REFERENCE MODEL

The *open*EHR Common Information Model

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

Revision: 2.0rc3

Pages: 49

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.

Founding Chairman	David Ingram, Professor of Health Informatics, CHIME, University College London
Founding Members	Dr P Schloeffel, Dr S Heard, Dr D Kalra, D Lloyd, T Beale

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

Copyright Notice

Amendment Record

Issue	Details	Who	Completed
2.0rc3	**CR-000147**: Make DIRECTORY Re-usable. Add new directory package. **CR-000162**. Allow party identifiers when no demographic data. **CR-000167**. Add AUTHORED_RESOURCE class. **CR-000179**. Move AUDIT_DETAILS to generic package; add AUDIT_TRAIL	T Beale S Heard H Frankel T Beale T Beale	05 Nov 2005
RELEASE 0.96			
1.6.2	**CR-000159**. Improve explanation of use of ATTESTATION in change_control package.	T Beale	10 Jun 2005
RELEASE 0.95			
1.6.1	**CR-000048**. Pre-release review of documents. Fixed UML in Fig 8 informal model of version control.	D Lloyd	22 Feb 2005
1.6	**CR-000108**. Minor changes to change_control package. **CR-000024**. Revert *meaning* to STRING and rename as *archetype_node_id*. **CR-000097**. Correct errors in version diagrams in Common model. **CR-000099**. PARTICIPATION.*function* type in diagram not in sync with spec. **CR-000116**. Add PARTICIPATION.*function* vocabulary and invariant. **CR-000118**. Make package names lower case. Improve presentation of identification section; move some text to data types IM document, basic package. **CR-000111**. Move Identification Package to Support	T Beale S Heard T Beale Ken Thompson R Shackel (DSTC) T Beale T Beale DSTC	10 Dec 2004
RELEASE 0.9			
1.5	**CR-000080**. Remove ARCHETYPED.*concept* - not needed in data **CR-000081**. LINK should be unidirectional. **CR-000083**. RELATED_PARTY.*party* should be optional. **CR-000085**. LOCATABLE.*synthesised* not needed. Add vocabulary for FEEDER_AUDIT.*change_type*. **CR-000086**. LOCATABLE.*presentation* not needed. **CR-000091**. Correct anomalies in use of CODE_PHRASE and DV_CODED_TEXT. Changed PARTICIPATION.*mode*, changed ATTESTATION.*status*, RELATED_PARTY.*relationship*, VERSION_AUDIT.*change_type*, FEEDER_AUDIT.*change_type* to to DV_CODED_TEXT. **CR-000094**. Add lifecycle state attribute to VERSION; correct DV_STATE. **Formally validated using ISE Eiffel 5.4.**	DSTC T Beale, S Heard DSTC	09 Mar 2004

Issue	Details	Who	Completed
1.4.12	**CR-000071**. Allow version ids to be optional in TERMINOLOGY_ID. **CR-000044**. Add reverse ref from VERSION_REPOSITORY<T> to owner object. **CR-000063**. ATTESTATION should have a status attribute. **CR-000046**. Rename COORDINATED_TERM and DV_CODED_TEXT.*definition*.	T Beale D Lloyd D Kalra T Beale	25 Feb 2004
1.4.11	**CR-000056**. References in COMMON.Version classes should be OBJECT_REFs.	T Beale	02 Nov 2003
1.4.10	**CR-000045**. Remove VERSION_REPOSITORY.*status*	D Lloyd, T Beale	21 Oct 2003
1.4.9	**CR-000025**. Allow ATTESTATIONs to attest parts of COMPOSITIONs. Change made due to CEN TC/251 joint WGM, Rome, Feb 2003. **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.)	D Kalra, D Lloyd, T Beale	09 Oct 2003
1.4.8	**CR-000041**. Visually differentiate primitive types in openEHR documents.	D Lloyd	04 Oct 2003
1.4.7	**CR-000013**. Rename key classes according to CEN ENV13606.	S Heard, D Kalra, T Beale	15 Sep 2003
1.4.6	**CR-000012**. Add *presentation* attribute to LOCATABLE. **CR-000027**. Move *feeder_audit* to LOCATABLE to be compatible with CEN 13606 revision. Add new class FEEDER_AUDIT.	D Kalra	20 Jun 2003
1.4.5	**CR-000020**. Move VERSION.*charset* to DV_TEXT, *territory* to TRANSACTION. Remove VERSION.*language*.	A Goodchild	10 Jun 2003
1.4.4	**CR-000007**. Add RELATED_PARTY class to GENERIC package. **CR-000017**. Renamed VERSION.*parent_version_id* to *preceding_version_id*.	S Heard, D Kalra	11 Apr 2003
1.4.3	**Major alterations** due to **CR-000003**, **CR-000004**. ARCHETYPED class no longer inherits from LOCATABLE, now related by association. Redesign of Change Control package. Document structure improved. (Formally validated)	T Beale, Z Tun	18 Mar 2003
1.4.2	Moved External package to Support RM. Corrected CONTRIBUTION.*description* to DV_TEXT. Made PARTICIPATION.*time* optional. (Formally validated).	T Beale	25 Feb 2003
1.4.1	**Formally validated using ISE Eiffel 5.2.** Corrected types of VERSIONABLE.*language*, *charset*, *territory*. Added ARCHETYPED.*uid*:OBJECT_ID. Renamed ARCHETYPE_ID.*rm_source* to *rm_originator*, and *rm_level* to *rm_concept*; added *archetype_originator*. Rewrote archetype id section. Changed PARTICIPATION.*mode* to COORDINATED_TERM & fixed invariant.	T Beale, D Kalra	18 Feb 2003

Issue	Details	Who	Completed
1.4	**Changes post CEN WG meeting Rome Feb 2003.** Changed ARCHETYPED.*meaning* from STRING to DV_TEXT. Added CONTRIBUTION.*name* invariant. Removed AUTHORED_VA and ACQUIRED_VA audit types, moved feeder audit to the EHR RM. VERSIONABLE.*code_set* renamed to *charset*. Fixed pre/post condition of OBJECT_ID.*context_id*, added OBJECT_ID.*has_context_id*. Changed TERMINOLOGY_ID string syntax.	T Beale, D Kalra, D Lloyd	8 Feb 2003
1.3.5	Removed segment from archetype_id; corrected inconsistencies in diagrams and class texts.	Z Tun, T Beale	3 Jan 2003
1.3.4	Removed inheritance from VERSIONABLE to ARCHETYPED.	T Beale	3 Jan 2003
1.3.3	Minor corrections: OBJECT_ID; changed syntax of TERMINOLOGY_ID. Corrected Fig 6.	T Beale	17 Nov 2002
1.3.2	Added Generic Package; added PARTICIPATION and changed and moved ATTESTATION class.	T Beale	8 Nov 2002
1.3.1	Removed EXTERNAL_ID.*iso_oid*. Remodelled EXTERNAL_ID into new classes - OBJECT_REF and OBJECT_ID. Remodelled all change control classes.	T Beale, D Lloyd, M Darlison, A Goodchild	22 Oct 2002
1.3	Moved ARCHETYPE_ID.*iso_oid* to EXTERNAL_ID. DV_LINK no longer a data type; renamed to LINK.	T Beale	22 Oct 2002
1.2	Removed Structure package to own document. Improved CM diagrams.	T Beale	11 Oct 2002
1.1	Removed HCA_ID. Included Spatial package from EHR RM. Renamed SPATIAL to STRUCTURE.	T Beale	16 Sep 2002
1.0	Taken from EHR RM.	T Beale	26 Aug 2002

Acknowledgements

The work reported in this paper has been funded in 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 describes the architecture of the *open*EHR Common Reference Model, which contains concepts used by other *open*EHR reference models.

The intended audience includes:

- Standards bodies producing health informatics standards;
- Software development groups using *open*EHR;
- Academic groups using *open*EHR;
- The open source healthcare community;
- Medical informaticians and clinicians intersted in health information;
- Health data managers.

1.2 Related Documents

Prerequisite documents for reading this document include:

- The *open*EHR Modelling Guide
- The *open*EHR Support Information Model
- The *open*EHR Data Types Information Model
- The *open*EHR Data Structures Information Model

1.3 Status

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

The latest version of this document can be found in PDF format at http://svn.openehr.org/specification/BRANCHES/Release-1.0-candidate/publishing/architecture/rm/common_im.pdf. New versions are announced on openehr-announce@openehr.org.

Blue text indicates sections under active development.

NOTE THAT NOT ALL CHANGES MADE TO THIS DOCUMENT HAVE BEEN APPROVED BY THE OPENEHR ARB, AND MAY BE SUBJECT TO FURTHER CHANGE.

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 Overview

The `common` Reference Model comprises a number of packages containing concepts used in higher level *open*EHR models. It is illustrated in FIGURE 1.

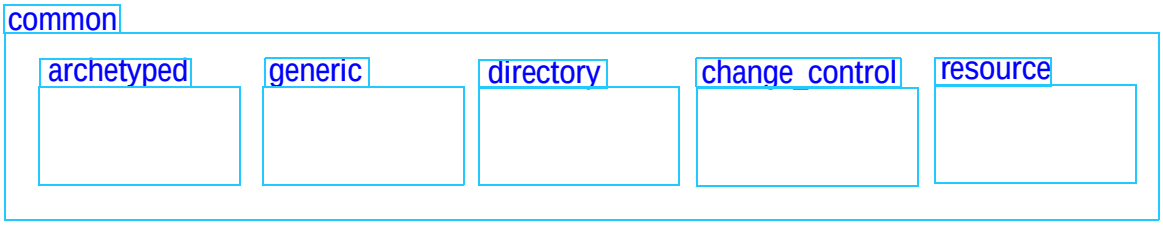

FIGURE 1 rm.common Package

The `archetyped` package described here is informed by a number of design principles, centred on the concept of "two-level" modelling. These principles are described in detail [1], and discussed with respect to the EHR [2].

The `generic` package contains classes representing concepts which are generic across the domain, mostly to do with referencing demographic entities from within other data including Participation, Proxy_party, Attestation and so on.

The `directory` package provides a simple reusable abstraction of a versioned folder structure.

The `change_control` package defines the generalised semantics of changes to a repository, such as an EHR, over time. Each item in such a repository is version controlled to allow the repository as a whole to be properly versioned in time. The semantics described are in response to medico-legal requirements defined in GEHR [9], and in the ISO Technical Specification 18308 [4]. Both of these requirements specifications mention specifically the version control of the health record.

The `resource` package defines semantics of the general idea of an online authored resource, such as a document, and supports multiple language translations, descriptive meta-data and revision history.

3 Archetyped Package

3.1 Overview

The `archetyped` package includes the core types `LOCATABLE`, `ARCHETYPED`, and `LINK`. It is illustrated in FIGURE 2.

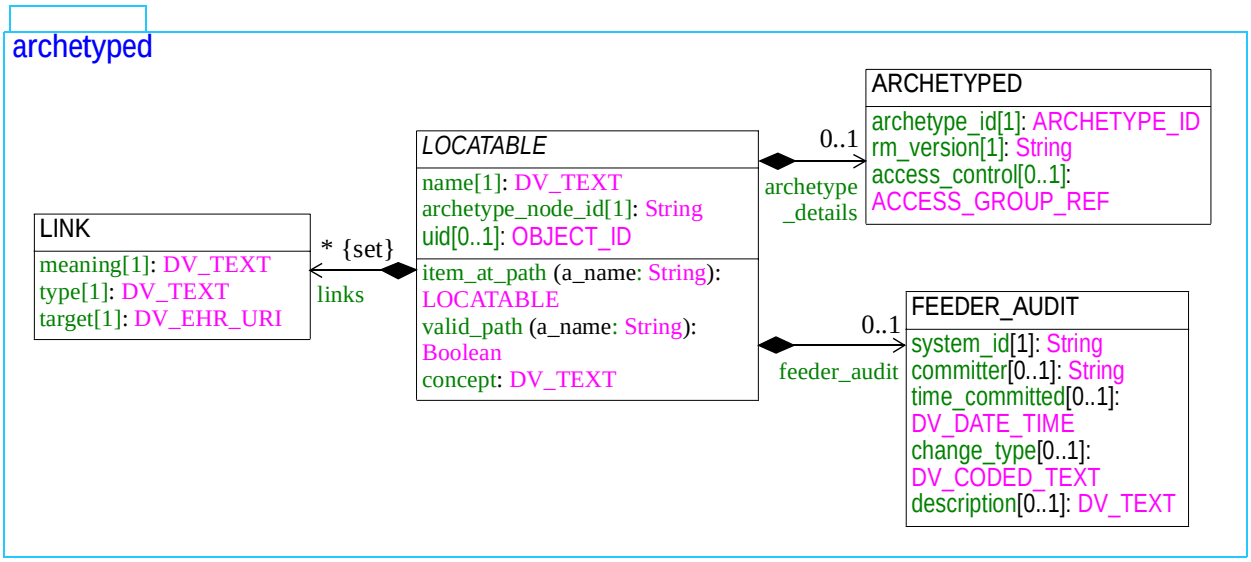

FIGURE 2 rm.common.archetyped Package

3.1.1 The Root Class LOCATABLE

Every structural element in the EHR model inherits from the `LOCATABLE` class, ensuring it has both a runtime *name*, and a meaning, derivable from the *archetype_node_id*. The *archetype_node_id* is the standardised semantic code for a node, while the *name* attribute carries the runtime name. The name and archetype_node_id values are often the same semantically, but may differ. For example, in "problem/SOAP" Sections (i.e. headings), the name of a section at the problem level might be "diabetes", but its meaning (obtained using the *archetype_node_id* value from the local ontology in its generating archetype) will be "problem". The default value for *names* should be assumed to be the text value in the local language for the *archetype_node_id* code on the node in question, unless explicitly set otherwise. `LOCATABLE` also provides the attribute *archetype_details*, which is non-Void for archetype root points in data. The "meaning" of any node is derived formally from the archetype by obtaining the "text" value for the *archetype_node_id* code from the archetype ontology, in the language required. In the *open*EHR documents, this operation is denoted by the following expression:

```
meaning(XXX.archetype_node_id)
```

Where "XXX" stands for some type, such as `SECTION` or `ENTRY`, which inherits from `LOCATABLE`. The concrete way of doing this operation is described in the openEHR Archetype Object Model (AOM) document.

`LOCATABLE` objects may also have a *uid*, typically implemented using an ISO Oid. In a given data composition, only those nodes which correspond to archetype root points should have a uid, since reliable paths can be generated to any point within the tree from a given root point. Thus, root points which might contain uids would normally be Compositions, Sections which are the root of a Section

tree, and Entry objects; they could be finer grained nodes inside Entries if finer grained archetypes are used.

Currently the model does not formally mandate uids to be used, or to be used on any particular kind of node, despite the statements above, because there is not enough documented experience with using Oids for data node identification (particularly the computational costs of dereferencing, and the storage costs in otherwise 'small' data). More experience with real *open*EHR deployments is required before the correct formal semantics can be specified.

3.1.2 Feeder Systems

The data in any part of the EHR may be obtained from a feeder system, i.e. a source system which does not obey any of the EHR semantics defined by *open*EHR. In particular, there is no guarantee that the granularity of information recorded in the feeder system obeys the rules of Entries, Compositions, etc. As a consquence, feeder information might correspond to any level of information defined in the *open*EHR models. In order to be able to record feeder audit information correctly, the model has to be able to associate an audit trail with any granularity of object. For this reason, feeder audit information is attached to the `LOCATABLE` class via the *feeder_audit* attribute, even though it is preferable by design to have it attached to the equivalent of Compositions or at least the equivalent of archetype entities (i.e. Compositions, Section trees and Entries). Its usual usage is to attach it to the outermost object to which it applies. In other words, in most cases, during a legacy data conversion process, the entirety of a Composition needs only one `FEEDER_AUDIT` to document its origins. In exceptional cases, where feeder data comes in in near real time, e.g. from an ICU database, separate `FEEDER_AUDIT` objects may need to be generated for parts of a Composition; each commit in this situation will create a stack of versions of one Composition, with a growing number of `FEEDER_AUDIT` objects attached to internal data nodes, each documenting the last import of data.

The feeder audit information is included as part of the data of the Composition, rather than part of the audit trail of version committal, because it remains revelant throughout the versioning of a logical Composition, i.e. when a new version is created, the feeder information is retained as part of the current version to be seen and possibly modified, just as for the rest of its content. If the main part of the content is modified so drastically as to make the feeder audit irrelevant, it too can be removed.

A second consequence of feeder and legacy systems is that structural data items may need to be synthesised in order to create valid structures, even though the source system does not have them. For example, a system may have the equivalent data of Entries, but no Sections or other higher-level data items; these have to be synthesised during conversion. To indicate synthesis of a data node, a `FEEDER_AUDIT` instance is attached to the `LOCATABLE` in question, and its *change_type* set to "synthesised".

3.2 Class Descriptions

3.2.1 Class LOCATABLE

CLASS	*LOCATABLE (abstract)*
Purpose	Root structural class of all information models.
GEHR	*Name* attribute in `ARCHETYPED`, *meaning* attribute in `G1_PLAIN_TEXT`.

CLASS	*LOCATABLE (abstract)*	
Synapses	Each record component includes a Synapses Object ID attribute to reference the Synapses Object (archetype) used as the basis for its construction. All record components include a name attribute intended for the same purpose as the *open*EHR equivalent.	
Attributes	**Signature**	**Meaning**
	uid: `OBJECT_ID`	Optional globally unique object identifier for root object of archetyped data structure.
	archetype_node_id: `String`	Design-time archetype id of this node taken from its generating archetype; used to build archetype paths. Always in the form of an “at” code, e.g. “at0005”. This value enables a "standardised" name for this node to be generated, by referring to the generating archetype local ontology.
	name: `DV_TEXT`	Runtime name of this fragment, used to build runtime paths. This is the term provided via a clinical application or batch process to name this EHR construct: its retention in the EHR faithfully preserves the original label by which this entry was known to end users.
	archetype_details: `ARCHETYPED`	Details of archetyping used on this node.
	feeder_audit: `FEEDER_AUDIT`	Audit trail from non-*open*EHR system of original commit of information forming the content of this node, or from a conversion gateway which has synthesised this node.
	links: `Set <LINK>`	Links to other archetyped structures (data whose root object inherits from `ARCHETYPED`, such as `ENTRY`, `SECTION` and so on). Links may be to structures in other compositions.
Functions	**Signature**	**Meaning**
	is_archetype_root: `Boolean`	True if this node is the root of an archetyped structure.
	path_of_item (a_loc: `LOCATABLE`): `String`	The path to an item relative to the root of this archetyped structure.
	item_at_path (a_path: `String`): `LOCATABLE`	The item at a path (relative to this item).

CLASS	*LOCATABLE (abstract)*	
	valid_path (a_path: `String`): `Boolean`	True if the path is valid with respect to the current item.
	concept: `DV_TEXT` ***require*** is_archetype_root	Clinical concept of the archetype as a whole (= derived from the 'archetype_node_id' of the root node)
Invariant	***Archetype_node_id_valid***: archetype_node_id /= Void ***Name_exists***: name /= Void ***Links_valid***: links /= Void ***implies not*** links.empty ***Archetyped_validity***: is_archetype_root ***xor*** archetype_details = Void	

3.2.2 ARCHETYPED Class

CLASS	ARCHETYPED	
Purpose	Archetypes act as the configuration basis for the particular structures of instances defined by the reference model. To enable archetypes to be used to create valid data, key classes in the reference model act as "root" points for archetyping; accordingly, these classes have the archetype_details attribute set. An instance of the class `ARCHETYPED` contains the relevant archetype identification information, allowing generating archetypes to be matched up with data instances	
GEHR	`G1_ARCHETYPED`	
Synapses/ SynEx	The SynEx approach does not distinguish between multiple layers of archetypes; hence an 'archetype' covers all information in an entire composition. Consequently, there is only one place where archetype identifiers in the *open*EHR sense are used (at the top); all other archetype identifiers are equivalent to the *archetype_node_id* attribute from `LOCATABLE`. The Synapses ObjectID attribute provides a unique reference to each fine-grained element of an archetype, and is therefore also functionally equivalent to the *archetype_id* attribute at the root points in an *open*EHR structure.	
CEN	The 1999 pre-standard does not include any equivalent to the archetype concept. However each architectural component must include a reference to an entry in the relevant normative table in the Domain Termlist pre-standard (part 2), to provide a high-level semantic classification of the component. All Architectural components include a component name structure to specify its label: the source of possible values for such a label was not clearly defined. The 2003 revision of ENV 13606 explicitly includes archetype identification attributes in the class `RECORD_COMPONENT`.	
Attributes	**Signature**	**Meaning**
	archetype_id: `ARCHETYPE_ID`	Globally unique archetype identifier.

CLASS	ARCHETYPED	
	access_control: ACCESS_GROUP_REF	The access control settings of this component.
	rm_version: String	Version of the *open*EHR reference model used to create this object.
Invariant	***archetype_id_exists***: archetype_id /= Void ***rm_version_exists***: rm_version /= Void ***and then not*** rm_version.empty	

3.2.3 LINK Class

CLASS	LINK	
Purpose	The LINK type defines a logical relationship between two items, such as two ENTRYs or an ENTRY and a COMPOSITION. Links can be used across compositions, and across EHRs. Links can potentially be used between interior (i.e. non archetype root) nodes, although this probably should be prevented in archetypes. Multiple LINKs can be attached to the root object of any archetyped structure to give the effect of a 1->N link	
Use	1:1 and 1:N relationships between archetyped content elements (e.g. ENTRYs) can be expressed by using one, or more than one, respectively, DV_LINKs. Chains of links can be used to see “problem threads” or other logical groupings of items.	
MisUse	Links should be between archetyped objects only, i.e. between objects representing complete domain concepts because relationships between sub-elements of whole concepts are not necessarily meaningful, and may be downright confusing. Sensible links only exist between whole ENTRYs, SECTIONs, COMPOSITIONs and so on.	
CEN	The Link Item class is a simplified form of the Synapses Link Item, permitting links to be established but with limited labelling and no representation for importance.	
Synapses	The Link Item class provides the means to link any arbitrary parts of a single EHR, for the overall linkage network to be labelled and revised, and for each direct link to be labelled explicitly. An importance attribute provides guidance on how links should be handled if only part of a linkage network is requested by a client process.	
GEHR	n/a	
HL7v3	The ACT_RELATIONSHIP class in some cases appears to correspond to LINK.	
Attributes	**Signature**	**Meaning**

CLASS	LINK	
	meaning: DV_TEXT	Used to describe the relationship, usually in clinical terms, such as “in response to” (the relationship between test results and an order), “follow-up to” and so on. Such relationships can represent any clinically meaningful connection between pieces of information. Values for *meaning* include those described in Annex C, ENV 13606 pt 2 [11] under the categories of “generic”, “documenting and reporting”, “organisational”, “clinical”, “circumstancial”, and “view management”.
	type: DV_TEXT	The *type* attribute is used to indicate a clinical or domain-level meaning for the kind of link, for example “problem” or “issue”. If type values are designed appropriately, they can be used by the requestor of EHR extracts to categorise links which must be followed and which can be broken when the extract is created.
	target: DV_EHR_URI	the logical “to” object in the link relation, as per the linguistic sense of the *meaning* attribute.
Functions	**Signature**	**Meaning**
Invariant	***meaning_exists***: meaning /= Void ***type_exists***: type /= Void ***target_exists***: target /= Void	

3.2.4 FEEDER_AUDIT Class

CLASS	FEEDER_AUDIT	
Purpose	Audit details for a feeder system.	
Attributes	**Signature**	**Meaning**
	system_id: `String`	Identity of the system where the item was originally committed.
	committer: `String`	Identity of party who committed the item.
	time_committed: `DV_DATE_TIME`	Time of committal of the item.
	change_type: `DV_CODED_TEXT`	Type of change, e.g. creation, correction, modification, synthesis etc. Coded using the *open*EHR Terminology "audit change type" group.
	description: `DV_TEXT`	Description of change from original system.
Invariants	***System_id_exists***: system_id /= Void ***and then not*** system_id.empty ***Committer_valid***: committer /= Void ***implies not*** committer.empty ***Change_type_valid***: change_type /= Void ***and then*** terminology("openehr").codes_for_group_name("audit change type", "en").has(change_type.defining_code)	

4 Generic Package

4.1 Overview

The classes presented in this section are abstractions of concepts which are generic to the domain of health (and most likely other domains), such as ‘participation’ and ‘attestation’. Here, “generic” means that the same model can be used, regardless of where they are contextually used in other models. The generic cluster is illustrated in FIGURE 3.

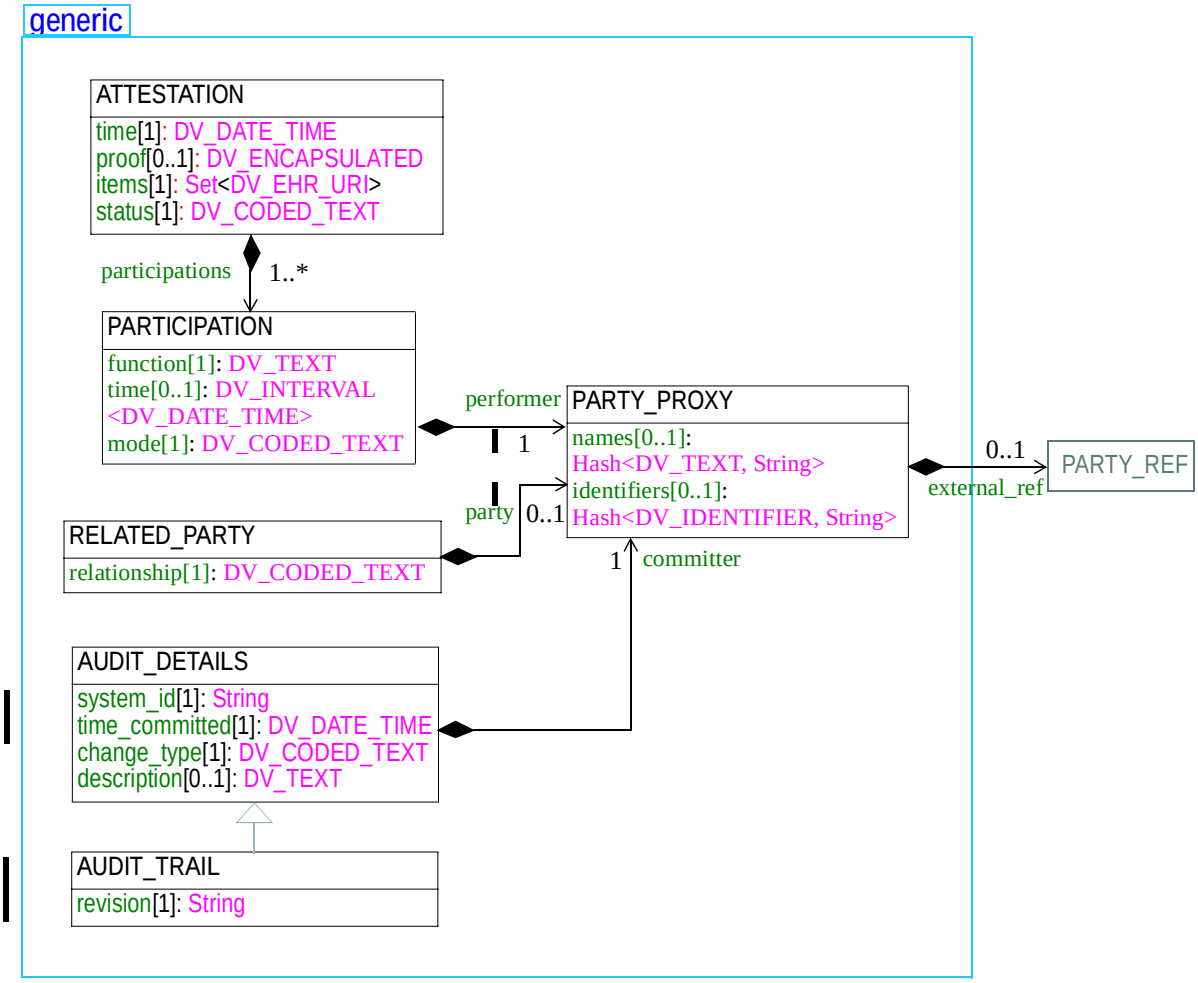

FIGURE 3 rm.common.generic Package

4.2 Design Approach

4.2.1 Referring to Demographic Data

The approach taken here for representing demographic and user entities other than the subject of the EHR in the EHR data is based on the following assumptions:

- there is at least one human readable name or official identifier of the party, such as “Julius Marlowe, MD”, “NHS provider number 1039385”, or a system user id such as “Rahil.Azam”;

- there might be data in a service external to the EHR for the party in question, such as a demographic, identity management or patient index service; if there is, we want to reference it;

The PARTY_PROXY class models references to parties based on these assumptions. It provides a flexible approach: in stricter environments that have identity management and demographic services, and where there is an entry in such a service for the party in question, PARTY_PROXY.*external_ref* will be non-Void, while in other environments, it will be empty.

Due to this flexibility, PARTY_PROXY is the preferred way to interface to identity or demographic information from within the *open*EHR EHR, with one exception: when referring to the patient, i.e. the subject of the EHR. The subject is referred to in only one place in the *open*EHR EHR, namely ENTRY.*subject*, which is of type RELATED_PARTY. To prevent the use of subject identifiers in the EHR, ENTRY.*subject* is Void when the subject of the Entry is the subject of the record (i.e. the patient) - which is most of the time. Consequently, RELATED_PARTY only ever refers to other parties, such as parents, donors, partners etc.

4.2.2 Participation

The concept of Participation is an abstraction of *the interaction between some Party and an activity.* In the *open*EHR reference models, participations are actually modelled in two ways. In situations where the kinds of participation are known and constant, they are modelled as a named attribute in the relevant reference model. For example, the *committer*: PARTY_PROXY attribute in AUDIT_DETAILS models a participation in which the function is "committal". Where the kind of participation is not known at design time, a generic PARTICIPATION class is used. This class refers to a Party via a PARTY_PROXY instsance, and records the function, time interval and (coded) mode of the participation. It can be used in any other reference model as required.

4.2.3 Attestation

Attestation is another concept which occurs commonly in health information. Here it is modelled as the one or more participations, the time of attestation, a proof object and the list of identifiers of the attested items. At a minimum, the proof should be a digital certificate which binds:

- the identity of the parties;
- the thing attested to, e.g. a statement like "Do you agree that the form below is an accurate record of the clinical session just completed?", and potentially a hash or other compressed, encoded representation of the attested-to content;
- the time;
- appropriate digital signatures.

Such a certificate may be included in the record, or it may exist in some other place such as a notary service or similar. The use of the DV_ENCAPSULATED type for the *proof* attribute allows for either.

Normally the list of items should be a single Entry or Composition, but there is nothing stopping it including fine-grained items, even though separate attestation of such items does not appear to be commensurate with good clinical information design or process.

The *status* attribute is used to indicate whether the attestation has occurred, or is planned - the latter may be optional or mandatory. It is coded using the *open*EHR Terminology group "attestation status".

4.2.4 Audit Information

Two classes are provided to represent audit information. The first, AUDIT_DETAILS expresses the details that would be captured about a user when committing some information to a repository of some kind, which may be version controlled. The second class, AUDIT_TRAIL is the same thing, which revsion information from a version control system included. The purposes of these classes is slightly different: the first is designed to represent just the audit information due to the commit action, and could be used to do so, inside a model of versioning, as it is in the change_control package. The second class is designed to represent the information that corresponds to an item in a revsion history, i.e. a list of all audits relating to some information item. The revision is therefore included to indicate which version/revision each audit corresponds to.

4.3 Class Descriptions

4.3.1 ATTESTATION Class

CLASS	ATTESTATION	
Purpose	Record of one or more Parties attesting something.	
Attributes	**Signature**	**Meaning**
	participations: List <PARTICIPATION>	Pariticipations in this attestation, e.g. witness, primary authority etc.
	time: DV_DATE_TIME	Time at which attestation was made.
	proof: DV_ENCAPSULATED	Proof of attestation.
	items: Set <DV_EHR_URI>	Items attested. Although not recommended, these may include fine-grained items which have been attested in some other system. Otherwise it is assumed to be for the entire VERSION with which it is associated.
	status: DV_CODED_TEXT	Status of this attestation. Coded by the *open*EHR Terminology group “attestation status”.
Invariants	***Participations_validity***: participations /= Void ***and then not*** participations.empty ***Time_exists***: time /= Void ***Items_validity***: items /= Void ***and then not*** items.is_empty ***Status_validity***: status /= Void ***and then*** terminology(“openehr”).codes_for_group_name(“attestation status”, “en”).has(status.defining_code)	

4.3.2 PARTICIPATION Class

CLASS	PARTICIPATION	
Purpose	Model of a participation of a Party (any Actor or Role) in an activity.	
Use	Used to represent any participation of a Party in some activity, which is not explicitly in the model, e.g. assisting nurse. Can be used to record past or future participations.	
Misuse	Should not be used in place of more permanent relationships between demographic entities.	
HL7v3	RIM Participation class.	
Attributes	**Signature**	**Meaning**
	performer: `PARTY_PROXY`	The id and possibly demographic system link of the party participating in the activity.
	function: `DV_TEXT`	The function of the Party in this participation (note that a given party might participate in more than one way in a particular activity). This attribute should be coded, but cannot be limited to the HL7v3:ParticipationFunction vocabulary, since it is too limited and hospital-oriented.
	mode: `DV_CODED_TEXT`	The mode of the performer / activity interaction, e.g. present, by telephone, by email etc.
	time: `DV_INTERVAL <DV_DATE_TIME>`	The time interval during which the participation took place, if it is used in an observational context (i.e. recording facts about the past); or the intended time interval of the participation when used in future contexts, such as EHR Instructions.
Invariant	***Function_valid***: function /= Void ***and then*** function.type.is_equal(“DV_CODED_TEXT”) ***implies*** terminology(“openehr”).codes_for_group_name(“participation function”, “en”) .has(function.defining_code) ***Mode_valid***: terminology(“openehr”).codes_for_group_name(“participation modes”, “en”).has(mode.defining_code) ***Performer_exists***: performer /= Void	

4.3.3 RELATED_PARTY Class

CLASS	RELATED_PARTY	
Purpose	Party and relationship of the party to the subject of the record.	
Attributes	**Signature**	**Meaning**

CLASS	RELATED_PARTY	
	party: PARTY_PROXY	Id and optional demographic system link for this Party
	relationship: DV_CODED_TEXT	Relationship of subject of this ENTRY to the subject of the record. May be coded. If it is the patient, coded as "self".
Invariants	***Relationship_valid***: relationship /= Void ***and then*** terminology("openehr").codes_for_group_name("related party relationship", "en") .has(relationship.defining_code)	

4.3.4 PARTY_PROXY Class

CLASS	PARTY_PROXY	
Purpose	Proxy data for a party, minimally consisting of human-readable identifier(s), such as name, formal (and possibly computable) identifiers such as NHS number, and an optional link to data for this party in a demographic system.	
Use	Used to describe parties where only identifiers may be known, and there is no entry at all in the demographic system (or even no demographic system. Typically for health care providers, e.g. name and provider number of an institution.	
Misuse	Should not be used to include patient identifying information.	
Attributes	**Signature**	**Meaning**
	names: Hash<String, DV_TEXT>	One or more human-readable names (in String form), keyed by meaning (potentially coded in the future using DV_CODED_TEXT).
	identifiers: Hash<DV_IDENTIFIER, DV_TEXT>	One or more formal identifiers (possibly computable), keyed by meaning (potentially coded in the future using DV_CODED_TEXT).
	external_ref: PARTY_REF	Optional reference to more detailed demographic or identification information for this party, in an external system.
Invariant	***Validity***: names /= Void ***or*** identifiers /= Void ***Names_validity***: names /= Void ***implies not*** names.is_empty ***Identifiers_validity***: identifiers /= Void ***implies not*** identifiers.is_empty	

4.3.5 AUDIT_DETAILS Class

CLASS	AUDIT_DETAILS	
Purpose	The set of attributes required to document the committal of an information item to a repository.	
Synapses	Composition class	
GEHR	`G1_COMMIT_AUDIT`	
Attributes	**Signature**	**Meaning**
	system_id: `String`	Identity of the system where the change was committed.
	committer: `PARTY_PROXY`	Identity and optional reference into identity management service, of user who committed the item.
	time_committed: `DV_DATE_TIME`	Time of committal of the item.
	change_type: `DV_CODED_TEXT`	Type of change. Coded using the *open*EHR Terminology “audit change type” group.
	description: `DV_TEXT`	Reason for committal.
Invariants	***System_id_exists***: system_id /= Void ***and then not*** system_id.empty ***Committer_exists***: committer /= Void ***Time_committed_exists***: time_committed /= Void ***Change_type_exists***: change_type /= Void ***and then*** terminology(“openehr”).codes_for_group_name(“audit change type”, “en”).has(change_type.defining_code)	

4.3.6 AUDIT_TRAIL Class

CLASS	AUDIT_TRAIL	
Purpose	An entry in a revision history, comprised of `AUDIT_DETAILS` instances with revision identifier of the revision to which the `AUDIT_DETAILS` intance belongs.	
Inherit	`AUDIT_DETAILS`	
Attributes	**Signature**	**Meaning**
	revision: `String`	Revision identifier of this audit trail.
Invariants	***Revsion_exists***: revision /= Void ***and then not*** revision.empty	

5 Directory Package

5.1 Overview

The `directory` package is illustrated in FIGURE 4. It provides a simple abstraction of a versioned folder structure. The `DIRECTORY` class is the binding of `VERSION_REPOSITORY<T>` to `FOLDER`, i.e. it is a `VERSION_REPOSITORY<FOLDER>`. This means that each of its versions is a `FOLDER` structure. It provides a means of versioning `FOLDER` structures over time, which is useful in the EHR, Demographics service or anywhere else where `FOLDER`s are used to group things. A `FOLDER` instance is simple: it contains more `FOLDER`s and/or *items*, which are references to other (usually versioned) objects. A `FOLDER` structure is therefore like a directory containing references to objects. Since they are only references, multiple references to the same object are possible, allowing the structure to be used to mutiply classify other objects. If it is used with `VERSIONED_COMPOSITION` for example, the folders might be used to represent episodes and at the same time problem groups.

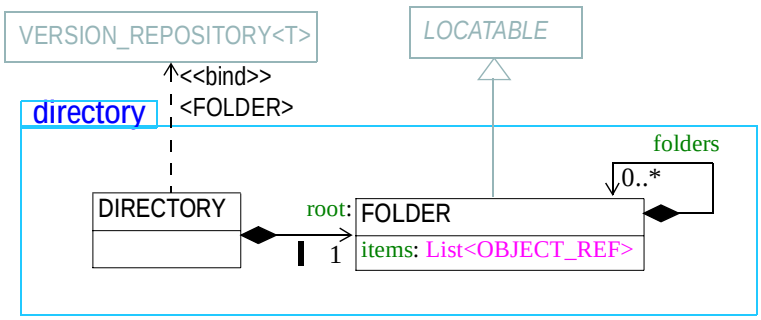

FIGURE 4 common.directory Package

`FOLDER` structures inside the `DIRECTORY` are archetypable structures, and `FOLDER` archetypes can be created in the same fashion as say `SECTION` archteypes for the EHR.

5.2 Class Descriptions

5.2.1 DIRECTORY Class

CLASS	DIRECTORY	
Purpose	A version-controlled hierarchy of `FOLDER`s.	
Inherit	`VERSION_REPOSITORY <FOLDER>`	
Attributes	**Signature**	**Meaning**
	root: `FOLDER`	Root `FOLDER` of the directory.
Invariants	***Root_exists***: root /= Void	

5.2.2 FOLDER Class

CLASS	FOLDER	
Purpose	The concept of a named folder.	
CEN	FOLDER class	
Synapses	RecordFolder class	
Inherit	LOCATABLE	
Attributes	**Signature**	**Meaning**
	folders: List<FOLDER>	Sub-folders of this FOLDER.
	items: List<OBJECT_REF>	The list of references to other (usually) versioned objects logically in this folder.
Invariants	***Folders_valid***: folders /= Void ***implies not*** folders.empty	

5.2.2.1 Folder Paths

Folder paths are built using the *name* attribute values inherited from LOCATABLE. In real data, these will usually be derived from the value of the *archetype_node_id* attribute, plus a uniqueness modifier if required. The syntax is as follows:

```
‘[’ FOLDER.name.value “]/folders[” FOLDER.name.value “]/” ...
‘/folders[” FOLDER.name.value “]/” ...
```

Example folder paths:

```
[hospital episodes]
[patient entered data]/folders[diabetes monitoring]
[homeopathy contacts]
```

Uniqueness modifiers are appended in parentheses, and only needed to differentiate folders at the same node that would otherwise have the same names, e.g.

```
[hospital episodes]
[hospital episodes(car accident Aug 1998)]
```

6 Change Control Package

6.1 Overview

In various *open*EHR reference models, the semantics of formal change control are required. There are two architectural aspects of managing changes to data. The first is the concept of a complex information object, being versioned in time, meaning that its creation and all subsequent modifications cause new "versions" to be created, rather than literally overwriting the existing data. Each version includes an audit trail, typically containing the identity of a user, the date/time of the change, and a reason for the change. The second aspect recognises that repositories are made up of complex information objects, and that changes are not in fact just made to individual objects, but to the respository itself. Any change by a user may in fact change more than one versioned object in the repository, and the set of such changes constitutes the logical unit of change to the repository, taking it from one valid state to the next.

These concepts are well-known, under the title "configuration management", and are used as the basis for most software and other change management systems, including numerous products on the market today.

The following sections describe the configuration management paradigm in more detail, and explain how it relates to the *open*EHR reference models, in particular, the model for the EHR.

6.2 The Configuration Management Paradigm

The "configuration management" (CM) paradigm is well-known in software engineering, and has its own IEEE standards. CM is about managed control of changes to a repository of items (formally called "configuration items" or CIs), and is relevant to any logical repository of distinct information items which changes in time. In health information systems, at least two types of information require such management: electronic health records, and demographic information. In most analyses in the past, the need for change management has been expressed in terms of specific requirements for audit trailing of changes, availability of previous states of the repository and so on. Here, we aim to provide a formal, general-purpose model for change control, and show how it applies to health information.

6.2.1 Organisation of the Repository

The general organisation of a repository of complex information items such as a software repository, or the EHR consists of the following:

- a number of distinct information items, or *configuration items*, each of which is uniquely identified, and may have any amount of internal complexity;
- optionally, a directory system of some kind, in which the configurations items are organised.

Thus, in a software or document repository, the CIs are files arranged in the directories of the file system; in an EHR based on the GEHR or CEN models, they are Compositions arranged in a Folder structure. Contributions are made to the repository by users. This general abstraction is visualised in FIGURE 5.

6.2.2 Change Management

As implied earlier, change doesn't occur to CIs in isolation, but to the repository as a whole. Possible types of change include:

- creation of a new CI;

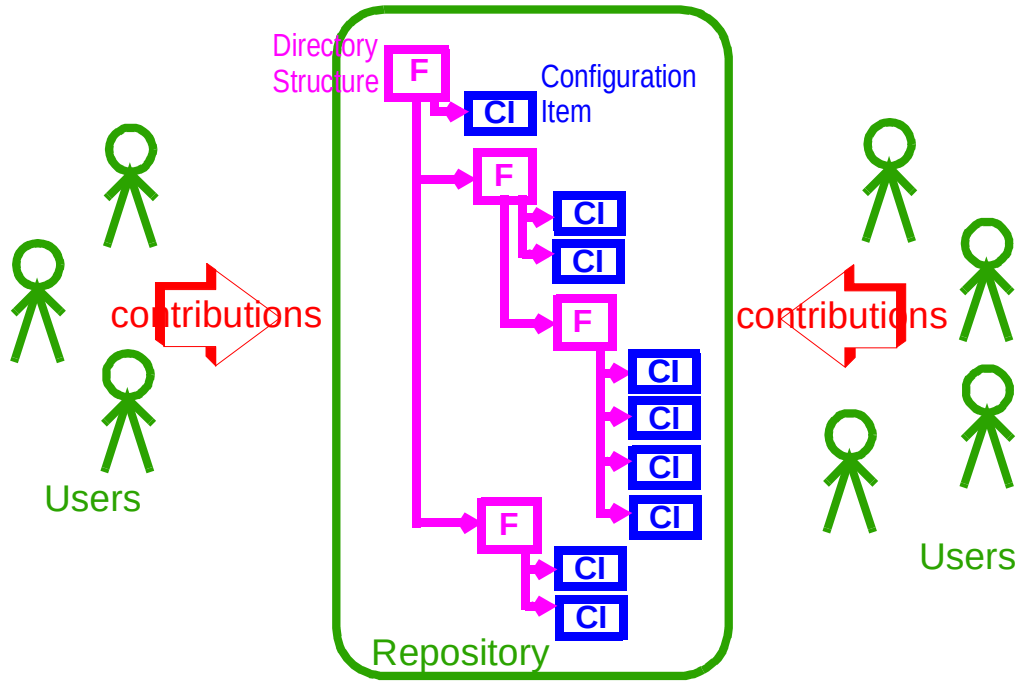

FIGURE 5 General Structure of a Controlled Repository

- removal of a CI;
- modification of a CI;
- creation of, change to or deletion of part of the directory structure;
- moving of a CI to another location in the directory structure.

The goal of configuration management is to ensure the following:

- the repository is always in a valid state;
- any previous state of the repository can be reconstructed;
- all changes are audit-trailed.

6.2.3 Changes in Time

Properly managing changes to the repository requires two mechanisms. The first, *version control*, is used to manage versions of each CI, and of the directory structure if there is one. The second is the concept of the “change-set”, or what we will call a *contribution*. This is the *set* of changes to individual CIs (and the directory structure) made by a user as part of some logical change. For example, in a document repository, the logical change might be an update to a document that consists of multiple files (CIs). There is one contribution, consisting of changes to the document file CIs, to the repository. In the EHR, a contribution might consist of changes to more than one Composition, and possibly to the organising Folder structure.

A typical sequence of changes to a repository is illustrated below. FIGURE 6 shows a notional repository containing a number of CIs in an initial state (note that the directory tree is not shown for the sake of simplicity).

The repository after four contributions is shown in FIGURE 7 (where each contribution is indicated by a blue oval). As each contribution is made, the repository is changed in some way. The first brings into existing a new CI, and modifies three others (changes indicated by the ‘C’ triangles). The second

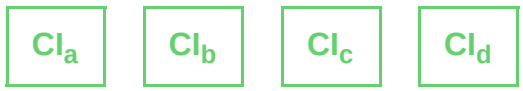

FIGURE 6 Initial State of Repository

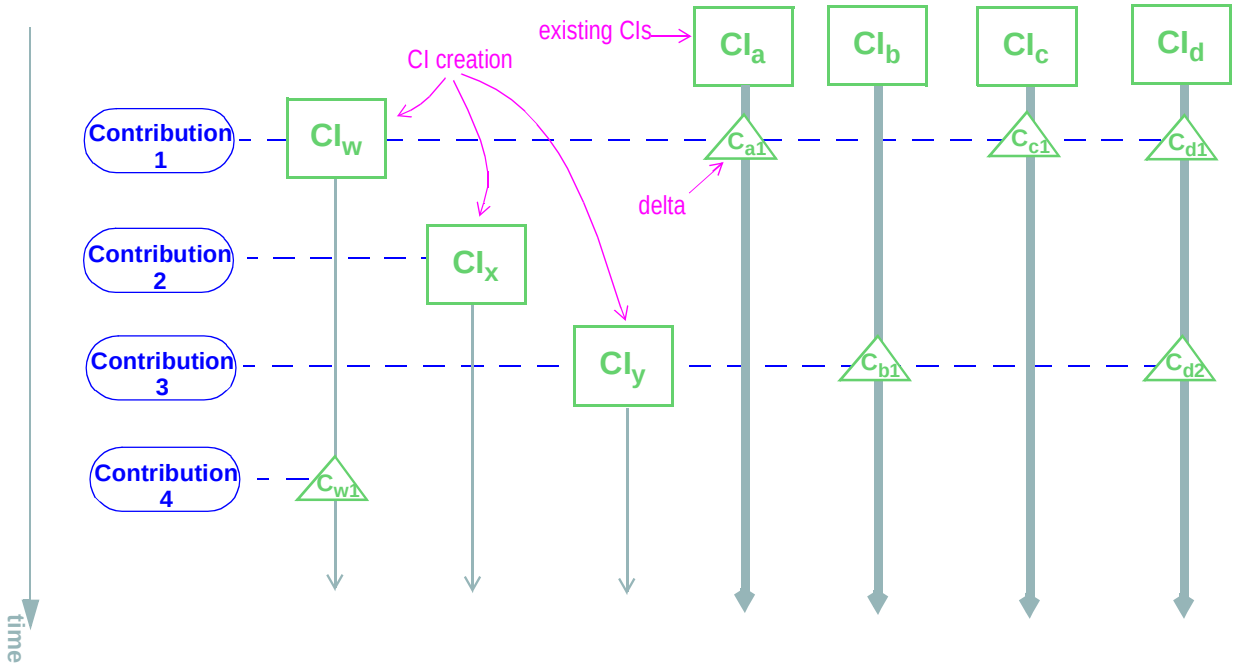

FIGURE 7 Contributions to the Repository (delta form)

contribution causes the creation of a new CI only. The third causes a creation as well as two changes, while the fourth causes only a change. (Again, changes to the folder structure are not shown here).

One nuance which should be pointed out is that, in FIGURE 7, contributions are shown as if they are literally a set of deltas, i.e. exactly the changes which occur to the record. Thus, the first contribution is the set $\{CI_w, C_{a1}, C_{c1}, C_{d1}\}$ and so on. Whether this is exactly true depends on the construction of applications. In some situations, some CIs may be updated by the user viewing the current list and entering just the changes - the situation shown in FIGURE 7; in others, the system may provide the current state of these CIs for editing by the user, and submit the updated versions, as shown in FIGURE 8. Some applications may do both, depending on which CI is being updated. The internal versioning implementation may or may not generate deltas as a way of efficient storage.

For our purposes here, we consider a contribution as being the logical set of CIs changed or created at one time, as implied by FIGURE 8.

6.2.4 General Model of a Change-controlled Repository

FIGURE 9 shows an abstract model of a change-controlled repository, which consists of:

- version-controlled configuration items - instances of `VERSION_REPOSITORY<T>`;
- `CONTRIBUTIONs`;
- an optional directory system of of folders. If folders are used, the folder structure must also be versioned as a unit.

The actual type of links between the controlled repository and the other entities might vary - in some cases it might be composition, in others aggregation; cardinalities might also vary. FIGURE 9 there-

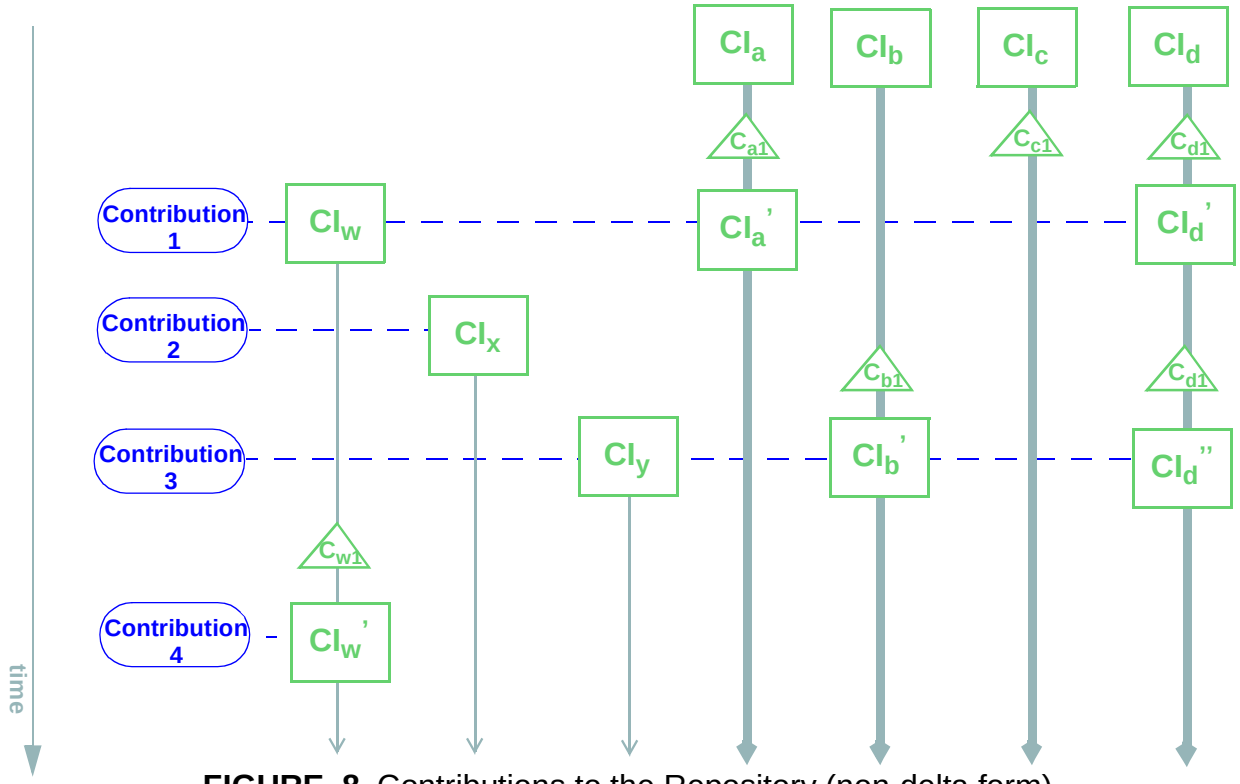

FIGURE 8 Contributions to the Repository (non-delta form)

fore provides a guide to the definition of actual controlled repositories, such as an EHR, rather than a formal specification for them.

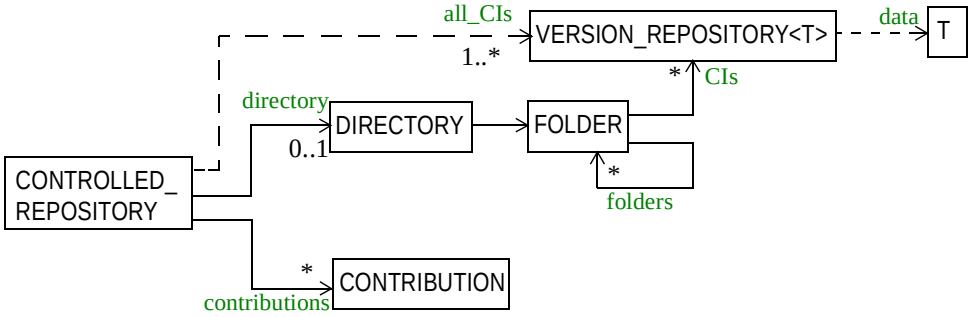

FIGURE 9 Informal Model of Change-controlled Repository

6.3 Formal Model

6.3.1 Versioned Items

FIGURE 10 illustrates a formal model of a version repository. In this model, the class `VERSION_REPOSITORY<T>` provides the versioning facilities for one CI, such as an EHR Composition, or a Party in a demographic system. Each version is an instance of the class `VERSION<T>`, which combines the data being versioned, the audit trail, and any attestations applied to the version. Both `VERSION_REPOSITORY<T>` and `VERSION<T>` are generic classes, with the generic parameter type T being the type of the data; ensuring that all versions in a given `VERSION_REPOSITORY` are of the same type, such as `ENTRY` or `FOLDER`, and that the repository itself is properly typed. Each `VERSION_REPOSITORY` has a unique identifier, its *uid* attribute, and a reference to the owning object

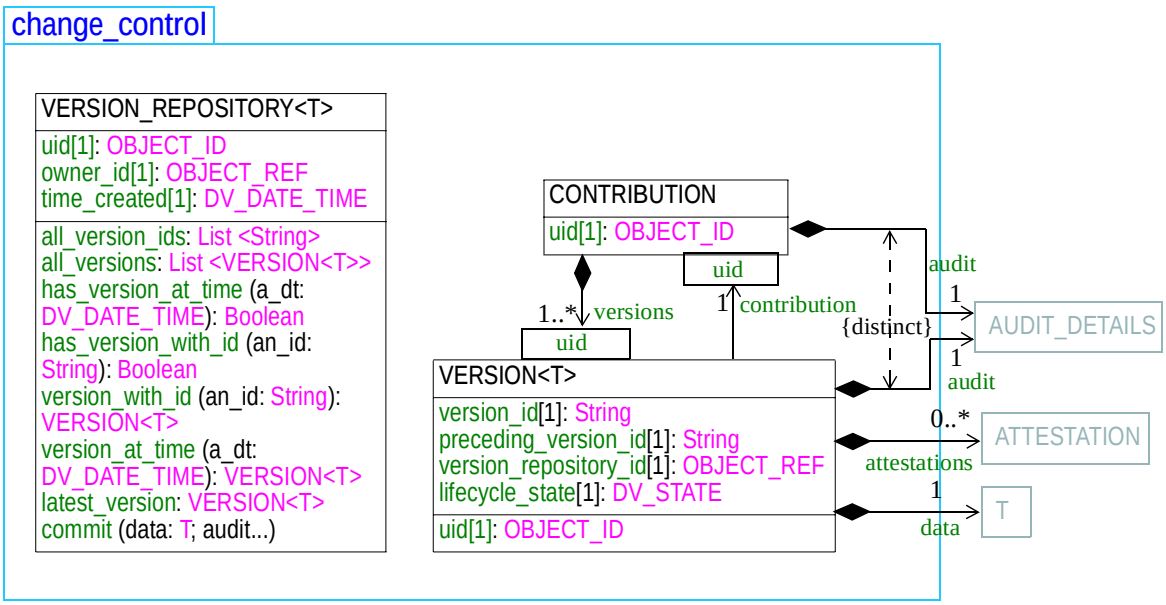

FIGURE 10 rm.common.change_control Package

- the *owner_id* attribute. The latter helps ensure that in storage systems, versioned objects are always correctly allocated to their enclosing repository, such as an EHR.

The *audit* attribute in `VERSION<T>`, of type `AUDIT_DETAILS`, includes of a set of attributes which form an audit trail, namely *system_id*, *committer*, *time_committed*, *change_type*, and *description*. The first three of these attributes always have values identical with the `CONTRIBUTION` instance for a given contribution, i.e. *they are copied in*. This is done to enable sharing of versioned entities independently of which Contributions they were part of.

The *attestations* attribute allows attestations to be associated with the data in the version. Attestations can be used as required by enterprise processes or legislation, and indicate who and when the item in question was attested. A digital “proof” is also required, although no assumption is made about the form of such proof. Normally, attestations refer to the entire version to which they are attached. However, it is possible for an `ATTESTATION` instance to refer to some finer-grained item within the data of the version, such as a single `ENTRY` within a `COMPOSITION`. If in subsequent versions, such an item is not changed (e.g. a different `ENTRY` in the same `COMPOSITION` is altered), then the relevant `VERSION` instances also need to refer to the `ATTESTATION` instances which remain valid. Since `ATTESTATIONs` are considered immutable objects once created, it does not really matter whether this is done by referring to a shared `ATTESTATION` instance, or by the use of copies.

6.3.2 Version Lifecycle

Versioned content has a lifecycle state associated with it, modelled using the `VERSION`.*lifecycle_state* attribute. Typically the lifecycle would follow the trajectory “draft” -> “active” -> “inactive”. No particular state machine is mandated here, although the state attribute is coded from the *open*EHR Terminology “version lifecycle state” group. Recording the state in `VERSION` enables two common scenarios to be faithfully represented.

The first is a user's need to save something in 'draft' state, e.g. because they have run out of time to finish writing part of the Composition, or due to an emergency. In hospitals this may well be a common occurrence. Such draft Compositions cannot be saved locally on the client machine, since this represents a security risk (a small client-side database would be much easier to hack into than a secure

server). They must therefore be persisted on the server, either in the actual EHR, or in a 'holding bay' which was recognised as not being part of the EHR proper. Either way, the author would have to explicitly retrieve the Composition(s) and after further work or review, 'promote' them into the EHR as 'active' Compositions; alternatively, they might decide to throw them away.

The second scenario is the need to be able to represent states like 'active', 'inactive' and 'deleted' of Compositions in a system. (The 'deleted' state is of course only logical - in a medico-legally safe record, nothing can be actually deleted).

In both scenarios, it should be possible to change the status of a Composition without actually changing the Composition itself - i.e. without creating a new version. This requirement prevents storing it in Composition itself; it is thus stored in the more logically correct place, `VERSION`. `VERSION`.*lifecycle_state* might in some systems be linked to the state of attestations (described below) which are also attached to `VERSION` objects.

6.3.3 Attestation of Versions

Scenarios relating to attestation may cause attestations to be created at different times with respect to the committal of data to the EHR, as follows:

- *at committal*: highly sensitive information is to be added to the EHR, e.g. recording the fact of sectioning of a patient under the mental health act, diagnosis of a fatal disease etc. In this case, attestation is added at committal to the EHR;
- *post-committal*: a data-entry person e.g. a secretary, transcriptionist or student is responsible for entering the data, including routine things such as referrals, discharge summaries etc, which need to be verified by the relevant clinician; this may occur after committal to the EHR in some cases, leading to the temporary presence of entries "awaiting attestation" in the record.
- *not required*: attestation not required - at many locations, many types of EHR additions do not require any special attestation at all, and can be committed to the EHR by a wider range of personnel.

As a result of these requirements, the model allows any number of attestations (from 0 to many) to be associated with each version of a versioned object. Attestations are considered to be neither part of the content, nor part of the audit information, but an external artifact which refers in to versions of versioned items. Attestations can be added at any time.

The class `CONTRIBUTION` defines the common audit information for the set of versions added to the repository due to a given contribution as well as a *description* of the contribution as a whole. `CONTRIBUTIONS` refer to their member `VERSION` objects via `OBJECT_IDs`; similarly, the audit object of any `VERSIONABLE` refers to its creating `CONTRIBUTION` using an `OBJECT_IDs` reference.

These classes can be used to provide versioning and contributions in repositories such as an EHR, or a demographic repository. In the EHR reference model for example, to obtain a versioned Composition, the type `VERSION_REPOSITORY<COMPOSITION>` is defined.

6.4 Class Descriptions

6.4.1 VERSION_REPOSITORY Class

CLASS	VERSION_REPOSITORY<T>	
Purpose	Version control abstraction, defining semantics for versioning one complex object.	
Attributes	**Signature**	**Meaning**
	uid: `OBJECT_ID`	Unique identifier of this version repository.
	owner_id: `OBJECT_REF`	Reference to object to which this versioned repository belongs, e.g. the id of the containing EHR.
	time_created: `DV_DATE_TIME`	Time of initial creation of this versioned object.
Functions	**Signature**	**Meaning**
	all_versions: `List <VERSION<T>>`	Return a list of all versions in this object.
	all_version_ids: `List <String>`	Return a list of ids of all versions in this object.
	version_count: `Integer`	Return the total number of versions in this object
	has_version_id (an_id: `String`): `Boolean` ***require*** an_id /= Void ***and then not*** an_id.is_empty	True if a version with id ‘an_id’ exists.
	has_version_at_time (a_time:`DV_DATE_TIME`): `Boolean` ***require*** a_time /= Void	True if a version for time ‘a_time’ exists.
	version_with_id (an_id:`String`): `VERSION<T>` ***require*** has_version_with_id(an_id)	Return the version with id ‘an_id’.

CLASS	VERSION_REPOSITORY<T>	
	version_at_time (a_time:DV_DATE_TIME): VERSION<T> ***require*** has_version_at_time(a_time)	Return the version for time 'a_time'.
	latest_version: VERSION<T>	Return the latest version.
	commit (an_audit: AUDIT_DETAILS; a_version: T) ***require*** an_audit /= Void a_version /= Void	Add a new version.
Invariant	***uid_exists***: uid /= Void ***owner_id_valid:*** owner_id /= Void ***time_created_exists***: time_created /= Void ***versions_exists***: version_count >= 1	

6.4.2 VERSION Class

CLASS	VERSION<T>	
Purpose	Versionable objects, with an audit trail containing details of change.	
Attributes	**Signature**	**Meaning**
	data: T	The data being versioned.
	attestations: List <ATTESTATION>	Set of attestations relating this version.
	audit: AUDIT_DETAILS	Audit trail of this version.
	version_id: String	Unique identifier of this version.
	preceding_version_id: String	Unique identifier of the version on which this version was based. May be the pseudo-version "first".
	version_repository_id: OBJECT_REF	A copy of the uid of the version repository to which this version was added.
	contribution: OBJECT_REF	Contribution in which this version was added.
	lifecycle_state: DV_STATE	Lifecycle state of the content item in this version.
Functions	**Signature**	**Meaning**

CLASS	VERSION<T>	
	uid: OBJECT_ID	Unique identifier of this version, derived from version repository id and version id.
Invariant	***version_id_exists***: version_id /= Void ***and then not*** version_id.is_empty ***preceding_version_id_exists***: preceding_version_id /= Void ***and then not*** preceding_version_id.is_empty ***version_repository_id_exists***: version_repository_id /= Void ***lifecycle_state_valid***: lifecycle_state /= Void ***and then*** terminology("openehr").codes_for_group_name("version lifecycle state", "en").has(lifecycle_state.value.defining_code) ***audit_exists***: audit /= Void ***attestations_valid***: attestations /= Void ***implies not*** attestations.is_empty ***Contribution_exists***: contribution /= Void ***uid_valid***: uid /= Void ***and*** uid.version_id.is_equal(version_id)	

6.4.3 CONTRIBUTION Class

CLASS	CONTRIBUTION	
Purpose	Documents a contribution of one or more versions added to a change-controlled repository.	
Attributes	**Signature**	**Meaning**
	uid: OBJECT_ID	Unique identifier for this contribution.
	versions: Set<OBJECT_REF>	Set of references to versions causing changes to this EHR. Each contribution contains a list of versions, which may include paths pointing to any number of VERSIONABLE items, i.e. items of type COMPOSITION and FOLDER.
	audit: AUDIT_DETAILS	Audit trail of this Contribution as a whole.
Invariants	***uid_exists***: uid /= Void ***audit_exists***: audit /= Void ***Versions_valid***: versions /= Void ***and then not*** versions.empty ***Description_exists***: audit.description /= Void	

6.5 Transaction Semantics of Contributions

In terms of database management, Contributions are considered as nested transactions. The attempt to commit a Contribution should only succeed if each VERSION instance in the Contribution is committed successfully. Failure to commit any of the member instances should cause failure of the Contribution.

7 Resource Package

7.1 Overview

The `common.resource` package defines the structure and semantics of the general notion of an online resource which has been created by a human author, and consequently for which natural language is a factor. The package is illustrated in FIGURE 11.

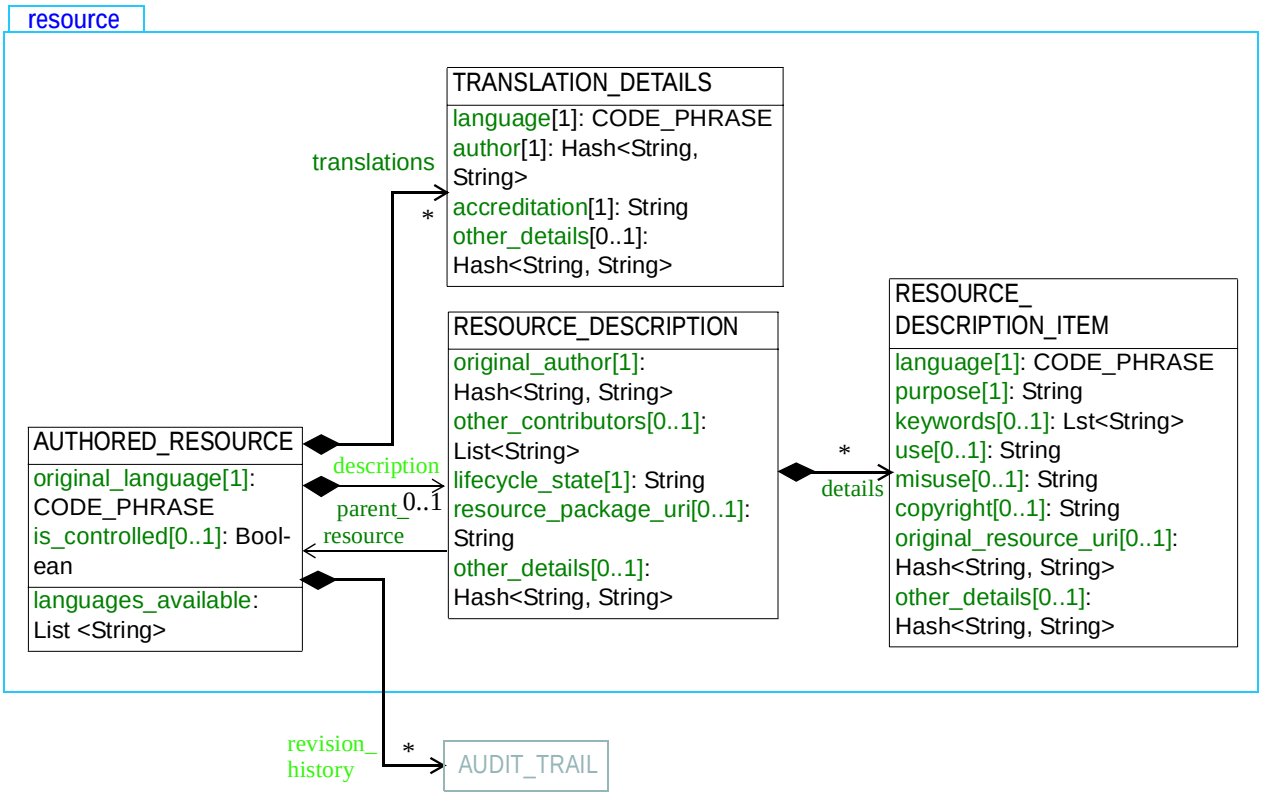

FIGURE 11 openehr.rm.common.resource Package

7.1.1 Natural Languages and Translation

Authored resources contain natural language elements, and are therefore created in some original language, recorded in the *orginal_language* attribute of the `AUTHORED_RESOURCE` class. Information about translations is included in the *translations* attribute, which allows for one or more sets of translation details to be recorded. A resource is translated by doing the following:

- translating every language-dependent element to the new language;
- adding a new `TRANSLATION_DETAILS` instance to *translations*, containing details about the translator, organisation, quality assurance and so on.
- any further translations to language-specific elements in a instances of descendent type of `AUTHORED_RESOURCE`.

The *languages_available* function provides a complete list of languages in the resource.

7.1.2 Meta-data

What is normally considered the 'meta-data' of a resource, i.e. its author, date of creation, purpose, and other descriptive items, is described by the `RESOURCE_DESCRIPTION` and

RESOURCE_DESCRIPTION_ITEM classes. The parts of this that are in natural language, and therefore may require translated versions, are represented in instances of the RESOURCE_DESCRIPTION_ITEM class. Thus, if a RESOURCE_DESCRIPTION has more than one RESOURCE_DESCRIPTION_ITEM, each of these should carry exactly the same information in a different natural language.

7.1.3 Revision History

The AUDIT_DETAILS class is concerned with the creation and modification of the resource in a repository. Each instance of this class in an actual resource represents one act of committal to the repository, with the attributes documenting who, when and why. Where resources are managed in a contolled document repository with versioning, audit information will be stored somewhere in the repository (e.g. in version control files); the revision_history within the resource is intended to act simply as a documentary copy, or trace of the revision history information so far, for the benefit of the reader of the resource. Given that resources in different places may well be managed in different kinds of repositories, having a copy of the revision history in a standardised form within the resource enables it to be used interoperably by resource tools.

Any change to a resource which is committed to the relevant repository causes a new addition to the *revision_history.* Some changes cause new revisions only, if they do not change the formal semantics of the resource, for example, the addition of a new language translation.

7.2 Class Definitions

7.2.1 AUTHORED_RESOURCE Class

CLASS	*AUTHORED_RESOURCE (abstract)*	
Purpose	Abstract idea of an online resource created by a human author.	
Attributes	**Signature**	**Meaning**
	original_language: CODE_PHRASE	Language in which this resource was initially authored. Although there is no language primacy of resources overall, the language of original authoring is required to ensure natural language translations can preserve quality. Language is relevant in both the description and ontology sections.
	translations: Hash <TRANSLATION_DETAILS, String>	List of details for each natural translation made of this resource, keyed by language. For each translation listed here, there must be corresponding sections in all language-dependent parts of the resource.
	description: RESOURCE_DESCRIPTION	Description and lifecycle information of the resource.

CLASS	*AUTHORED_RESOURCE (abstract)*	
	revision_history: `List<AUDIT_TRAIL>`	The revision history of the resource; only required if *is_controlled* = True (avoids large revision histories for informal or private editing situations).
	is_controlled: `Boolean`	True if this resource is under any kind of change control (even file copying), in which case revision history is created.
Functions	**Signature**	**Meaning**
	languages_available: `Set<String>`	Total list of languages available in this resource, dervied from *original_language* and *translations*.
Invariant	***original_language_valid***: original_language /= void **and then** language /= Void ***and then*** code_set("languages").has(original_language) ***revision_history_validity***: is_controlled **implies** (revision_history /= Void ***and then*** revision_history.is_empty)	

7.2.2 TRANSLATION_DETAILS Class

CLASS	TRANSLATION_DETAILS	
Purpose	Class providing details of a natural language translation.	
Attributes	**Signature**	**Meaning**
	language: `CODE_PHRASE`	Language of translation
	author: `Hash<String, String>`	Translator name and other demographic details
	accreditation: `String`	Accreditation of translator, usually a national translator's association id
	other_details: `Hash<String, String>`	Any other meta-data
Invariant		

7.2.3 RESOURCE_DESCRIPTION Class

CLASS	RESOURCE_DESCRIPTION	
Purpose	Defines the descriptive meta-data of a resource.	
Attributes	**Signature**	**Meaning**

CLASS	RESOURCE_DESCRIPTION	
	original_author: `Hash<String, String>`	Original author of this resource, with all relevant details, including organisation.
	other_contributors: `List<String>`	Other contributors to the resource, probably listed in “name <email>” form.
	lifecycle_state: `String`	Lifecycle state of the resource, typically including states such as: `initial`, `submitted`, `experimental`, `awaiting_approval`, `approved`, `superseded`, `obsolete`.
	details: `List<RESOURCE_DESCRIPTION_ITEM>`	Details of all parts of resource description that are natural language-dependent.
	resource_package_uri: `String`	URI of package to which this resource belongs.
	other_details: `Hash<String, String>`	Additional non language-senstive resource meta-data, as a list of name/value pairs.
	parent_resource: `AUTHORED_RESOURCE`	Reference to owning resource.
Invariant	***original_author_validity***: original_author /= Void **and then not** original_author.is_empty ***details_exists***: details /= Void **and then not** details.is_empty ***language_validity***: details.for_all (d | parent_resource.languages_available.has(d.language)) ***Parent_resource_valid***: parent_resource /= Void and then parent_resource.description = Current	

7.2.4 RESOURCE_DESCRIPTION_ITEM Class

CLASS	RESOURCE_DESCRIPTION_ITEM	
Purpose	Language-specific detail of resource description. When a resource is translated for use in another language environment, each `RESOURCE_DESCRIPTION_ITEM` needs to be copied and translated into the new language.	
Attributes	**Signature**	**Meaning**
	language: `CODE_PHRASE`	The localised language in which the items in this description item are written. Coded from *open*EHR Code Set “languages”.
	purpose: `String`	Purpose of the resource.
	keywords: `List<String>`	Keywords which characterise this resource, used e.g. for indexing and searching.

CLASS	RESOURCE_DESCRIPTION_ITEM	
	use: `String`	Description of the uses of the resource, i.e. contexts in which it could be used.
	misuse: `String`	Description of any misuses of the resource, i.e. contexts in which it should not be used.
	copyright: `String`	Optional copyright statement for the resource as a knowledge resource.
	original_resource_uri: `Hash<String, String>`	URIs of original clinical document(s) or description of which resource is a formalisation, in the language of this description item; keyed by meaning.
	other_details: `Hash<String, String>`	Additional language-senstive resource metadata, as a list of name/value pairs.
Invariant	***Language_valid***: language /= Void ***and then*** code_set("languages").has(language) ***purpose_exists***: purpose /= Void **and then not** purpose.is_empty ***use_valid***: use /= Void **implies not** use.is_empty ***misuse_valid***: misuse /= Void **implies not** misuse.is_empty ***copyright_valid***: copyright /= Void **implies not** copyright.is_empty	

7.2.5 AUDIT_DETAILS Class

THIS CLASS TO BE RATIONALISED WITH AUDIT_DETAILS IN Change_control package

CLASS	AUDIT_DETAILS	
Purpose	Revision history information for one committal of the resource to a repository.	
Attributes	**Signature**	**Meaning**
	committer: `Hash<String, String>`	Identification of the author of the main content of this resource, along with all relevant details.
	time_committed: `DATE_TIME`	Date/time of this change
	revision: `String`	Revision corresponding to this change. Various kinds of change cause only a new revision, not a version change, for example, adding a new translation, changing metadata.
	reason: `String`	Natural language reason for change.
	change_type: `DV_CODED_TEXT`	Type of change. Coded using the *open*EHR Terminology “audit change type” group.
Invariant	***committer_validity***: committer /= Void **and then not** committer.is_empty ***time_committed_exists***: time_committed /= Void ***reason_valid***: reason /= Void **implies not** reason.is_empty ***revision_valid***: revision /= Void **and then not** revision.is_empty ***Change_type_exists***: change_type /= Void ***and then*** terminology(“openehr”).codes_for_group_name(“audit change type”, “en”).has(change_type.defining_code)	

A References

A.1 General

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

2 Beale T *et al*. *Design Principles for the EHR*. See http://www.openEHR.org.

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

4 Schloeffel P. (Editor). Requirements for an Electronic Health Record Reference Architecture. International Standards Organisation, Australia; Feb 2002; ISO TC 215/SC N; ISO/WD 18308.

A.2 European Projects

5 Dixon R., Grubb P.A., Lloyd D., and Kalra D. *Consolidated List of Requirements. EHCR Support Action Deliverable 1.4*. European Commission DGXIII, Brussels; May 200159pp Available from http://www.chime.ucl.ac.uk/HealthI/EHCR-SupA/del1-4v1_3.PDF.

6 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.

7 Dixon R, Grubb P, Lloyd D. *EHCR Support Action Deliverable 2.4 "Guidelines on Interpretation and implementation of CEN EHCRA"*. Oct 2000. Available at http://www.chime.ucl.ac.uk/HealthI/EHCR-SupA/documents.htm.

8 Ingram D. *The Good European Health Record Project*. Laires, Laderia Christensen, Eds. Health in the New Communications Age. Amsterdam: IOS Press; 1995; pp. 66-74.

9 *Deliverable 19,20,24: GEHR Architecture.* GEHR Project 30/6/1995

A.3 CEN

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

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

12 ENV 13606-4 - *Electronic Healthcare Record Communication standard Part 4: Messages for the exchange of information*. CEN/ TC 251 Health Informatics Technical Committee.

A.4 OMG

13 CORBAmed document: *Person Identification Service*. (March 1999). (Authors?)

14 CORBAmed document: *Lexicon Query Service*. (March 1999). (Authors?)

A.5 Software Engineering

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

16 Fowler M. *Analysis Patterns: Reusable Object Models.* Addison Wesley 1997

17 Fowler M, Scott K. *UML Distilled (2nd Ed.).* Addison Wesley Longman 2000

A.6 Resources

18 IANA - http://www.iana.org/.

END OF DOCUMENT