined elsewhere, either by standards organisations or which are accepted *de facto* standards. The package structure is illustrated in FIGURE 1.

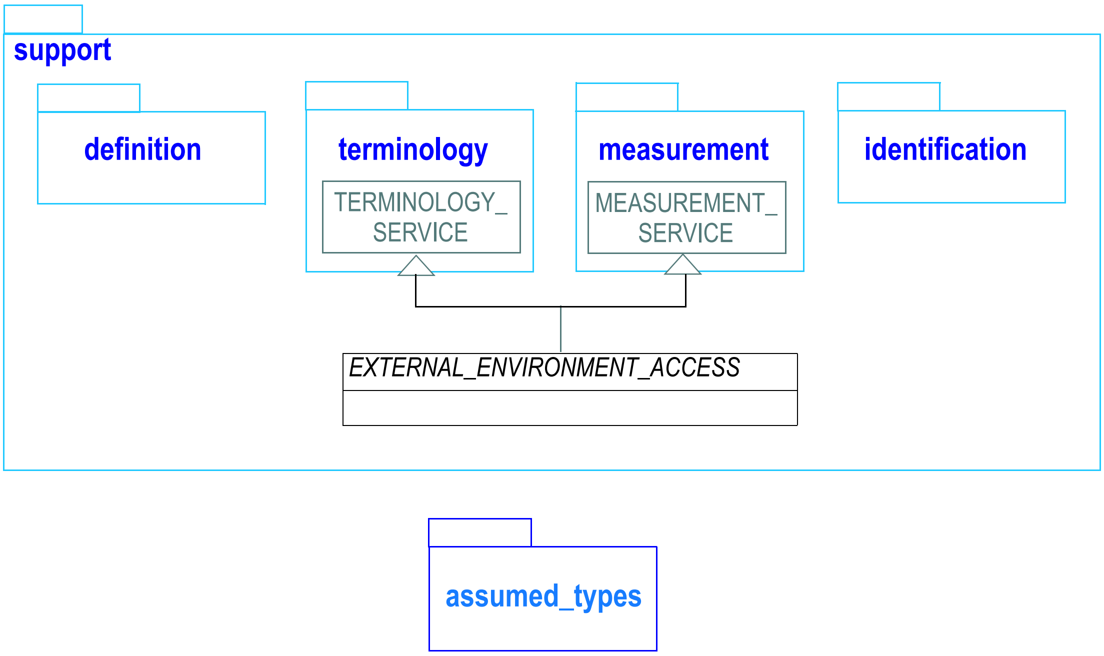

FIGURE 1 rm.support and assumed_types Packages

The four Support packages define the semantics respectively for constants, terms, scientific measurement and identifiers, which are assumed by the rest of the *open*EHR specifications. The class `EXTERNAL_ENVIRONMENT_ACCESS` is a mixin class providing access to external services.

2.2 Class Definitions

2.2.1 EXTERNAL_ENVIRONMENT_ACCESS Class

CLASS	***EXTERNAL_ENVIRONMENT_ACCESS (abstract)***	
Purpose	A mixin class providing access to services in the external environment.	
Functions	**Signature**	**Meaning**
	eea_terminology_svc: `TERMINOLOGY_SERVICE`	Return an interface to the terminology service
	eea_measurement_svc: `MEASUREMENT_SERVICE`	Return an interface to the measurement service

CLASS	*EXTERNAL_ENVIRONMENT_ACCESS (abstract)*
Invariants	***Terminology_service_exists***: eea_terminology_svc /= Void ***Measurement_service_exists***: eea_measurement_svc /= Void

3 Assumed Types

3.1 Overview

This section describes types assumed by all *open*EHR models. The set of types chosen here is based on a lowest common denominator set from threes sources, as follows.

- ISO 11404 (2003 revision).
- Well-known interoperability formalisms, including OMG IDL, W3C XML-schema.
- Well-known object-oriented programming languages, including C++, Java, C#, and Eiffel.

The intention in *open*EHR is to make the minimum possible assumptions about types found in implementation formalisms, while making sufficient assumptions to both enable *open*EHR models to be conveniently specified, and to allow the typical basic types of these formalisms to be used in their normal way, rather than being re-invented by *open*EHR. The ISO 11404 (2003) standard contains basic semantics of "general purpose data types" (GPDs) for information technology, and is used here as a normative basis for describing assumptions about types. The operations and properties described here are compatible with those used in ISO 11404, but not always the same, as 11404 has not chosen to use object-oriented functions. For example, the notional function `has(x:T)` (test for presence of a value in a set) defined on the type `Set<T>` below is not defined on the ISO 11404 Set type; instead, the function `IsIn(x: T; s: Set<T>)` is defined. However, in object-oriented formalisms, the function `IsIn` defined on a Set type would usually mean "subset of", i.e. true if this set is contained inside another set. In the interests of clarity for developers, an object-oriented style of functions and properties has been used here.

Two groups of assumed types are identified: primitive types, which are those built in to a formalism's type system, and library types, which are assumed to be available in a (class) library defined in the formalism. Thus, the type `Boolean` is always assumed to exist in a formalism, while the type `Array<T>` is assumed to be available in a library. For practical purposes, these two categories do not matter that much - whether `String` is really a library class (the usual case) or an inbuilt type doesn't make much difference to the programmer. They are shown separately here mainly as an explanatory convenience.

The assumptions that *open*EHR makes about existing types are documented below in terms of interface definitions. Each of these definitions contains *only the assumptions required for the given type to be used in the openEHR Reference Model* - **it is not by any means a complete interface definition**. The name and semantics of any function used here for an assumed type might not be identical to those found in some implementation technologies, but should be very close. Any mapping required should be stated in the relevant ITS. The definitions are compatible with the ISO 11404 standard, 2003 revision. Operation semantics are described formally using pre- and post-conditions. The keyword "Current" stands for "the current instance" (known as "this" or "self" in various languages). The keyword "like" anchors the type of the reference to the type of the object whose reference follows *like*. Not all types have definition tables - only those which add features to their inheritance parent have a table.

3.2 Inbuilt Primitive Types

The following types consititute the minimum built in set of types assumed by *open*EHR of an implementation formalism.

Type name in *open*EHR	Description	ISO 11404 Type
Character	represents a type whose value is a member of an 8-bit character-set (ISO: “repertoire”).	Character
Boolean	represents logical True/False values; usually physically represented as an integer, but need not be	Boolean
Integer	represents 32-bit integers	Integer
Real	represents 32-bit real numbers in any interoperable representation, including single-width IEEE floating point	Real
Double	type which represents 64-bit real numbers, in any interoperable representation including double-precision IEEE floating point.	Real

As shown in the table, *open*EHR assumes that Character is an 8-bit type. This is because the only use of Character in *open*EHR is in encapsulated data (*open*EHR Data Types), where the intention is to represent opaque data. Note that “octet” or “byte” may be the appropriate type names to map to in various programming languages.

FIGURE 2 illustrates the inbuilt types. Simple inheritance relationships are shown which facilitate the type descriptions below. A class “Any” is therefore used to stand for the usual top-level class in all object-oriented type systems, typically called something like “Any” or “Object”. Inheritance from or subsitutability for an `Any` class is not assumed at all in *open*EHR (hence the dotted lines in the UML). It is used to enable basic operations like ‘=’ to be described once for the type `Any`, rather than in every subtype. The type `Ordered_numeric` is on the other hand assumed for purposes of specification in the *open*EHR `data_types.quantity` package, and is intended to be mapped to an equivalent type in a real type system (e.g. in Java, `java.lang.Number`). Here it is assumed that the operations defined on `Ordered_numeric` are available on the types `Integer`, `Real` and `Double` in implementation type systems, where relevant. Data-oriented implementation type systems such as XML-schema are not expected to have such operations.

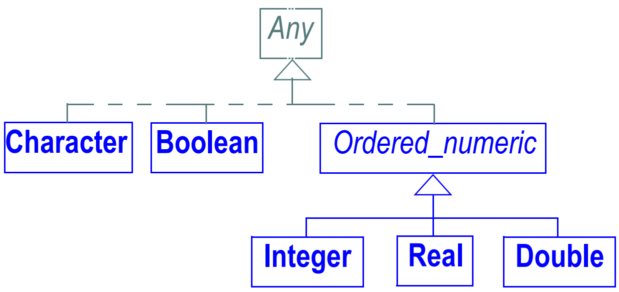

FIGURE 2 Primitive Types Assumed by *open*EHR

3.2.1 Any Type

INTERFACE	*Any (abstract)*	
Description	Abstract supertype. Usually maps to a type like “Any” or “Object” in an object system. Defined here to provide the value and reference equality semantics.	
Abstract	Signature	Meaning
	is_equal (other: `Any`): `Boolean`	Value equality
Functions	Signature	Meaning
	infix ‘=’ (other: `Any`): `Boolean`	Reference equality
	instance_of (a_type: `String`)	Dynamic type of object as a String. Used for type name matching.
Invariants		

3.2.2 Boolean Type

INTERFACE	Boolean	
Purpose	Boolean type used for two-valued mathematical logic.	
Abstract	Signature	Meaning
	infix "and" (other: `Boolean`): `Boolean` ***require*** *other_exists*: other /= void ***ensure*** *de_morgan*: Result = **not** (**not** Current **or not** other) *commutative*: Result = (other **and** Current)	Logical conjunction
	infix "and then" (other: `Boolean`): `Boolean` ***require*** *other_exists*: other /= void ***ensure*** *de_morgan*: Result = **not** (**not** Current **or else not** other)	Boolean semi-strict conjunction with *other*

INTERFACE	Boolean	
	infix "or" (other: `Boolean`): `Boolean` ***require*** *other_exists*: other /= void ***ensure*** *de_morgan*: Result = **not** (**not** Current **and not** other) *commutative*: Result = (other **or** Current) *consistent_with_semi_strict*: Result **implies** (Current **or else** other)	Boolean disjunction with *other*
	infix "or else" (other: `Boolean`): `Boolean` ***require*** *other_exists*: other /= void ***ensure*** *de_morgan*: Result = **not** (**not** Current **and then not** other)	Boolean semi-strict disjunction with `other'
	infix "xor" (other: `Boolean`): `Boolean` ***require*** *other_exists*: other /= void ***ensure*** *definition*: Result = ((Current **or** other) **and not** (Current **and** other))	Boolean exclusive or with `other'
	infix "implies" (other: `Boolean`): `Boolean` ***require*** *other_exists*: other /= void ***ensure*** *definition*: Result = (**not** Current **or else** other)	Boolean implication of `other' (semi-strict)
Invariants	*involutive_negation*: is_equal (**not** (**not** Current)) *non_contradiction*: **not** (Current **and** (**not** Current)) *completeness*: Current **or else** (**not** Current)	

3.2.3 Ordered_numeric Type

INTERFACE	*Ordered_numeric (abstract)*	
Purpose	Abstract notional parent class of ordered, numeric types, which are types which have various arithmetic and comparison operators defined. All ordered, quantified types (i.e. types with a notion of precise "magnitude") have these operations. Maps to various types in implementation technologies.	
Abstract	**Signature**	**Meaning**

INTERFACE	*Ordered_numeric (abstract)*	
	infix "*" (other: ***like*** Current): ***like*** Current ***require*** *other_exists*: other /= void ***ensure*** *result_exists*: Result /= void	Product by `other'. Actual type of result depends on arithmetic balancing rules.
	infix "+" (other: ***like*** Current): ***like*** Current ***require*** *other_exists*: other /= void ***ensure*** *result_exists*: Result /= void *commutative*: equal (Result, other + Current)	Sum with `other' (commutative). Actual type of result depends on arithmetic balancing rules.
	infix "-" (other: ***like*** Current): ***like*** Current ***require*** *other_exists*: other /= void ***ensure*** *result_exists*: Result /= void	Result of subtracting `other'. Actual type of result depends on arithmetic balancing rules.
	infix '<' (other: ***like*** Current): `Boolean`	Arithmetic comparison. In conjunction with '=', enables the definition of the operators '>', '>=', '<=', '<>'. In real type systems, this operator might be defined on another class for comparability.
Invariants		

3.3 Assumed Library Types

The types described in this section are also assumed to be fairly standard in implementation technologies by *open*EHR, but usually come from type libraries rather than be built into the type system of implementation formalisms.

Type name in *open*EHR	Description	ISO 11404: 2003 Type
`String`	represents unicode-enabled strings	`Character-String/ Sequence`
`Array<T>`	physical container of items indexed by number	`Array`
`List<T>`	container of items, implied order, non-unique membership	`Sequence`
`Set<T>`	container of items, no order, unique membership	`Set`
`Bag<T>`	container of items, no order, non-unique membership	`Bag`

Type name in *open*EHR	Description	ISO 11404: 2003 Type
`Hash<T, U:Comparable>`	a table of values of any type T, keyed by values of any basic comparable type U, typically String or Integer, but may be more complex types, e.g. a coded term type.	`Table`
`Interval<T>`	Intervals	

FIGURE 3 illustrates the assumed library types. As with the assumed primitive types, inheritance and abstract classes are used for convenience of the definitions below, but are not formally assumed in *open*EHR.

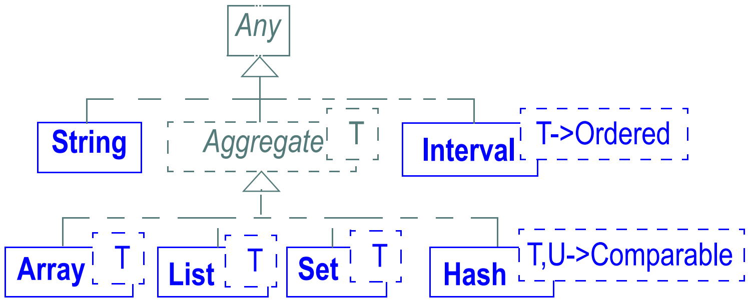

FIGURE 3 Library Types Assumed by *open*EHR

3.3.1 String Type

INTERFACE	String	
Description	Strings of characters, as used to represent textual data in any natural or formal language.	
Functions	**Signature**	**Meaning**
	infix '+' (other: `String`): `String`	Concatenation operator - causes 'other' to be appended to this string
	is_empty: `Boolean`	True if string is empty, i.e. equal to "".
	is_integer: `Boolean`	True if string can be parsed as an integer.
Invariants		

3.3.1.1 UNICODE

It is assumed in the *open*EHR specifications that Unicode is supported by the type `String`. Unicode is needed for all Asian, Arabic and other script languages, for both data values (particularly plain text and coded text) and for many predefined string attributes of the classes in the *open*EHR Reference Model. It encompasses all existing character sets.

3.3.2 Aggregate Type

INTERFACE	*Aggregate <T> (abstract)*	
Description	Abstract parent of of the aggregate types `List<T>`, `Set<T>`, `Bag<T>`, `Array<T>` and `Hash<T,K>`.	
Functions	**Signature**	**Meaning**
	has (v: `T`): `Boolean`	Test for membership of a value
	count: `Integer`	Number of items in container
	is_empty: `Boolean`	True if container is empty.
Invariants		

3.3.3 List Type

INTERFACE	*List <T> (abstract)*	
Description	Ordered container that may contain duplicates.	
Functions	**Signature**	**Meaning**
	first: `T`	Return first element.
	last: `T`	Return last element.
Invariants	***First_validity***: **not** is_empty **implies** first /= Void ***Last_validity***: **not** is_empty **implies** last /= Void	

3.3.4 Hash Type

INTERFACE	Hash <T, U: Comparable>	
Description	Type representing a keyed table of values. T is the value type, and U the type of the keys.	
Functions	**Signature**	**Meaning**
	has_key (a_key: `U`): `Boolean`	Test for membership of a key
	item (a_key: `U`): `T`	Return item for key 'a_key'. Equivalent to ISO 11404 *fetch* operation.
Invariants		

3.3.5 Interval Type

INTERFACE	**Interval <T:Ordered>**	
Purpose	Interval of ordered items.	
Attributes	**Signature**	**Meaning**
	lower: `T`	lower bound
	upper: `T`	upper bound
	lower_unbounded: `Boolean`	lower boundary open (i.e. = -infinity)
	upper_unbounded: `Boolean`	upper boundary open (i.e. = +infinity)
	lower_included: `Boolean`	lower boundary value included in range if not *lower_unbounded*
	upper_included: `Boolean`	upper boundary value included in range if not *upper_unbounded*
Functions	**Signature**	**Meaning**
	has(e:`T`): `Boolean`	True if (lower_unbounded **or** ((lower_included **and** v >= lower) **or** v > lower)) **and** (upper_unbounded **or** ((upper_included **and** v <= upper **or** v < upper)))
Invariants	***Lower_included_valid***: lower_unbounded **implies not** lower_included ***Upper_included_valid***: upper_unbounded **implies not** upper_included ***Limits_consistent***: (**not** upper_unbounded **and not** lower_unbounded) **implies** lower <= upper ***Limits_comparable***: (**not** upper_unbounded **and not** lower_unbounded) **implies** lower.strictly_comparable_to(upper)	

3.4 Date/Time Types

Although the ISO 11404 (2003) standard defines a date-and-time type generator (section 8.1.6), and a `timeinterval` type (section 10.1.6), the reality is that dates and times are provided in significantly differing ways in implementation formalisms, and as a result, *open*EHR assumes nothing at all about them. Accordingly, types for date, time, date/time and duration are defined in the *open*EHR Data Types Information Model, ensuring standardised meanings of these types within *open*EHR. ISO 8601 is used as the normative basis for both string literal representation and properties chosen within these models.

4 Identification Package

4.1 Overview

The `identification` package describes a model of references and identifiers for information entities only and is illustrated in FIGURE 4. Real-world entity identifiers are defined in the *open*EHR Data Types information model.

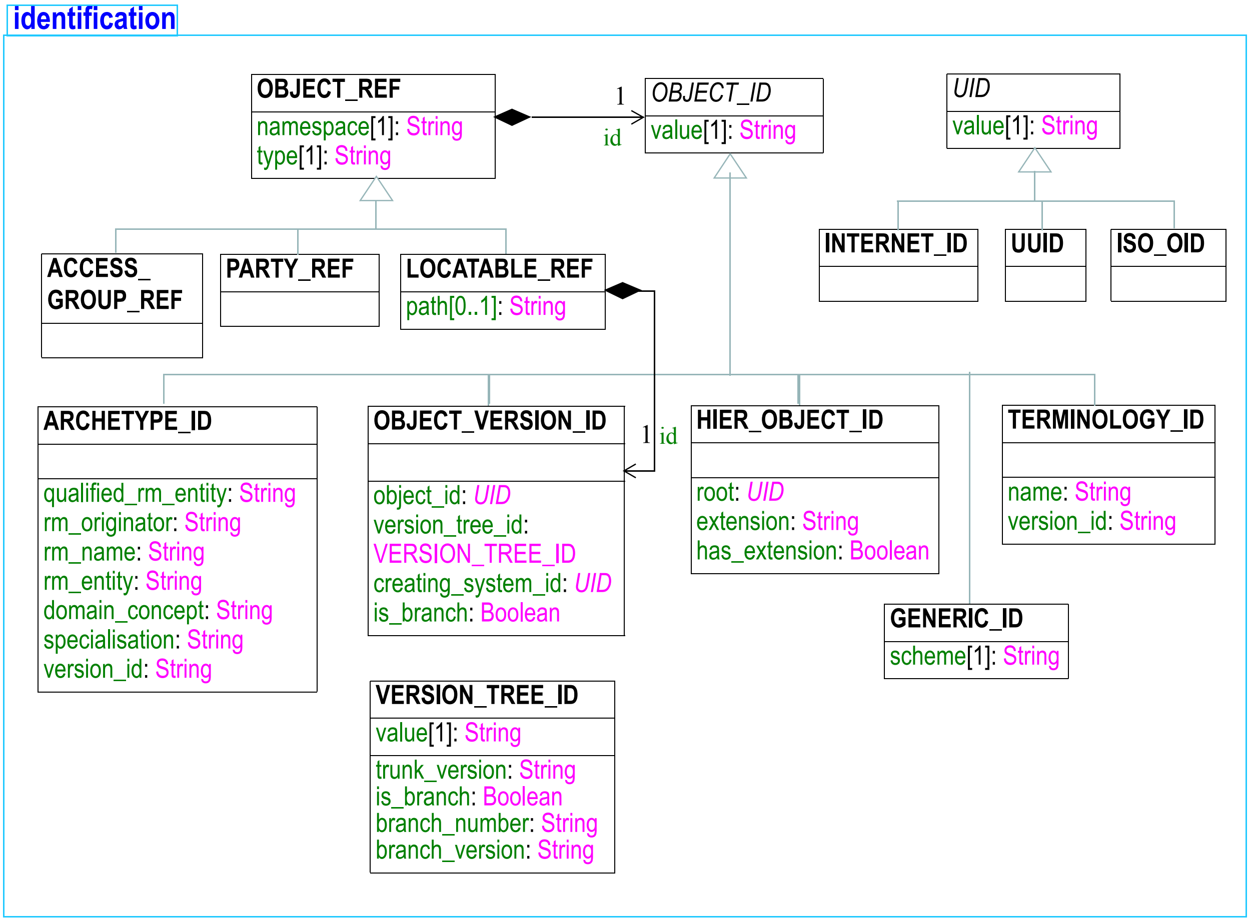

FIGURE 4 rm.support.identification Package

4.1.1 Requirements

Identification of entities both in the real world and in information systems is a non-trivial problem. The scenarios for identification across systems in a health information environment include the following:

- real world identifiers such as social security numbers, veterans affairs ids etc can be recorded as required by health care facilities, enterprise policies, or legislation;
- identifiers for informational entities which represent real world entities or processes should be unique;
- it should be possible to determine if two identifiers refer to information entities that are linked to the same real world entity, even if instances of the information entities are maintained in different systems;

- versions or changes to real-world entity-linked informational entities (which may create new information instances) should be accounted for in two ways:
  - it should be possible to tell if two identifiers refer to distinct versions of the same informational entity in the same version tree;
  - it should not be possible to confuse same-named versions of informational entities maintained in multiple systems which purport to represent the same real world entity. E.g. there is no guarantee that two systems' "latest" version of the Person "Dr Jones" is the same.

  Medico-legal use of information relies on previous states of information being identifiable in some way.
- it should be possible for an entity in one system or service (such as the EHR) to refer to an entity in another system or service in such a way that:
  - the target of the reference is easily finable within the shared environment, and
  - the reference does is valid regardless of the physical architecture of servers and applications.

The following subsections describe some of the features and challenges of identification.

Identification of Real World Entities (RWEs)

Real world entities such as people, car engines, invoices, and appointments all have identifiers. Although many of these are designed to be unique within a jurisdiction, 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. It should also be the case that if two RWE identifiers are equal, they refer to the same RWE.

Identification of Informational Entities (IEs)

As soon as information systems are used to record facts about RWEs, the situation becomes more complex because of the intangible nature of information. In particular:

- the same RWE can be represented simultaneously on more than one system ("spatial multiplicity");
- the same RWE may be represented by more than one "version" of the same IE in a system ("temporal multiplicity").

At first sight, it appears that there can also be purely informational entities, i.e. IEs which do not refer to any RWE, such as books, online-only documents and software. However, as soon as one considers an example it becomes clear that there is always a notional "definitive" or "authoritative" (i.e. trusted) version of every such entity. These entities can better be understood as "virtual RWEs". Thus it can still be said that multiple IEs may refer to any given RWE.

The underlying reason for the multiplicity of IEs is that "reality" - time and space - in computer systems is not continuous but discrete, and each "entity" is in fact just a snapshot of certain attribute values of a RWE.

If identifiers are assigned to IEs without regard to versions or duplicates, then no assertion can be made about the identified RWE when two IE ids are compared.

Identification of Versions of Informational Entities

The notion of "versioning" applies only to informational entities, i.e. distinct instances of content each representing a snapshot of some notional information. Where such instances are stored and managed in versioned containers, within a versioning system of some kind, explicit identification of the versions is required. The requirements are discussed in detail in the Common IM, `change_control` package. They can be summarised as follows:

- it must be possible to distinguish two versions of the same notional entity, i.e. know from the identifier if they are the same or different versions of the same thing;
- it must be possible to tell the relationship between the items in a versioned lineage, from the version identifiers.

Referencing of Informational Entities

Within a distributed information environment, there is a need for entities not connected by direct references in the same memory space to be able to refer to each other. There are two competing requirements:

- that the separation of objects in a distributed computing environment not compromise the semantics of the model. At the limit, this mandates the use of proxy types which have the same abstract interface as the proxied type; i.e. the "static" approach of Corba.
- that different types of information can be managed relatively independently; for example EHR and demographic information can be managed by different groups in an organisation or community, each with at least some freedom to change implementation and model details.

4.1.2 Identifying Real World Entities (RWE)

In *open*EHR, Real world entities are identified with a multipart identifier expressed in the data type `DV_IDENTIFIER`. This type should be used to express lab result identifiers, veterans affairs numbers and so on, i.e. any identifier issued by an organisation and corresponding to a *continuant* (an entity that continues to exist even if its attributes change over time).

4.1.3 Identifying Informational Entities (IEs)

The class `OBJECT_ID` is an abstract model of identifiers of IEs. It is assumed *a priori* that there can in general be more than one IE referring to the same underlying real world entity (RWE), such as a person or invoice; this is due to the possible existence of multiple copies, and also multiple versions. An `OBJECT_ID` therefore implicitly refers to a version of something; two versions of a Person object must have two distinct `OBJECT_ID`s. The rule for versioning is that if any attribute value of the IE changes, a new `OBJECT_ID` must be generated. Some `OBJECT_ID` subtypes explicitly model a version identifier. In practice, it can usually be omitted for ids of terminologies, where the terminology obeys the rule that a given code never changes its meaning through all versions of the terminology (i.e. ICD10 code F40.0 will mean "Agoraphobia" for all time (in English)).

The subtype `HIER_OBJECT_ID` defines a hierarchical identifier model, along the lines of ISO Oids; it includes the attributes *root* and *extension*, to make up a complete, unique identifier. The *root* attribute is of type `UID`, meaning it has the properties of a timeless unique object identifier. Subtypes of `UID` include the `ISO_OID` and `UUID` types. The latter models a DCE UUID (also known as a GUID).

The other subtypes, `ARCHETYPE_ID` and `TERMINOLOGY_ID` define different kinds of identifier, the former being a multi-axial identifier for archetypes, and the latter being a globally unique single string identifier for terminologies.

4.1.4 Identifying Versions of Informational Entities

The scheme used in *open*EHR for identifying versions uses a three-part identifier, consisting of:

- the identifier of the version container, in the form of an OBJECT_ID;
- the location in the version tree, as a 1- or 3-part numeric identifier, where the latter type expresses branching;
- the identifier of the system in which this version was created.

Under this scheme, multiple versions in the same container all have the same value for the first identifier, while their location in the version tree is given by the combination of the version tree identifier and the identifier of the creating system.

The format of the *creating_system_id* attribute is not currently fixed, hence its type is HIER_OBJECT_ID, allowing for various possibilities. The requirements on this identifier are that it be unique per system, and that it be easy to obtain or generate. It is also helpful if it is a meaningful identifier. The two most practical candidates appear to be GUIDs (which are not meaningful, but are easy to generate) and reverse internet domain identifiers, as recommended in [3] (these are easy to determine if the system has an internet address, and are meaningful and directly processible, however unconnected systems pose a problem). ISO Oids might also be used. All of these identifier types are accommodated via the use of HIER_OBJECT_ID.

A full explanation of the version identification scheme and its capabilities is given in the change_control section of the Common IM.

4.1.5 Referring to Informational Entities

All OBJECT_IDs are used as identifier attributes within the thing they identify, in the same way as a database primary key. To *refer* to an identified object, an instance of the class OBJECT_REF is required, in the same way as a database foreign key. OBJECT_REF is provided as a means of distributed referencing, and includes the object namespace (typically 1:1 with some service, such as "terminology") and type. The general principle of object references is to be able to refer to an object available in a particular namespace or service. Usually they are used to refer to objects in other services, such as a demographic entity from within an EHR, but they may be used to refer to local objects as well. The type may be the concrete type of the referred-to object (e.g. "GP") or any proper ancestor (e.g. "PARTY"). The notion of object reference provided here is a compromise between the static binding notion of Corba (where each model is dependent on all the interface details of the classes in other models) and a purely dynamic referencing scheme, where the holder of a reference cannot even tell what type of object the reference points to.

4.2 Class Descriptions

4.2.1 OBJECT_REF Class

CLASS	OBJECT_REF
Purpose	Class describing a reference to another object, which may exist locally or be maintained outside the current namespace, e.g. in another service. Services are usually external, e.g. available in a LAN (including on the same host) or the internet via Corba, SOAP, or some other distributed protocol. However, in small systems they may be part of the same executable as the data containing the Id.

CLASS	OBJECT_REF	
Attributes	**Signature**	**Meaning**
	id: *OBJECT_ID*	Globally unique id of an object, regardless of where it is stored.
	namespace: String	Namespace to which this identifier belongs in the local system context (and possibly in any other *open*EHR compliant environment) e.g. "terminology", "demographic". These names are not yet standardised. Legal values for the namespace are `"local" | "unknown" | "[a-zA-Z][a-zA-Z0-9_-:/&+?]*"`
	type: String	Name of the class (concrete or abstract) of object to which this identifier type refers, e.g. "PARTY", "PERSON", "GUIDELINE" etc. These class names are from the relevant reference model. The type name "ANY" can be used to indicate that any type is accepted (e.g. if the type is unknown).
Invariant	***Id_exists***: id /= Void ***Namespace_exists***: namespace /= Void ***and then not*** namespace.empty ***Type_exists***: type /= Void ***and then not*** type.empty	

4.2.2 ACCESS_GROUP_REF Class

CLASS	ACCESS_GROUP_REF	
Purpose	Reference to access group in an access control service.	
Inherit	OBJECT_REF	
Functions	**Signature**	**Meaning**
Invariant	***Type_validity***: type.is_equal("ACCESS_GROUP")	

4.2.3 PARTY_REF Class

CLASS	PARTY_REF	
Purpose	Identifier for parties in a demographic or identity service. There are typically a number of subtypes of the PARTY class, including PERSON, ORGANISATION, etc. Abstract supertypes are allowed if the referenced object is of a type not known by the current implementation of this class (in other words, if the demographic model is changed by the addition of a new PARTY or ACTOR subtypes, valid PARTY_REFs can still be constructed to them).	
Inherit	OBJECT_REF	
Functions	Signature	Meaning
Invariant	***Type_validity***: type.is_equal("PERSON") **or** type.is_equal("ORGANISATION") **or** type.is_equal("GROUP") **or** type.is_equal("AGENT")**or** type.is_equal("ROLE") **or** type.is_equal("PARTY") **or** type.is_equal("ACTOR")	

4.2.4 LOCATABLE_REF Class

CLASS	LOCATABLE_REF	
Purpose	Reference to a LOCATABLE instance inside the top-level content structure inside a VERSION<T>; the *path* attribute is applied to the object that VERSION.*data* points to.	
Inherit	OBJECT_REF	
Attributes	Signature	Meaning
1..1 (redefined)	**id**: OBJECT_VERSION_ID	The identifier of the Version.
0..1	**path**: String	The path to an instance in question, as an absolute path with respect to the object found at VERSION.*data*. An empty path means that the object referred to by id being specified.
Functions	Signature	Meaning
	as_uri: String	A URI form of the reference, created by concatenating the following: "ehr://" + id.value + "/" + path
Invariant	***Path_valid***: path /= Void **implies not** path.is_empty	

4.2.5 OBJECT_ID Class

CLASS	*OBJECT_ID (abstract)*	
Purpose	Ancestor class of identifiers of informational objects. Ids may be completely meaningless, in which case their only job is to refer to something, or may carry some information to do with the identified object.	
Use	Object ids are used inside an object to identify that object. To identify another object in another service, use an `OBJECT_REF`, or else use a UID for local objects identified by UID. If none of the subtypes is suitable, direct instances of this class may be used.	
Attributes	Signature	Meaning
	value: `String`	The value of the id in the form defined below.
Invariant	***Value_exists***: value /= Void ***and then not*** value.empty	

4.2.6 HIER_OBJECT_ID Class

CLASS	HIER_OBJECT_ID	
Purpose	Hierarchical identifiers consisting of a root part and an optional extension.	
HL7	The HL7v3 II Data type.	
Functions	Signature	Meaning
	root: `UID`	The identifier of the conceptual namespace in which the object exists, within the identification scheme.
	has_extension: `Boolean`	True if there is an extension part.
	extension: `String`	Optional local identifier of the object within the context of the root identifier.
Invariant	***Root_valid***: root /= Void ***Extension_validity***: has_extension **xor** extension = Void	

4.2.6.1 Identifier Syntax

The syntax of the *value* attribute by default follows the following production rules (EBNF):

```
value:       root [ '::' extension ]
root:        uid                                    -- see UID below
extension:   alpha_numeric_string
```

4.2.7 OBJECT_VERSION_ID Class

CLASS	OBJECT_VERSION_ID	
Purpose	Globally unique identifier for one version of a versioned object.	
Inherit	*OBJECT_ID*	
Functions	**Signature**	**Meaning**
1..1	**object_id**: UID	Unique identifier for logical object of which this identifier identifies one version; normally the *object_id* will be the unique identifier of the version container containing the version referred to by this OBJECT_VERSION_ID instance.
1..1	**version_tree_id**: VERSION_TREE_ID	Tree identifier of this version with respect to other versions in the same version tree, as either 1 or 3 part dot-separated numbers, e.g. "1", "2.1.4".
1..1	**creating_system_id**: UID	Identifier of the system that created the Version corresponding to this Object version id.
Functions	**Signature**	**Meaning**
	is_branch: Boolean	True if this version identifier represents a branch.
Invariants	***Object_valid***: object_id /= Void ***Version_tree_id***: version_tree_id /= Void ***creating_system_id***: creating_system_id /= Void	

4.2.7.1 Identifier Syntax

The string form of an OBJECT_VERSION_ID stored in its *value* attribute consists of three segments separated by double colons ("::"), i.e. (EBNF):

```
value:               object_id '::' creating_system_id '::' version_tree_id
object_id:           uid                                  -- see UID below
creating_system_id:
```

An example is as follows:

```
F7C5C7B7-75DB-4b39-9A1E-C0BA9BFDBDEC::87284370-2D4B-
4e3d-A3F3-F303D2F4F34B::2
```

4.2.8 VERSION_TREE_ID Class

CLASS	VERSION_TREE_ID
Purpose	Version tree identifier for one version.

CLASS	VERSION_TREE_ID	
Attributes	**Signature**	**Meaning**
1..1	**value**: `String`	String form of this identifier.
Functions	**Signature**	**Meaning**
	trunk_version: `String`	Trunk version number.
	branch_number: `String`	Number of branch from the trunk point.
	branch_version: `String`	Version of the branch.
	is_branch: `Boolean`	True if this version identifier represents a branch, i.e. has *branch_number* and *branch_version* parts.
	is_first: `Boolean`	True if this version identifier corresponds to the first version, i.e. *trunk_version* = “1”
Invariants	***Value_valid***: value /= Void **and then not** value.is_empty ***Trunk_version_valid***: trunk_version /= Void **and then** trunk_version.is_integer ***Branch_number_valid***: branch_number /= Void **implies** branch_number.is_integer ***Branch_version_valid***: branch_version /= Void **implies** branch_version.is_integer ***Branch_validity***: (branch_number = Void **and** branch_version = Void) **xor** (branch_number /= Void **and** branch_version /= Void) ***Is_branch_validity***: is_branch **xor** branch_number = Void ***Is_first_validity***: **not** is_first **xor** trunk_version.is_equal(“1”)	

4.2.8.1 Syntax

The format of the value attribute is (EBNF):

```
value:           trunk_version [ . branch_number . branch_version ]
trunk_version:   { digit }+
branch_number:   { digit }+
branch_version:  { digit }+
```

4.2.9 ARCHETYPE_ID Class

CLASS	ARCHETYPE_ID	
Purpose	Identifier for archetypes.	
Inherit	*`OBJECT_ID`*	
Functions	**Signature**	**Meaning**

CLASS	ARCHETYPE_ID	
	qualified_rm_entity: `String`	Globally qualified reference model entity, e.g. "`openehr-composition-OBSERVATION`".
	domain_concept: `String`	Name of the concept represented by this archetype, including specialisation, e.g. "`biochemistry_result-cholesterol`".
	rm_originator: `String`	Organisation originating the reference model on which this archetype is based, e.g. "`openehr`", "`cen`", "`hl7`".
	rm_name: `String`	Name of the reference model, e.g. "`rim`", "`ehr_rm`", "`en13606`".
	rm_entity: `String`	Name of the ontological level within the reference model to which this archetype is targeted, e.g. for openEHR, "folder", "composition", "section", "entry".
	specialisation: `String`	Name of specialisation of concept, if this archetype is a specialisation of another archetype, e.g. "cholesterol".
	version_id: `String`	Version of this archetype.
Invariant	***Qualified_rm_entity_valid***: qualified_rm_entity /= Void **and then not** qualified_rm_entity.is_empty ***Domain_concept_valid***: domain_concept /= Void **and then not** domain_concept.is_empty ***Rm_originator_valid***: rm_originator /= Void **and then not** rm_originator.is_empty ***Rm_name_valid***: rm_name /= Void **and then not** rm_name.is_empty ***Rm_entity_valid***: rm_entity /= Void **and then not** rm_entity.is_empty ***Specialisation_valid***: specialisation /= Void **implies not** specialisation.is_empty ***Version_id_valid***: version_id /= Void **and then not** version_id.is_empty	

4.2.9.1 Archetype ID Syntax

Archetype identifiers are "multi-axial", meaning that each identifier instance denotes a single archetype within a multi-dimensional space. In this case, the space is essentially a versioned 3-dimensional space, with the dimensions being:

- reference model entity, i.e. target of archetype
- domain concept
- version

As with any multi-axial identifier, the underlying principle of an archetype id is that all parts of the id must be able to be considered immutable. This means that no variable characteristic of an archetype (e.g. accrediting authority, which might change due to later accreditation by another authority, or may be multiple) can be included in its identifier. The syntax of an `ARCHETYPE_ID` is as follows (EBNF):

```
archetype_id: qualified_rm_entity '.' domain_concept '.' version_id

qualified_rm_entity: rm_originator '-' rm_name '-' rm_entity
rm_originator: V_NAME
rm_name: V_NAME
rm_entity: V_NAME

domain_concept: concept_name { '-' specialisation }*
concept_name: V_NAME
specialisation: V_NAME

version_id: 'v' V_NUMBER

NUMBER: [0-9]*
NAME: [a-z][a-z0-9()/%$#&]*
```

The field meanings are as follows:

rm_originator: id of organisation originating the reference model on which this archetype is based;

rm_name: id of the reference model on which the archetype is based;

rm_entity: ontological level in the reference model;

domain_concept: the domain concept name, including any specialisations;

version_id: numeric version identifier;

Examples of archetype identifiers include:

- `openehr-composition-SECTION.physical_examination.v2`
- `openehr-composition-SECTION.physical_examination-prenatal.v1`
- `hl7-rim-act.progress_note.v1`
- `openehr-composition-OBSERVATION.progress_note-naturopathy.v2`

Archetypes can also be identified by other means, such as ISO oids.

4.2.10 TERMINOLOGY_ID Class

CLASS	TERMINOLOGY_ID	
Purpose	Identifier for terminologies such accessed via a terminology query service. In this class, the value attribute identifies the Terminology in the terminology service, e.g. “SNOMED-CT”. A terminology is assumed to be in a particular language, which must be explicitly specified. The value if the id attribute is the precise terminology id identifier, including actual release (i.e. actual “version”), local modifications etc; e.g. “ICPC2”	
Inherit	*OBJECT_ID*	
Functions	Signature	Meaning

CLASS	TERMINOLOGY_ID	
	name: `String`	Return the terminology id (which includes the "version" in some cases). Distinct names correspond to distinct (i.e. non-compatible) terminologies. Thus the names "ICD10AM" and "ICD10" refer to distinct terminologies.
	version_id: `String`	Version of this terminology, if versioning supported, else the empty string.
Invariants	***Name_valid***: name /= Void **and then not** name.is_empty ***Version_id_valid***: version_id /= Void	

4.2.10.1 Identifier Syntax

The syntax of the *value* attribute is as follows:

```
name [ "(" version ")" ]
```

Examples of terminology identifiers include:

- `"snomed-ct"`
- `"ICD9(1999)"`

Versions should only be needed for those terminologies which break the rule that the thing being identified with a code loses or changes its meaning over versions of the terminology. This should not be the case for well known modern terminologies and ontologies, particularly those designed since the publication of Cimino's 'desiderata' [1] of which the principle of "concept permanance" is applicable here - "A concept's meaning cannot change and it cannot be deleted from the vocabulary". However, there maybe older terminologies, or specialised terminologies which may not have obeyed these rules, but which are still used; version ids should always be used for these.

4.2.11 GENERIC_ID Class

CLASS	GENERIC_ID	
Purpose	Generic identifier type for identifiers whose format is othterwise unknown to *open*EHR. Includes an attribute for naming the identification scheme (which may well be local).	
Inherit	*`OBJECT_ID`*	
attributes	**Signature**	**Meaning**
1..1	**scheme**: `String`	Name of the scheme to which this identifier conforms. Ideally this name will be recognisable globally but realistically it may be a local *ad hoc* scheme whose name is not controlled or standardised in any way.
Invariants	***Scheme_valid***: scheme /= Void **and then not** scheme.is_empty	

4.2.12 UID Class

CLASS	*UID (abstract)*	
Purpose	Abstract parent of classes representing unique identifiers which identify information entities in a durable way. UIDs only ever identify one IE in time or space and are never re-used.	
HL7	The HL7v3 UID Data type.	
Attributes	**Signature**	**Meaning**
	value: `String`	The value of the id.
Invariant	***Value_exists***: value /= Void ***and then not*** value.empty	

4.2.13 ISO_OID Class

CLASS	ISO_OID	
Purpose	Model of ISO's Object Identifier (oid) as defined by the standard ISO/IEC 8824 . Oids are formed from integers separated by dots. Each non-leaf node in an Oid starting from the left corresponds to an assigning authority, and identifies that authority's namespace, inside which the remaining part of the identifier is locally unique.	
HL7	The HL7v3 OID Data type.	
Inherit	`UID`	
Functions	**Signature**	**Meaning**
Invariant		

4.2.14 UUID Class

CLASS	UUID	
Purpose	Model of the DCE Universal Unique Identifier or UUID which takes the form of hexadecimal integers separated by hyphens, following the pattern 8-4-4-4-12 as defined by the Open Group, CDE 1.1 Remote Procedure Call specification, Appendix A. Also known as a GUID.	
HL7	The HL7v3 UUID Data type.	
Inherit	`UID`	
Functions	**Signature**	**Meaning**

CLASS	UUID
Invariant	

4.2.15 INTERNET_ID Class

CLASS	INTERNET_ID	
Purpose	Model of a reverse internet domain, as used to uniquely identify an internet domain. In the form of a dot-separated string in the reverse order of a domain name, specified by IETF RFC 1034 (http://www.ietf.org/rfc/rfc1034.txt).	
Inherit	UID	
Functions	Signature	Meaning
Invariant		

4.2.15.1 Syntax

According to IETF RFC1034, the syntax of a domain name follows the BNF grammar:

```
domain: subdomain | ' '
subdomain: label | subdomain '.' label
label: letter [ [ ldh-str ] let-dig ]
ldh-str: let-dig-hyp | let-dig-hyp ldh-str
let-dig-hyp: let-dig | '-'
let-dig: letter | digit

letter: any one of the 52 alphabetic characters A through Z in
upper case and a through z in lower case

digit: any one of the ten digits 0 through 9
```

It can also be expressed using the regular expression:

```
[a-zA-Z]([a-zA-Z0-9-]*[a-zA-Z0-9])?(\.[a-zA-Z]([a-zA-Z0-9-]*[a-zA-Z0-9]))*
```

5 Terminology Package

5.1 Overview

This section describes the `terminology` package, which contains classes for accessing the *open*EHR support terminology from within instances of classes defined in the reference model.

5.2 Service Interface

A simple terminology service interface is defined according to FIGURE 5, enabling *open*EHR terms to be referenced formally from within the Reference Model.

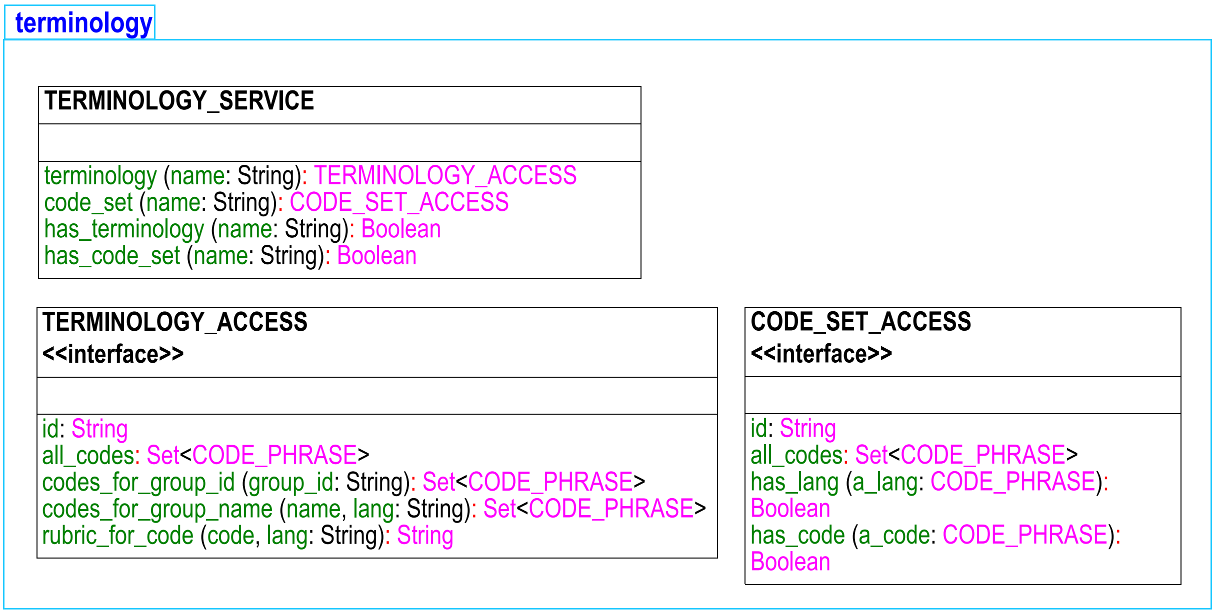

FIGURE 5 rm.support.terminology Package

Structural attributes in the Reference Model, such as `FEEDER_AUDIT`.*change_type* are defined by an invariant in the enclosing class, such as the following:

> ***Change_type_valid***: terminology("openehr").codes_for_group_name("audit change type", "en").has(change_type.defining_code)

This is a formal way of saying that the attribute *change_type* must have a value such that its *defining_code* (its `CODE_PHRASE`) is in the set of `CODE_PHRASEs` in the *open*EHR Terminology which are in the group called (in english) "audit change type".

A similar invariant is used for attributes of type `CODE_PHRASE`, which come from a code_set:

> ***Media_type_terminology***: media_type /= Void ***and then***
> code_set("media types").all_codes.has(media_type)

5.2.1 Class Definitions

5.2.1.1 TERMINOLOGY_SERVICE Class

CLASS	TERMINOLOGY_SERVICE	
Purpose	Defines an object providing proxy access to a terminology service.	
Functions	Signature	Meaning
	terminology (name: `String`): `TERMINOLOGY_ACCESS` ***require*** name /= Void ***and then*** has_terminology (name: `String`) ***ensure*** Result /= Void	Return an interface to the terminology named 'name'
	code_set (name: `String`): `CODE_SET_ACCESS` ***require*** name /= Void ***and then*** has_code_set (name: `String`) ***ensure*** Result /= Void	Return an interface to the code_set named 'name'
	has_terminology (name: `String`): `Boolean` ***require*** name /= Void ***and then*** not name.is_empty	True if terminology named 'name' known by this service.
	has_code_set (name: `String`): `Boolean` ***require*** name /= Void ***and then*** not name.is_empty	True if code_set named 'name' known by this service.
Invariants		

5.2.1.2 TERMINOLOGY_ACCESS Class

CLASS	TERMINOLOGY_ACCESS	
Purpose	Defines an object providing proxy access to a terminology.	
Functions	Signature	Meaning
	id: `String`	Identification of this Terminology

CLASS	TERMINOLOGY_ACCESS	
	all_codes: `Set<CODE_PHRASE>`	Return all codes known in this terminology
	codes_for_group_id (group_id: `String`): `Set<CODE_PHRASE>`	Return all codes under grouper 'group_id' from this terminology
	codes_for_group_name (name, lang: **String**): `Set<CODE_PHRASE>`	Return all codes under grouper whose name in 'lang' is 'name' from this terminology
	rubric_for_code (code, lang: `String`): `String`	Return all rubric of code 'code' in language 'lang'.
Invariants	***id_exists***: id /= Void ***and then not*** id.is_empty	

5.2.1.3 CODE_SET_ACCESS Class

CLASS	CODE_SET_ACCESS	
Purpose	Defines an object providing proxy access to a code_set.	
Functions	**Signature**	**Meaning**
	id: `String`	Identification of this Terminology
	all_codes: `Set<CODE_PHRASE>`	Return all codes known in this terminology
	has_lang (a_lang: `CODE_PHRASE`): `Boolean`	True if code set knows about 'a_lang'
	has_code (a_code: `CODE_PHRASE`): `Boolean`	True if code set knows about 'a_code'
Invariants	***id_exists***: id /= Void ***and then not*** id.is_empty	

6 Measurement Package

6.1 Overview

The Measurement package defines a minimum of semantics relating to quantitative measurement, units, and conversion, enabling the Quantity package of the *open*EHR Data Types Information Model to be correctly expressed. As for the Terminology package, a simple service interface is assumed, which provides useful functions to other parts of the reference model. The definitions underlying measurement and units come from a variety of sources, including:

- CEN ENV 12435, Medical Informatics - Expression of results of measurements in health sciences (see http://www.centc251.org);
- the Unified Code for Units of Measure (UCUM), developed by Gunther Schadow and Clement J. McDonald of The Regenstrief Institute (available in HL7v3 ballot materials; http://www.hl7.org).

These of course rest in turn upon a vast amount of literature and standards, mainly from ISO on the subject of scientific measurement.

6.2 Service Interface

A simple measurement data service interface is defined according to FIGURE 6, enabling quantitative semantics to be used formally from within the Reference Model. Note that this service as currently defined in no way seeks to properly model the semantics of units, conversions etc - it provides only the minimum functions required by the *open*EHR Reference Model.

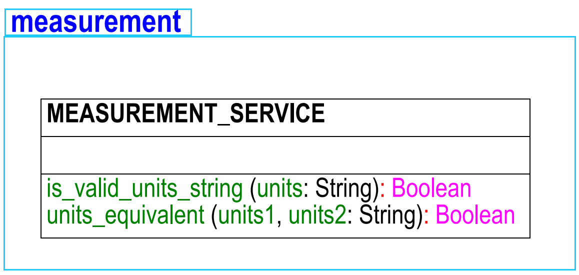

FIGURE 6 rm.support.measurement Package

6.2.1 Class Definitions

6.2.1.1 MEASUREMENT_SERVICE_ACCESS Class

CLASS	**MEASUREMENT_SERVICE**	
Purpose	Defines an object providing proxy access to a measurement information service.	
Functions	**Signature**	**Meaning**
	is_valid_units_string (units: `String`): `Boolean` ***require*** units /= Void	True if the units string 'units' is a valid string according to the HL7 UCUM specification.

CLASS	MEASUREMENT_SERVICE	
	units_equivalent (units1, units2: String): Boolean ***require*** units1 /= Void ***and then*** is_valid_units_string(units1) units2 /= Void ***and then*** is_valid_units_string(units2)	True if two units strings correspond to the same measured property.
Invariants		

7 Definition Package

7.1 Overview

The `definition` package, illustrated in FIGURE 7, describes symbolic definitions used by the *open*EHR models.

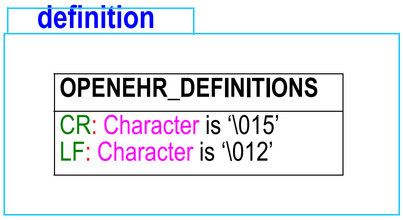

FIGURE 7 rm.support.definition Package

7.1.1 Class Definitions

7.1.1.1 OPENEHR_DEFINITIONS Class

CLASS	**OPENEHR_DEFINITIONS**	
Purpose	Defines globally used constant values.	
Attributes	**Signature**	**Meaning**
	CR: Character is '\015'	Carriage return character
	LF: Character is '\012'	Linefeed character
Invariants		

A References

A.1 General

1 Cimino J J. *Desiderata for Controlled Medical vocabularies in the Twenty-First Century*. IMIA WG6 Conference, Jacksonville, Florida, Jan 19-22, 1997.

END OF DOCUMENT