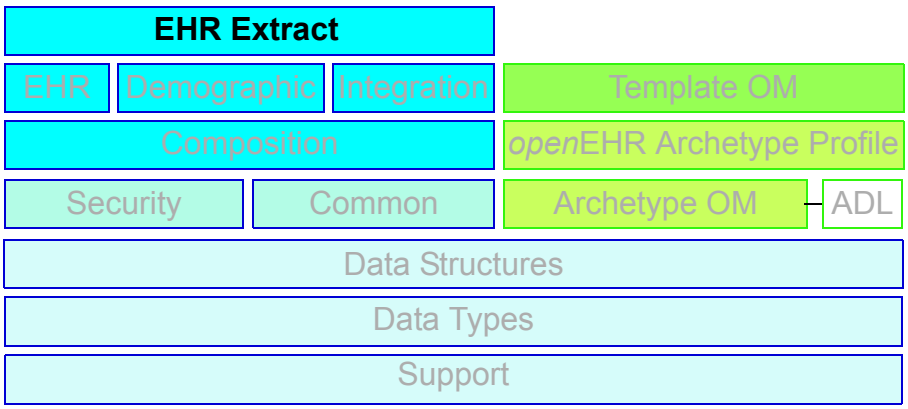
The *open*EHR Reference Model

Extract Information Model

Editors: {T Beale, H Frankel}[a]		
Revision: 2.0	*Pages:* 57	*Date of issue:* 20 Feb 2007

a. Ocean Informatics

Keywords: EHR, reference model, extract, openehr



Copyright Notice

Amendment Record

Issue	Details	Who	Completed
RELEASE 1.0.1			
2.0	**CR-000189**. Add LOCATABLE.*parent*. New invariant in EHR_EXTRACT. **CR-000186**: Upgrade EHR_EXTRACT to Release-1.0. Major redevelopment **CR-000219**: Use constants instead of literals to refer to terminology in RM.	T Beale T Beale R Chen	20 Feb 2007
RELEASE 1.0			
RELEASE 0.95			
1.3.5	**CR-000118**. Make package names lower case.	T Beale	10 Dec 2004
RELEASE 0.9			
1.3.4	**CR-000041**. Visually differentiate primitive types in openEHR documents.	D Lloyd	04 Oct 2003
1.3.3	**CR-000013**. Change key class names, according to CEN ENV 13606.	S Heard, D Kalra, D Lloyd, T Beale	15 Sep 2003
1.3.2	**CR-000003**, **CR-000004** (changes to versioning and LOCATABLE). MESSAGE_CONTENT now inherits from LOCATABLE.	T Beale, Z Tun	18 Mar 2003
1.3.1	**Formally validated using ISE Eiffel 5.2.** Revised structure of MESSAGE class to align better with CEN 13606-4. Renamed EHR_EXTRACT.*hca_authorising* to *originator*, similar to 13606.	T Beale	26 Feb 2003
1.3	**Changes post CEN WG meeting Rome Feb 2003.** Added attestations to X_TRANSACTION class. Significantly improved documentation of requirements, comparison to CEN 13606-4.	T Beale, S Heard, D Kalra, D Lloyd	07 Feb 2003
1.2.2	Minor corrections to diagrams and class definitions.	Z Tun	08 Jan 2003
1.2.1	Added senders_reference to conform to CEN 13606-4:2000 section 7.4.	T Beale	04 Jan 2003
1.2	Rewritten and restructured as two packages.	T Beale	07 Nov 2002
1.1	Moved part of EHR_EXTRACT into MESSAGE. Allow for multi-level archetypable Folder structures.	T Beale, D Kalra, D Lloyd	07 Oct 2002
1.0	Taken from EHR RM.	T Beale	07 Oct 2002

Acknowledgements

Thanks to...

The work reported in this paper has been funded in by a number of organisations, including The University College, London; Ocean Informatics, Australia; The Cooperative Research Centres Program

through the Department of the Prime Minister and Cabinet of the Commonwealth Government of Australia.

Table of Contents

1 Introduction

1.1 Purpose

This document describes the architecture of the *open*EHR EHR Extract Information Model. This model is equivalent in scope to the CEN ENV 13606:2000 part 4 standard.

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

1.2 Related Documents

Prerequisite documents for reading this document include:

- The *open*EHR Architecture Overview
- 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
- The *open*EHR Common Information Model
- The *open*EHR EHR Information Model
- The *open*EHR Demographic 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/TRUNK/publishing/architecture/rm/ehr_extract_im.pdf. New versions are announced on openehr-announce@openehr.org.

THIS DOCUMENT IS UNDER ACTIVE DEVELOPMENT AND IS NOT YET SUBJECT TO ARB CONTROL.

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 Requirements

2.1 Overview

This section describes the requirements that the *open*EHR Extract Information Model (IM) is designed to meet. Requirements are expressed in terms of a description of the assumed operational environments (which acts as a design constraint), a set of use cases, functional and security requirements. The family of use cases describe coarse-grained information import to and export from health information systems, using *open*EHR standardised information structures as the *lingua franca*. The Extract IM is neutral with respect to the communication technology used between systems: the information structures can equally be used in a web services environment or in a messaging environment, including secure email. The concrete method of communication is therefore not a factor in the scenarios described here.

2.2 Operational Environment

2.2.1 *open*EHR Environments

The assumed operational *open*EHR environment for *open*EHR Extracts is shown in FIGURE 1. In this figure, a Request for "information from the records of one or more 'subjects'" is created by a Requesting system. A *subject* record may be a patient EHR, a Person record in a demographic system, or any other logically meaningful top-level entity. Responding system(s) reply in the form of one or more Extracts. The Request/Response interaction is enabled by a transport mechanism and possibly other services. This be be in the form of comprehensive middleware, web services, or simple point-to-point protocols such as SMTP (email) transport.

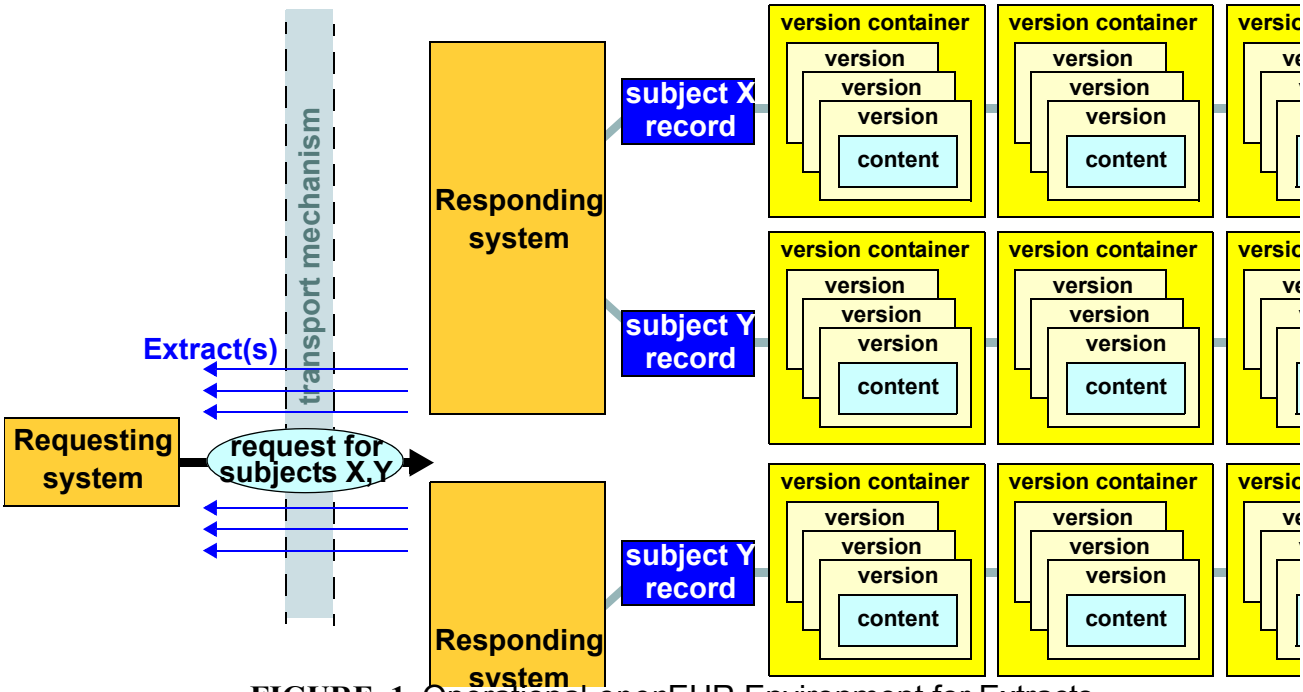


FIGURE 1 Operational *open*EHR Environment for Extracts

Information in Responding systems is assumed to be in the following form.

- Each such system contains one or more Subject records (e.g. EHRs); there may be records for the same Subject in more than one system.
- Each Subject record consists of one or more Version containers, each of which contains the version history of a particular piece of content. For an EHR, distinct containers are used for persistent Compositions (e.g. “medications list”, “problem list”) and event Compositions (Compositions created due to patient encounters and other events in the clinical care process).
- Each Version within a Version container corresponds to one version of the content managed by that container, in other words the state of a particlar content item at some point in time when it was committed by a user.
- Groups of Versions, each one from a different Version container within a Responding system correspond to Contributions, i.e. the *open*EHR notion of a “change-set”. Any particular Contribution corresponds to the set of Versions committed at one time, by a particular user to a particular system.

The above relationships reveal a hierarchy of potential 1:N relationships in the information accessible to the requesting system, with Contributions forming an alternative view of the content. At each level of the hierarchy, a system of identifiers is needed, e.g. to distinguish Subjects, to distinguish Versions and so on. In some specific circumstances, some of these may be reduced to 1:1 relationships, e.g. there may be no versioning, removing the need for specific identifiers for versions of an item.

2.2.2 Non-*open*EHR Systems

It is expected that some non-*open*EHR systems will use *open*EHR Extracts to either receive or send information. Not much can be assumed about the internal data architecture of such systems. For the purposes of this specification, the existence of two levels of data hierarchy is assumed:

- “records”, each corresponding to a ‘subject’ (including demographic subjects);
- an equivalent of Compositions or “documents”, which are the coarsest grain item making up a record.

Nothing is assumed about versioning in non-*open*EHR systems.

2.2.3 Location of Information

In more advanced environments, there may be a health information location service which obviates the need for any knowledge on the part of the requestor about which systems contain information on a particular Subject of interest (e.g. a certain patient); in simpler environments, the requesting system may need to explicitly identify the target systems of the request. FIGURE 2 illustrates a direct request and a request mediated by a location service.

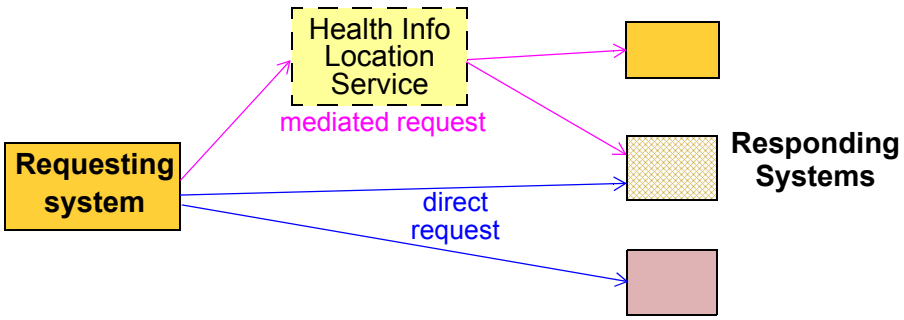


FIGURE 2 Health Information Reference Environment

This specification assumes that the EHR Request and Extract is between the Requesting system and each Responding system, even if the list of relevant Responding systems has been generated by a location service. In other words, this Extract specification does not encompass the idea of a compendium of Extracts from multiple Responding systems.

2.2.4 Granularity of Extract Data

In FIGURE 1 the lowest level of information shown in a Responding system is simply marked "content". This corresponds to top-level information structures such as Compositions, Folder trees, Parties etc in *open*EHR. Each such content item potentially contains a whole hierarchy of information items, only some of which is generally of interest to the Requestor. The typical database idea of a "query result" is usually expected to return only such fine-grained pieces. However, the Extract specification here only allows for a granularity of Compositions ("documents" etc), rather than fine-grained query responses which are dealt with by other means. This is because the primary use case of an Extract is to make parts of an EHR available somewhere else, rather than to intelligently query the record *in situ* and return the result.

2.2.5 Time

Versioned health record systems are "bitemporal" systems, because they include two notions of time. Times mentioned in the data relate to *real world* events or states, such as the time of diagnosis of diabetes, or the date of discharge of a patient from a hospital. Real world time is distinquished from *system time*, which is the time of events in the information system itself, such as committal of Contributions. Both real world time and system time may need to be specified in a Request.

2.3 Use Cases

The following sections describe typical use cases that the Request/Extract model must satisfy.

2.3.1 Single Patient, *Ad Hoc* Request

A key clinical use case is the need to obtain some or all of a patient's EHR from a remote system or systems. The request is most likely to be made due to the patient having been referred to another provider (including discharge to tertiary care), but other reasons such as falling ill while travelling will also figure.

The request might be made to an identified system, such as the system of the referring provider, or it may be made to all systems containing data on the given patient, if a health information location service is available.

The contents of the request may be specified in different ways, either by the clinician and/or the software, as follows:

- get this patient's entire EHR from an identified source for the first time;
- get all changes to this patient's EHR from specified (e.g. referring or GP) system since the last time this was done by me;
- get persistent Compositions such as "current medications", "problem list" and "allergies and interactions";
- get Compositions matching a specific query, such as "blood sugar level measurements during the last six months".

The meaning of time is content-dependent. For Observations in an *open*EHR EHR, the sample times might be specified; for an Evaluation that documents a diagnosis, times for date of onset, date of last episode, date of resolution could all be specified.

2.3.2 Multiple Patient, Batch Send

A very common requirement for pathology laboratories is to be able to send result data for tests done for a number of patients, on a periodic batch basis, to a known receiver such as a hospital, clinic or state health system. The batch send is usually seen as a "standing request" by the receiver, and may be on a periodic basis (e.g. send all the results available every hour), an "as-available" basis, or according to some other scheme.

Such data are currently often sent as HL7v2, Edifact or other similar messages.

To Be Continued: patient per message? Send trigger?

To Be Continued: Extract may be sent unsolicited - i.e. no Request

2.3.3 Previous Versions and Revision Histories

In some circumstances, a request may be made for versions other than the latest of certain content. This might happen due to a study or medico-legal investigation that needs to establish what information was visible in certain systems at an earlier time. For example there may be a need to determine if the problem list, list of known allergies, and patient preferences were all compatible with the medications list at some earlier time.

As part of querying for previous versions, the revision histories of Versioned containers might be requested, in order to allow a user or software agent to determine which Versions are of interest.

2.3.4 Systematic Update and Persisted Requests

In larger healthcare delivery environments such as state and regional health services, patients are routinely treated by multiple providers, some or all of which are part of a large distributed clinical computing environment. They may visit various clinics, specialists and hospitals, each of which has its own patient record for each patient. However, there is usually a need for centralised aggregation of patient data within the overall health authority, with updating required on a routine basis.

In such situations, the general requirement is for a request for update, typically for more than one patient, to be made once, and for it to be acted upon repeatedly until further notice. Specific requirements may include:

- periodic updates of changes since last update, with a specified period;
- event-driven updates, whereby an update occurs when a certain event occurs in the server, e.g. "any change to the EHR", "any change to medications or allergies" and so on.

For these situations, the request can be persisted in the server. Even for one-off *ad hoc* requests, the requestor may require the request to be persisted in the server, so that it can be referred to simply by an identifier for later invocations.

2.3.5 Sharing of non-EHR *open*EHR Data

There will be a need to be able to request information from *open*EHR systems and services other than the EHR, such as demographics and workflow, as they are developed. One likely purpose for such requests is for import from *open*EHR systems into non-*open*EHR systems, for example from an *open*EHR demographics service to an existing hospital Patient Master index.

It should be possible to use the same general form of request as used for an EHR. However, instead of specifying Extracts of patient records (EHRs), the data shared in these other cases will be whichever top-level business objects are relevant for that kind of service, e.g. PARTYs for the demographic service and so on.

2.3.6 Provision of data from non-*open*EHR Systems

One of the major uses of an *open*EHR system is as a shared EHR system, aggregating data from various existing systems in a standardised form. Data from such systems may be provided in different ways, including various messaging forms (HL7, Edifact), various kinds of EMR document (CEN EN13606, HL7 CDA), and other formats that may or may not be standardised.

The developers of some such systems may decide to provide an *open*EHR-compatible export gateway, capable of serialising various data into *open*EHR structures, particularly the Composition / `GENERIC_ENTRY` form (see *open*EHR Integration IM) which is highly flexible and can accommodate most existing data formats. Types of non-*open*EHR systems that may supply *open*EHR Extracts in this fashion include pathology systems and departmental hospital systems, such as radiology, (RIS), histopathology and so on.

Extract Requests might be specified in *open*EHR form (i.e. according to this document) or in some other form, such as web service calls or messages; either way, the logical request is the same, i.e. which patients, which content, which versions, and the update basis. The responses must be some subset of the *open*EHR Extract presented in this document.

2.3.7 Patient Access to Health Data

Direct access by patients to their own health data outside of clinical encounters is a common aspiration of e-health programmes around the world. It seems clear that there will be various modes by which such access will occur, including:

- patient carrying USB stick or other portable device containing some or all of health record;
- access from home PC via secure web service to EHR, in a manner similar to online banking;
- access to EHR data in the form of encrypted email attachments on home PC either sent unsolicited (e.g. a scan cc:d to patient by imaging lab) or by request of the patient;
- access to EHR in the waiting rooms of doctors' surgeries, clinics etc via kiosks or normal PCs.

Both the USB stick and email scenarios involve asynchronous access of EHR information, and can be addressed by the EHR Extract.

In the case of a portable device, the most obvious need is for the device to act as a synchronising transport between a home PC containing a copy of patient / family EHRs, and the EHR systems at various clinics that the patient visits. To achieve this, when changes are made at either place (in the home record or in the record held at a clinic), it should be possible to copy just the required changes to the device. In *open*EHR terms, this corresponds to copying Contributions made since the last synchronisation.

The email attachment scenario is more likely to involve Extracts containing either information requested by the patient in a similar manner as for the *ad hoc* clinical requests described above (e.g. most recent test result) or laboratory information in the form of an *open*EHR Extract, destined for integration into an *open*EHR EHR. In the latter case, the information is likely to be in the form of Compositions containing `GENERIC_ENTRIES`, built according to legacy archetypes, although it could equally be in "pure" *open*EHR form (i.e. Compositions containing proper *open*EHR Observations).

2.3.8 Move of Entire Record

A patient's entire EHR may be moved permanently to another system for various reasons, due to the patient moving, a permanent transfer of care, or re-organisation of data centres in which EHRs are managed. This is known as *change of custodianship* of the record, and is distinct from situations in which copying and synchronisation take place.

The record is deleted (possibly after archival) from active use at the sending system. In these situations, the usual need is for an interoperable form of the complete EHR (including all previous versions) to be exported from the existing system and sent to the destination system, since in general the two systems will not have the same implementation platform, versioning model and so on. In some cases, the implementations may be identical, allowing a copy and delete operation in native representation could be used.

2.3.9 System Synchronisation

Mirroring

Two *open*EHR systems containing the same kind of data (i.e. EHR, demographic etc) may contain records that are intended to be logical "mirrors" (i.e. clones) of each other. In some cases, an entire system may be a clone of another, i.e. a "mirror system". Mirrored records are purely read-only, and in all cases, the mirror record or system is a slave of its source, and no local updates occur. Synchronisation is therefore always in one direction only.

To maintain the information in mirrored records, an efficient update mechanism is required. The *open*EHR Contribution provides the necessary semantic unit because it is the unit of change to any record in a system, it can also be used as the unit of update to the same record in a mirroring system.

The Virtual EHR

If changes are allowed to multiple systems that also systematically synchronise, a "virtual EHR" exists. This term indicates that the totality of changes taken together form a complete EHR, even if any any particular instant in any given system, not all such changes are visible. The virtual EHR is the usual situation in any large-scale distributed e-health environment. Synchronisation might be on an *ad hoc* or systematic basis, and may or may not be bidirectional. The difference between a synchronisation request and any other kind of request is that the request is specified not in terms of a user query but in terms of bringing the record up to date, regardless of what changes that might require.

Due to the way version identification is defined in *open*EHR (see Common IM, `change_control` package), the virtual EHR is directly supported, and synchronisation is possible simply by copying Versions from one place to another and adding them to the relevant Versioned container at the receiver end. In a large health computing environment, cloning and mirroring might be used systematically to achieve a truly de-centralised system.

The defining condition of this use case is that one or more (possibly all) records in a system are maintained as perfect copies of the same records in other systems, with possible delays, depending on when and how updating occurs.

2.3.10 Communication between non-*open*EHR EMR/EHR systems

Since the *open*EHR Extract represents a generic, open standardised specification for representing clinical information, there is no reason why it cannot be used for systems that not otherwise implement *open*EHR. In this case, the Extract content is most likely to consist of Compositions and Generic_entries, and may or may not contain versioning information, depending on whether versioning is supported in the generating systems.

2.4 Technical Requirements

2.4.1 Specification of Content

Content is specifiable in terms of matching criteria. This can take two forms: lists of specific top-level content items, or queries that specify top-level items in terms of matching subparts.

To Be Continued:

Queries are expressed in the Archetype Query Language.

To Be Continued:

2.4.2 Specification of Versions

The *open*EHR Extract supports detailed access to the versioned view of data. Which versions of the content should be returned can be specified in a number of ways:

- as a version time of the source EHR at which all content should be taken
- in more specific terms, such as:
  - time window of committal;
  - with or without revision history, or revision history only;
  - all, some, latest versions of each content item;
- in terms of identified Contributions, Contributions since a certain time etc.

To Be Continued:

2.4.3 Completeness of Data

Information transferred in an EHR Extract needs to be self-standing in the clinical sense, i.e. it can be understood by the requestor without assuming any other means of access to the responding system. In general, this means that for references of any kind in the transferred EHR data, the Extract needs to either contain the reference target, or else it must be reasonable to assume that the requestor has independent access, or else doesn't need it.

References to Other Parts of the Same EHR

In *open*EHR, there are two kinds of cross reference within an EHR: `LINKs` (defined in `LOCATABLE`), and hyperlinks (`DV_TEXT`.*hyperlink*). Both of these use a `DV_EHR_URI` instance to represent the link target. The contents of the URI are defined in the Architecture Overview.

To Be Continued: Are EHR ids always included in URIs?

In some cases, referenced items within the same EHR will need to travel with the originally requested item, in order for the latter to make sense. For example, a discharge summary or referral might refer (via `LINKs`) to other Compositions in the EHR, such as Medication list, Problem list, and lab reports. On the other hand, there may be links within the requested item to objects that are not required to be sent in an Extract.

Which links should be followed when building an Extract can be specified in terms of:

- link depth to follow, i.e. how many jumps to continue from the original item; a value of 0 means "don't follow any links".

In addition, for `LINK` instances, the following can be specified:

- link type to follow, i.e. only follow links whose *type* attribute matches the specification.

References to Other EHRs

References to items in other EHRs may occur within an EHR, e.g. to the EHR of a parent, other relation, organ donor etc. There is no requirement for such links to be followed when constructing EHR Extracts.

References to Resources Outside the EHR

Computable references can also be made to external items from within an *open*EHR EHR. Instances of the data type DV_URI occurring either on their own or in a DV_TEXT hyperlink are typically used to refer to resources such as online guidelines and references. Instances of DV_MULTIMEDIA can contain DV_URI instances that refer to multimedia resources usually within the same provider enterprise, e.g. radiology images stored in a PACS. Since URIs are by definition globally unique, they remain semantically valid no matter where the data containing them moves. However, there is no guarantee that they can always be resolved, as in the case of a URI referring to a PACS image in one provider environment when transferred to another. This is unlikely to be a problem since images are usually represented in the EHR as a small (e.g. 200kb) JPG or similar, and it is almost never the intention to have original image sets (which may be hundreds of Mb) travel with an EHR. Requests to access original images would be made separately to a request for EHR Extracts.

References to Demographic Entities

Two kinds of demographic entities are referred to throughout an *open*EHR EHR. Individual providers and provider institutions are referenced from PARTY_PROXY objects in the record, via the *external_ref* attribute, which contains references to objects within a demographic repository, such as an *open*EHR demographic repository, a hospital MPI or a regional or national identity service. The PARTY_IDENTIFIED subtype of PARTY_PROXY can in addition carry human readable names and other computational identifiers for the provider in question.

The second kind of demographic reference, found in the PARTY_SELF subtype of PARTY_PROXY, is to the EHR subject (i.e. patient), and may or may not be present in the EHR, depending on the level of security in place. Where it is present, it is to a record in a demographic or identity system.

For the contents of an EHR Extract to make sense to the receiver, referenced demographic items may have to be included in the Extract, if the receiver has no access to a demographic system containing the entities. Whether patient demographics are included is a separate matter, since the requestor system already knows who the patient is, and may or may not need them. The requestor should therefore be able to specify whether the Extract includes referenced demographic entities other than the subject, and independently, whether subject demographics should be included.

Archetypes and Terminology

Another kind of “reference” is terminology codes, stored in instances of the data type DV_TEXT (via the *mapping* attribute) and DV_CODED_TEXT.*defining_code*. In *open*EHR systems, all coded terms (i.e. instances of DV_CODED_TEXT) carry the text value of the code, in the language of the locale of the EHR. For normal use, this is typically sufficient. However, for the purposes of decision support or other applications requiring inferencing, the terminology itself needs to be available. This specification assumes that where the requestor requires inferencing or other terminology capabilities, independent access to the complete terminology will be obtained.

The same assumption is made with respect to archetypes whose identifiers are mentioned in the EHR data or meta-data: archetype are not themselves included in Extracts, and have to be resolved separately.

2.4.4 Security and Privacy

Security becomes one of the most important requirements for the EHR Extract for the obvious reason of its exposure in potentially uncontrolled environments outside the (supposedly) secure boundaries of requesting or responding systems. The general requirement is that the contents of an Extract are based on:

- the access control rules defined in the `EHR_ACCESS` object at source;
- any other access rules defined in policy services or other places;
- authentication of the requesting user.

Digital signing should be used based on the (preferably certified) public key of the requestor. Notarisation might also be used to provide non-repudiable proof of sending and/or receiving Extracts, although this is outside the scope of this specification.

2.4.5 Update Basis

In addition to specifying the content a basis for update also needs to be specified. The simplest possible case is that of an *ad hoc* one-off query.

More complex cases are periodic update and event-driven update.

Persistent Request

`To Be Continued:`

3 Overview

3.1 Design Approach

The *open*EHR Extract model uses two design ideas. Firstly, the notion of a Request and an Extract (the reply) are clearly distinguished. Extracts may include a copy of the Request to which the Extract is a reply, indicating what is actually in the Extract, which may differ from what was requested. Secondly, the common semantics of Requests and Extracts are modelled in a generic way, with a number of specialised Request and Extract types being based on the common classes. Different concrete types of Extract are thus used to satisfy particular groups of requirements, rather than trying to make one kind of Extract perform all possible tasks. FIGURE 3 illustrates key Extract communication scenarios, along with the various concrete Extract types defined by the model.

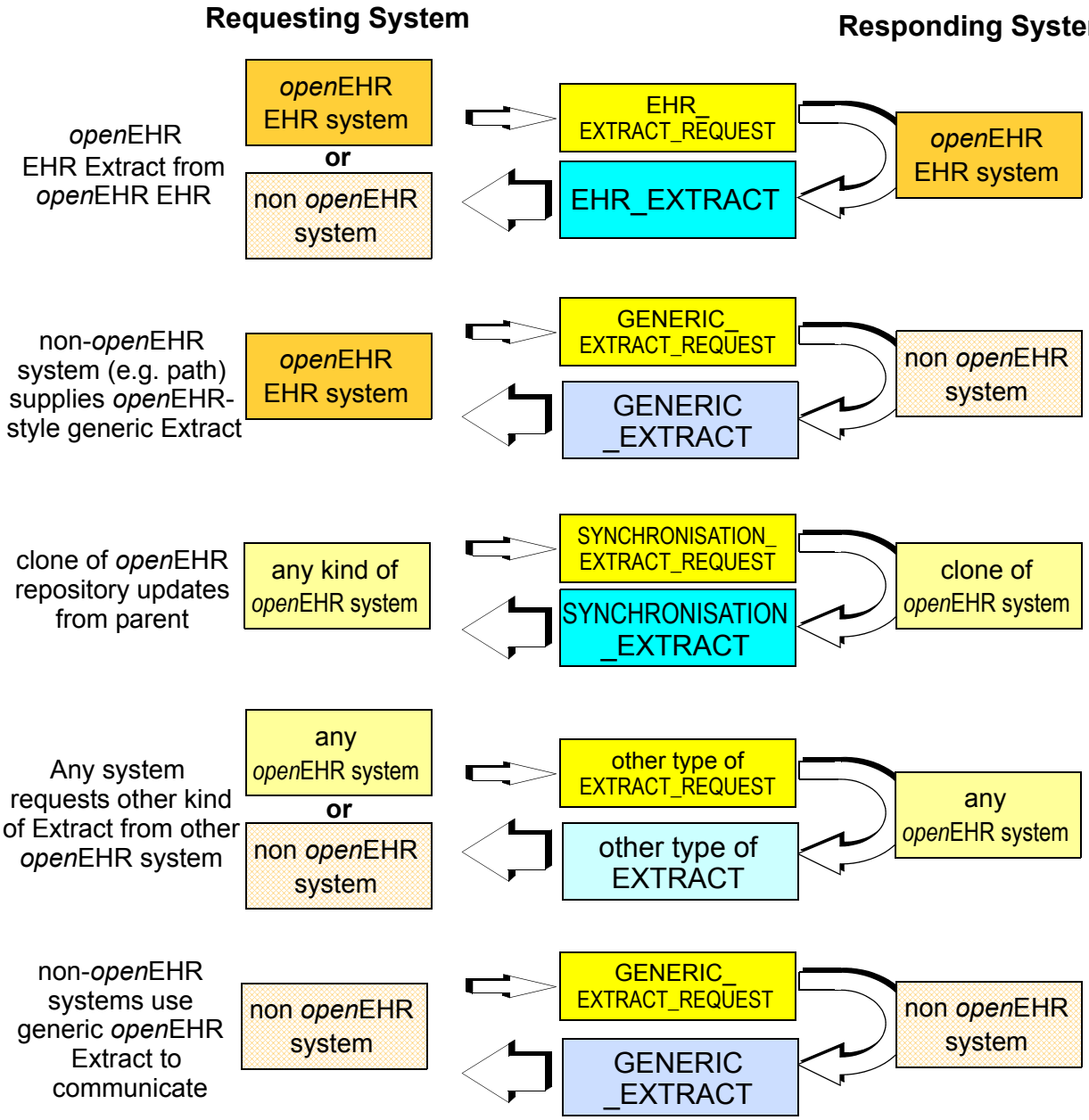


FIGURE 3 Use cases for *open*EHR Extracts

The EHR Extract is just one concrete type of Extract. The other types include:

- *Generic Extract*: an Extract type designed for use with non-*open*EHR systems communicating to *open*EHR systems, and users of the CEN EN 13606 Extract specification. The Generic Extract assumes the absolute minimum about what is in a system, while remaining withing the *open*EHR type system;
- *Synchronisation Extract*: an Extract used for updating mirrored EHRs across systems; uses Contributions as the grouping concept.

Other types of Extract may be defined in the future.

3.2 Package Structure

The `rm.extract` package defines the semantics of Extracts from *open*EHR data sources, including EHRs. The modelling approach taken is to define generic Request and Extract types, whose semantics are extended in various specific kinds of Extract, such as for an EHR or demographic system. Further flexibility is provided in terms of the form of data sent in an Extract, such as whether it is *open*EHR versioned data (sufficient for recreating versions at the receiver end) or more basic forms of data e.g. with no assumed versioning (sufficient for use in a CEN EN13606-like. environment). FIGURE 4 illustrates the package structure of the `rm.extract package`.

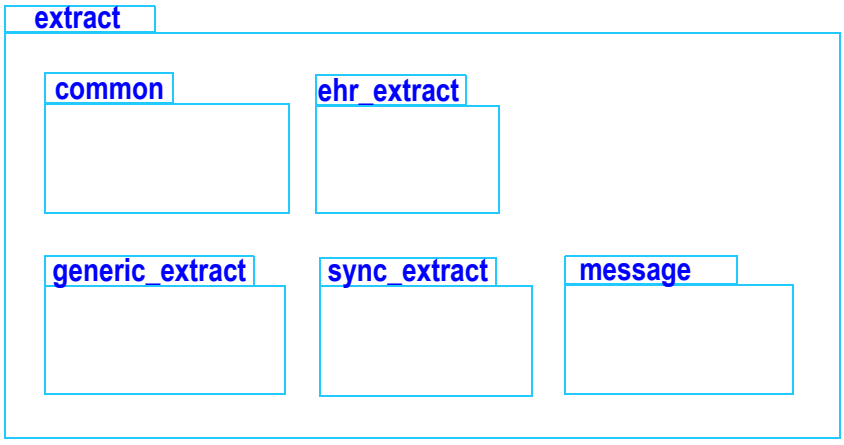


FIGURE 4 rm.extract Package

The sub-packages are as follows:

- `common`: semantics common to all Extracts;
- `ehr_extract`: semantics for the EHR Extract type;
- `generic_extract`: defines semantics of the Generic Extract type;
- `synchronisation_extract`: defines semantics of the Synchronisation Extract type;
- `message`: simple model of a message containing an Extract.

4 Extract.common Package

4.1 Overview

The `rm.extract.common` package defines the semantics common to all kinds of Extract requests and Extracts. Requests and Extracts can be implemented as messages, or used as types in a webservice environment. In the latter, the Extract request semantics would most likely be replaced by equivalent service function definitions.

The Request contains a detailed specification of the repository content required in the Extract. A Request is not always needed, and an Extract may be sent unsolicited. Requests can be made persistent and/or event-driven, supporting various periodic and continuous update scenarios. Each request may specify the data from one or more entities, e.g. EHRs or subjects.

The Extract reply may optionally include a copy of the request, while the main content is in the form of *chapters*, each containing the requested data for one entity, such as an EHR. Each chapter may contain a directory (i.e. folder structure), for CEN EN13606 compatibility. The detailed structure of the content is defined in various specialised variants of the common Extract model.

The instance structure of the `EXTRACT_REQUEST` and `EXTRACT` types is illustrated in FIGURE 5. In this figure, abstract types are indicated in italics, meaning that the actual types in a real Request and Extract will be subtypes of the relevant types shown here.

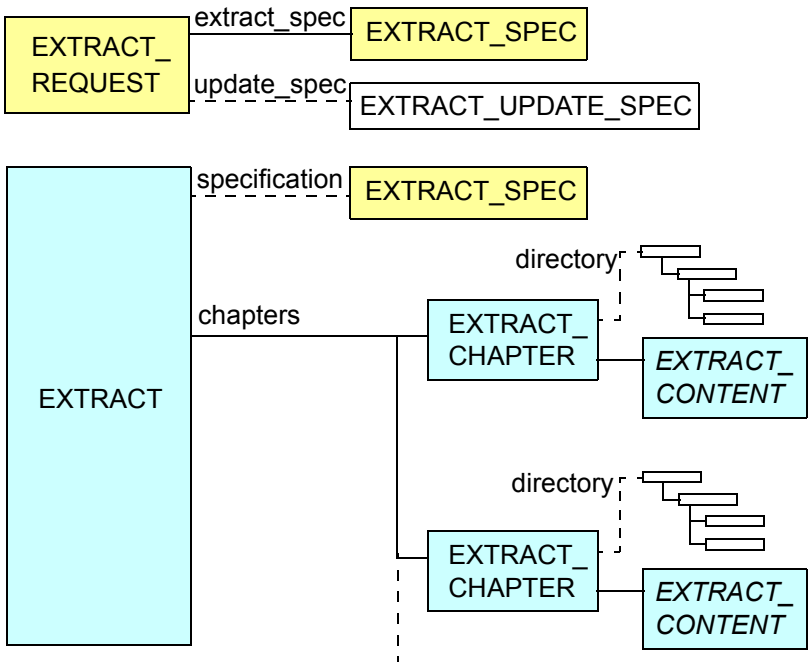


FIGURE 5 Request and Extract structure

4.2 Design

4.2.1 Extract Request

FIGURE 6 illustrates the `rm.extract.common` package. The `EXTRACT_REQUEST` class consists of an *update_spec* specifying rules for update (one-off, periodic, event-driven etc), and a *extract_spec*,

indicating what information from the target repository is required. The latter consists of an optional *version_spec*, indicating which versions (default is latest) and a *manifest*, specifying which entities (records or subjects, and optionally which items from those entities should be included.

The *update_spec* part of the Request specifies how the Request is to be processed by the server. The default situation, requiring no *update_spec* at all, is a one-off request. Other alternatives include:

- repeat Reqest with a defined period and/or trigger events;
- persisting the Request in the server so that the requestor can make later repeat (but *ad hoc* requests simply by referring to a previously stored request by its identifier.

An *ad hoc* repeat of a request persisted earlier is made using an `EXTRACT_REQUEST_REPEAT`. This specifies only the identifier of a request previously specified using an `EXTRACT_REQUEST`. Since requests are uniquely identified for all time, there can be no error in the identification.

The *extract_spec* part of the Request applies to all chapters in the Extract. The core attributes indicate the following:

- what kind of Extract this is, e.g. “openehr-ehr”, “openehr-demographic”, “openehr-synchronisation”, “openehr-generic”, “generic-emr”, etc;
- what directory structure to use in the Extract if any; specified as an archetype identifier;
- an *includes_multimedia* flag indicating whether inline binary objects are included or not;
- a *link_depth* value, indicating how many levels of link to follow when constructing the content of the Extract;
- a *criteria* attribute, specifying queries defining the required content of each entity record;
- *other_details*, an archetypable structure including further extract details.

The remainder of the specification consists of two further structures:

- a *version_spec*, specifying which versions of the content are required; this is expressed in terms of the flags *all_versions* (i.e. all versions available of each item); *include_revision_history* for each item and *include_data* (which can be set False to get revision histories only), as well as a time range of versions;
- a *manifest* indicating the entities for which record extracts will be required, and optionally identifiers of particular items (e.g. Compositions) known to be part of that entity. An entity is identified using an instance of `EXTRACT_ENTITY_IDENTIFIER`, which allows for identification of the record or the subject in question, depending on what is available in the deployment environment.

4.2.2 Content Specification

The content to be included in an Extract is specified in three dimensions. The *manifest* specifies which entity records are to be included, thus determining who the Extract is about - one patient or many. The *criteria* attribute specifies the semantic criteria to be matched within a given entity record, e.g. items corresponding to certain dates, types of clinical activity and so on. The 3rd dimension is the *version_spec*, which indicates which versions in system time are to be obtained of the items defined by the manifest and criteria attributes. The details are as follows.

The Manifest

It is the *manifest* that decides the scope of records to be retrieved. In the simplest case, only a single entity and no items will be identified; this will be used in the vast majority of single-patient request scenarios. However, some scenarios will be of a batch update nature, including pathology lab result update and situations where patient records are requested corresponding to a list of referrals received

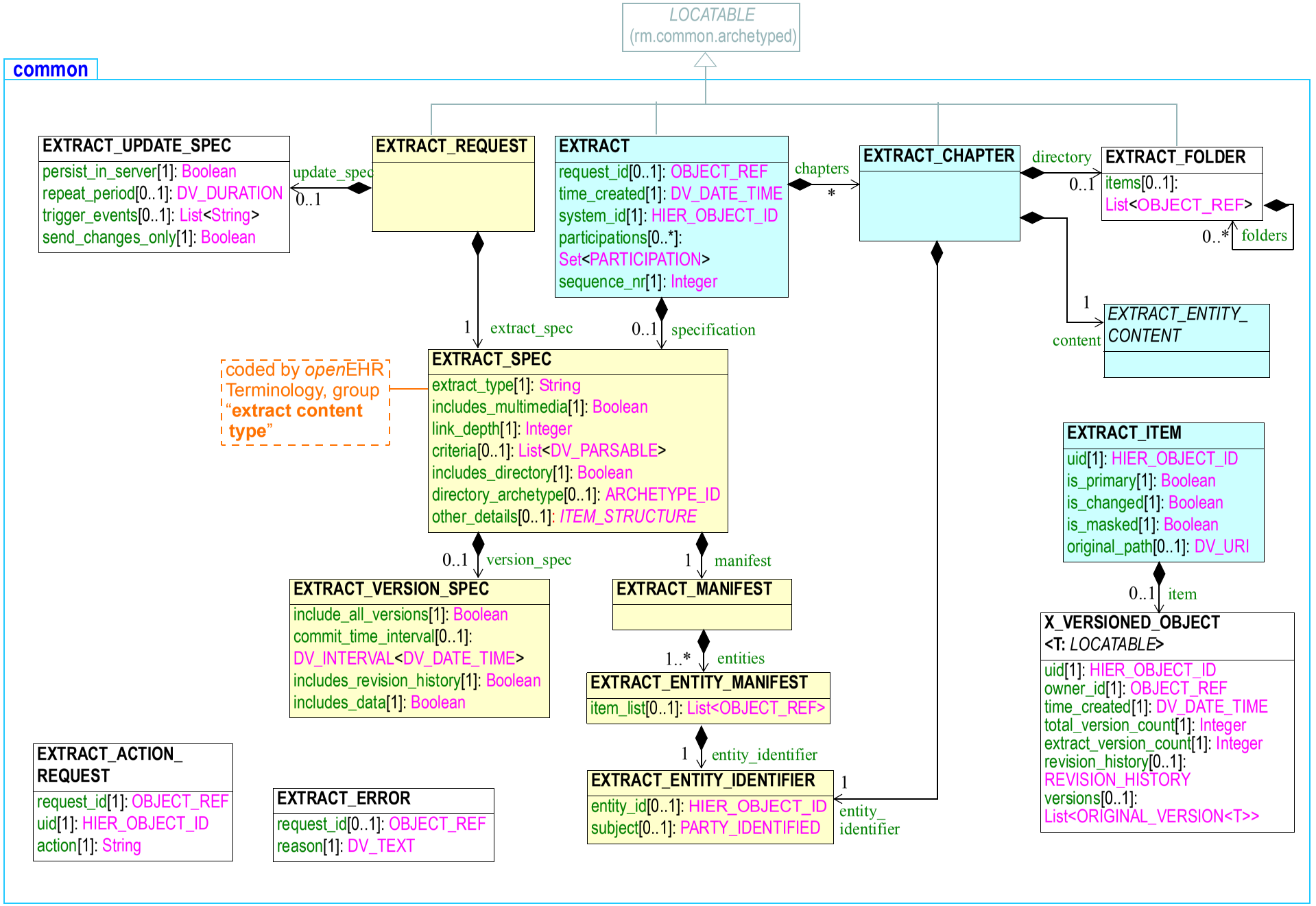


FIGURE 6 rm.extract.common Package

by a hospital since the last such update. In these cases, the manifest will identify a number of entities, each of which will be allocated a separate chapter in the resulting Extract.

The *item_list* of each entity manifest identifies individual items using OBJECT_REF instances each containing the HIER_OBJECT_ID identifier of a particular top-level object such as a Composition, or Party. This mechanism for identifying contents of an Extract is only expected to be used when a specific identifier is known, rather than when items corresponding to filtering criteria are requested. The latter are specified using the *criteria* attribute.

Retrieval Criteria

The *criteria* attribute defines in the form of generic queries (i.e. queries that apply sensibly to any record) which items are to be retrieved from each entity's record. Each query is expressed as a DV_PARSABLE instance, enabling any formalism to be used. Some *open*EHR query formalisms are already in development, and generally include a path-based approach to identifying items. Query expressions use variables such as $ehr to mean the current EHR from the *manifest* list. Queries may be as simple as the following:

- SELECT * FROM $ehr - get all items in the record
- SELECT /ehr_access FROM $ehr - retrieve the EHR_ACCESS object in an EHR
- SELECT /ehr_status FROM $ehr - retrieve the EHR_STATUS object in an EHR

More sophisticated queries can be used to obtain items corresponding to a specific criteria, e.g.:

- SELECT .... - retrieve last 6 months' worth of blood glucose measurements
- ... - retrieve ongoing medications list
- ..... - retrieve items relating to active pregnancy
- ..... - retrieve all GP encounter notes since 12-03-2005

To Be Continued:

Version Specification

A Extract request in its simplest form has no version specification, corresponding to the assumption of "latest available version" for each matched item. However, in some situations there is a need to retrieve previous versions, or information about versions. The version specification part of the Extract request allows this to be done. The possibilities available are as follows.

include_all_versions: the whole version 'stack' of each item matched by the *manifest* and retrieval *criteria* should be returned. Note that the result of this will be all available versions from the repository in question, which is in general not the same as all versions that have ever been created, since versions in the same logical version tree may exist at other repositories due to local modifications that have not been propagated to the target repository of the Extract Request.

includes_revision_history: this flag indicates whether the *revision_history* of a VERSIONED_OBJECT is to be included in the Extract form of the version

4.2.3 Extract

The EXTRACT consists of an optional *participations* list, an optional *specification* and a set of *chapters*, each containing the content for a given entity. The *participations* attribute denotes parties that were responsible for creating the Extract. The *specification* takes the same form as the *extract_spec* of the EXTRACT_REQUEST, but in an EXTRACT instance, indicates the actual contents of the Extract, which may differ from the request (e.g. due to limited availability of versions, revision histories, or simply the fact that a requested entity has no record at the repository). Each EXTRACT_CHAPTER

includes a *content* object and an optional *directory*, which is a simple folder structuring mechanism for content held in the chapters, required by the CEN EN 13606 standard. Note that this directory structure has nothing to do with directory structures that may be retrieved from the target repository - the latter are included in the content part of an Extract chapter. The directory structure, if used, refers to content items by means of `OBJECT_REFs` referring to `EXTRACT_ITEM.uid`

The *content* object is subtyped by specific Extract models described in later sections of this specification.

4.3 Class Descriptions

4.3.1 EXTRACT_REQUEST Class

CLASS	EXTRACT_REQUEST	
Purpose	Generic model of a Request for an Extract, containing an Extract specification.	
Inherit	`LOCATABLE`	
Attributes	**Signature**	**Meaning**
1..1	**extract_spec**: `EXTRACT_SPEC`	Specification details of the request.
0..1	**update_spec**: `EXTRACT_UPDATE_SPEC`	Update details of the request.
1..1 (redefined)	**uid**: `HIER_OBJECT_ID`	Identifier of this Request, generated by requestor.
Invariants	***Extract_spec_valid***: extract_spec /= Void ***Uid_exists***: uid /= Void	

4.3.2 EXTRACT_ACTION_REQUEST Class

CLASS	EXTRACT_ACTION_REQUEST	
Purpose	Generic model of a Request for an Extract, containing an Extract specification.	
Inherit	`LOCATABLE`	
Attributes	**Signature**	**Meaning**
1..1	**request_id**: `OBJECT_REF`	Identifier of previous `EXTRACT_REQUEST`.
1..1	**action**: `String` should be coded	Requested action: cancel, resend, send new
Invariants	***Request_id_valid***: request_id /= Void ***Action_valid***: action /= Void	

4.3.3 EXTRACT_SPEC Class

CLASS	EXTRACT_SPEC	
Purpose	Specification of an Extract's contents. Subtypes can be used to add details specific to the type of Extract. The specification consists of attributes specifying the directory, and two further groups of attributes in their own classes, namely a version specfication (which versions of information items are to be included) and a manifest (which entities are to be included in the extract).	
Use	Used in a request to specify an Extract, as well as to describe what is contained in an Extract.	
Attributes	**Signature**	**Meaning**
1..1	**extract_type**: `String` TB: should we code this one? I think we should...	Coded term indicating the type of content required, e.g. • "openehr-ehr" • "openehr-demographic" • "generic-emr" • "other"
1..1	**includes_multimedia**: `Boolean`	Indicates whether inline instances of `DV_MULTIMEDIA` in the source data are included or not.
1..1	**link_depth**: `Integer`	Degree of links to follow emanating from content items specified for inclusion. The kind of links to follow is dependent on the type of Extract. All items at the target end of followed links at the given depth are also included in the extract; `EXTRACT_ITEM.`*is_primary* is used to differentiate. - 0 = don't follow; - 1 = follow first degree links; - 2 = follow 2nd degree links; - .... - n = follow nth degree links
0..1	**criteria**: `List<DV_PARSABLE>`	Queries specifying the contents of this Extract.
1..1	**manifest**: `EXTRACT_MANIFEST`	Specification of entities (e.g. records) to include in the Extract.
0..1	**version_spec**: `EXTRACT_VERSION_SPEC`	Specification of which versions of information items to include in the Extract. If Void, the default is latest versions only (which is reasonable for non-versioning systems as well).

CLASS	EXTRACT_SPEC	
1..1	**includes_directory**: `Boolean`	True if the Extract includes a Folder directory.
0..1	**directory_archetype**: `ARCHETYPE_ID`	Identifier of archetype to use for local Folder structure; if Void and *includes_directory* is True, a non-archetyped directory is used.
0..1	**other_details**: *`ITEM_STRUCTURE`*	Other specification items. Archetypable.
Invariants	***Extract_type_valid***: extract_type /= Void **and then not** extract_type.is_empty ***Link_depth_valid***: link_depth >= 0 ***Manifest_valid***: manifest /= Void ***Directory_spec_valid***: directory_archetype /= Void **implies** includes_directory	

4.3.4 EXTRACT_MANIFEST Class

CLASS	EXTRACT_MANIFEST	
Purpose	Specification of the candidate entities and optionally top-level items to be included in the Extract.	
Attributes	**Signature**	**Meaning**
1..1	**entities**: `List<EXTRACT_ENTITY_MANIFEST>`	List of entity manifests uids of items included in the Extract; for *open*EHR data, these are uids identifying the version containers.
Invariants	***Entities_valid***: entities /= Void **and then** not entities.is_empty	

4.3.5 EXTRACT_ENTITY_MANIFEST Class

CLASS	EXTRACT_ENTITY_MANIFEST	
Purpose	The manifest for one entity, identifying the entity and optionally specifying top-level items to be included in the Extract. The list actually included may be modified by the *version_spec* part of the specification, and also by the *link_depth* attribute. In repeat (standing order) Requests, the final inclusion may be modified by the *send_changes_only* flag of the *update_spec*.	
Attributes	**Signature**	**Meaning**
1..1	**entity_identifier**: `EXTRACT_ENTITY_IDENTIFIER`	Identifies an entity, such as an EHR or subject (i.e. usually a person or other demographic entity).

CLASS	EXTRACT_ENTITY_MANIFEST	
0..1	**item_list**: `List<OBJECT_REF>`	List of uids of items included in the Extract; for *open*EHR data, these are uids identifying the version containers.
Invariants	***Entity_identifier_valid***: entity_identifier /= Void ***Item_list_valid***: item_list /= Void **implies not** item_list.is_empty	

4.3.6 EXTRACT_ENTITY_IDENTIFIER Class

CLASS	EXTRACT_ENTITY_IDENTIFIER	
Purpose	Identifier for a single demographic entity or record thereof. Because identification is poorly standardised and also heavily dependent on existing systems, this class provides two possibilities for identification: an id for the record, or an id for the demographic entity. These are not always 1:1 anyway, due to errors that occur in systems causing duplicate records for a given entity.	
Attributes	Signature	Meaning
0..1	**entity_id**: `PARTY_IDENTIFIED`	Identifies a demographic entity for which there is a record.
0..1	**record_id**: `HIER_OBJECT_ID`	Identifies a record for a demographic entity.
Invariants	***Identifier_exists***: entity_id /= Void or record_id /= Void	

4.3.7 EXTRACT_VERSION_SPEC Class

CLASS	EXTRACT_VERSION_SPEC	
Purpose	Specification of what versions should be included in an Extract. By default, only latest versions are included in the Extract, in which case this part of the Extract specification is not needed at all. The attributes *include_all_versions* and *commit_time_interval* are used to modify this; the former forces all versions to be included; the latter limits the versions to be those latest versions committed in the time interval, or if *include_all_versions* is True, all versions committed in the time interval.	
Attributes	Signature	Meaning
0..1	**commit_time_interval**: `DV_INTERVAL<DV_DATE_TIME>`	Specifies commit time interval of items to source repository to include in Extract. By default, only latest versions whose commit times fall in the range are included. If *include_all_versions* is True, then the range includes all versions committed within the interval.

CLASS	EXTRACT_VERSION_SPEC	
1..1	**include_all_versions**: `Boolean`	True if all versions of each item in the Extract are included.
1..1	**includes_revision_history**: `Boolean`	True if revision histories of the items in the Extract are included. If included, it is always the full revision history.
1..1	**includes_data**: `Boolean`	True if the data of items matched by the content spec should be included. This is the default. If False, only revision history is included in serialised versions. Turning this option on in *open*EHR systems causes `X_VERSIONED_OBJECTs` to have *revision_history* set, but *versions* Void. Useful for interrogating a server without having to look at any content data. In other systems it may or may not have a sensible meaing.
Invariants	***Includes_revision_history_valid***: **not** includes_data **implies** includes_revision_history.	

4.3.8 EXTRACT_UPDATE_SPEC Class

CLASS	EXTRACT_UPDATE_SPEC	
Purpose	Specification of the how Request should be processed by server. The Request can be persisted in the server, meaning that a) it can be re-activated by the requesting system simply by indicating Request id, and b) that a changes-only pattern of Extract updates can be set up. To achieve this, the server has to remember what was send in the previous reponse. The update mode may be event-driven and periodic update or a mixture of both. The candidate items to be sent each time are the result of re-evaluating the content and versioning parts of the specification; what is actually sent is determined by the *send_changes_only* flag.	
Attributes	**Signature**	**Meaning**
1..1	**persist_in_server**: `Boolean`	If True, this Request is persisted in the server until further notice.
1..1	**send_changes_only**: `Boolean`	If True, send only items that are changed (including logical deletions) or new since last send. For persist_in_server Requests only.

CLASS	EXTRACT_UPDATE_SPEC	
0..1	**repeat_period**: DV_DURATION	Period for resending update Extracts in response to original Request.
0..1	**trigger_events**: List<String> TB: should we code this one? I think we should...	Set of Event names that will cause sending of update Extracts. Event types include: • "any_change" - any change in content items matched by content specification, e.g. new versions of persistent compositions. If the content list allows matching of any, or a wide range of archetypes, this event type will match any additions to the record. • "correction" - match error corrections only, including deletions. • "update" - match updates (i.e. new versions) of included content items.
Invariants	***Overall_validity***: repeat_period /= Void **or** trigger_events /= Void ***Trigger_events_validity***: trigger_events /= Void **implies not** trigger_events.is_empty ***Send_changes_only_validity***: send_changes_only **implies** persist_in_server	

4.3.9 EXTRACT Class

CLASS	EXTRACT	
Purpose	Generic model of an Extract of some information from a repository; the generic parameters select which kind of specification and content the Extract carries.	
Inherit	LOCATABLE	
Attributes	**Signature**	**Meaning**
1..1 (redefined)	**uid**: HIER_OBJECT_ID	Identifier of this Extract.
0..1	**request_id**: OBJECT_REF	Reference to causing Request, if any.
1..1	**sequence_nr**: Integer	Number of this Extract response in sequence of responses to Extract request identified by *request_id*. If this is the sole response, or there was no request, value is 1.
0..1	**specification**: EXTRACT_SPEC	The specification that this Extract actually conforms to; might not be identical with the specification of the corresponding request.
0..1	**participations**: Set<PARTICIATION>	Participations relevant to the creation of this Extract.

CLASS	EXTRACT	
1..1	**chapters**: `List<EXTRACT_CHAPTER>`	The content extracted and serialised from the source repository for this Extract.
Invariants	***Uid_exists***: uid /= Void ***Request_id_valid***: request_id /= Void **implies** request_id.type.is_equal("EXTRACT_REQUEST") ***Participations_valid***: participations /= Void **implies not** participations.is_empty ***Specification_valid***: specification /= Void ***Sequence_nr_valid***: sequence_nr >= 1 ***Chapters_valid***: chapters /= Void	

4.3.10 EXTRACT_CHAPTER Class

CLASS	EXTRACT_CHAPTER	
Purpose	One content chapter of an Extract; contains information relating to only one entity.	
Inherit	`LOCATABLE`	
Attributes	Signature	Meaning
1..1	**entity_identifier**: `EXTRACT_ENTITY_IDENTIFIER`	Reference to causing Request, if any.
1..1	**content**: *`EXTRACT_ENTITY_CONTENT`*	The information content of this chapter. Void if the requested entity does not exist in the repository.
0..1	**directory**: `EXTRACT_FOLDER`	Optional Folder structure for this Extract.
Invariants	***Entity_identifier_valid***: entity_identifier /= Void ***Content_valid***: content /= Void ***Directory_valid***: directory /= Void **implies** directory.is_archetype_root	

4.3.11 EXTRACT_FOLDER Class

CLASS	EXTRACT_FOLDER	
Purpose	Folder in local Folder structure in an Extract. Empty Folders are allowed.	
Inherit	`LOCATABLE`	
Attributes	Signature	Meaning
0..1	**folders**: `List<EXTRACT_FOLDER>`	Sub-folders of this folder, including distinct Folder trees, which may be separately archetyped.

CLASS	EXTRACT_FOLDER	
0..1	**items**: `List<OBJECT_REF>`	List of references to `EXTRACT_ITEMs` in this Extract.
Invariants	***Folders_valid***: folders /= Void **implies not** folders.is_empty ***Items_valid***: items /= Void **implies not** items.is_empty	

4.3.12 EXTRACT_ENTITY_CONTENT Class

CLASS	*EXTRACT_ENTITY_CONTENT (abstract)*	
Purpose	Container of extracted and serialised content. Intended to be subtyped into e.g. `EHR_EXTRACT_CONTENT` etc.	
Attributes	Signature	Meaning
Invariants		

4.3.13 EXTRACT_ITEM Class

CLASS	EXTRACT_ITEM	
Purpose	Wrapper for one content item in Extract, containing various meta-data. Indicates whether it was part of the primary set and what its original path was.	
Attributes	Signature	Meaning
1..1	**uid**: `HIER_OBJECT_ID`	Unique id of this Extract item; used for all referencing from elsewhere in Extract, including from Extract Folders.
1..1	**is_primary**: `Boolean`	True if the content item carried in this container was part of the primary set for the Extract , i.e. not added due to link-following.
1..1	**is_changed**: `Boolean` TB: I added this one..useful?	True if the content item carried in this container is any kind of change since last send, in repeat sending situations.
0..1	**original_path**: `DV_URI` Is this a useful attribute? The item will already have its HIER_OBJECT_ID in the ORIGINAL_VERSION.	The original path of the item in the source repository, used for matching items in the receiver's repository.
1..1	**is_masked**: `Boolean`	True if the content of this item has not been included due to insufficient access rights of requestor.

CLASS	EXTRACT_ITEM	
0..1	**item**: X_VERSIONED_OBJECT <LOCATABLE>	Information item.
Invariants	***Uid_valid***: uid /= Void	

4.3.14 X_VERSIONED_OBJECT Class

CLASS	X_VERSIONED_OBJECT <T: LOCATABLE>	
Purpose	Variety of Extract content that consists is a sharable data-oriented version of VERSIONED_OBJECT<T>.	
Attributes	Signature	Meaning
1..1	**uid**: HIER_OBJECT_ID	Uid of original VERSIONED_OBJECT.
1..1	**owner_id**: LOCATABLE_REF	Owner_id from original VERSIONED_OBJECT, which identifies source EHR.
1..1	**time_created**: DV_DATE_TIME	Creation time of original VERSIONED_OBJECT.
1..1	**total_version_count**: INTEGER	Total number of versions in original VERSIONED_OBJECT at time of creation of this X_VERSIONED_OBJECT.
1..1	**extract_version_count**: INTEGER	The number of Versions in this extract for this Versioned object, i.e. the count of items in the *versions* attribute. May be 0 if only revision history is requested.
0..1	**revision_history**: REVISION_HISTORY	Optional revision history of the original VERSIONED_OBJECT. If included, it is the complete revision history.
0..1	**versions**: List <ORIGINAL_VERSION<T>>	0 or more Versions from the original VERSIONED_OBJECT, according to the Extract specification.
Invariants	***Uid_valid***: uid /= Void ***Owner_id_valid***: owner_id /= Void ***Time_created_valid***: time_created /= Void ***Total_version_count_valid***: total_version_count >= 1 ***Extract_version_count_valid***: extract_version_count >= 0 ***Versions_valid***: versions /= Void **implies not** versions.is_empty	

5 Ehr_extract Package

5.1 Overview

The rm.extract.ehr_extract package defines the EHR variant of an Extract. An EHR Request allows for the specification of EHR id and/or the subject of care, while the EHR Extract is defined in terms of a general model of Folders / Compositions / demographics / other items, following the top-level structure of the *open*EHR EHR. The contents of each of these parts of the Extract are in the form of a serialised variant of the VERSIONED_OBJECT class (rm.common package) whose versions are a copy of the ORIGINAL_VERSION objects matching the Request content specification, with the addition of a cryptographic digest. FIGURE 7 illustrates the rm.extract.ehr_extract package. FIGURE 8 illustrates the instance structure of an EHR_REQUEST and EHR_EXTRACT..

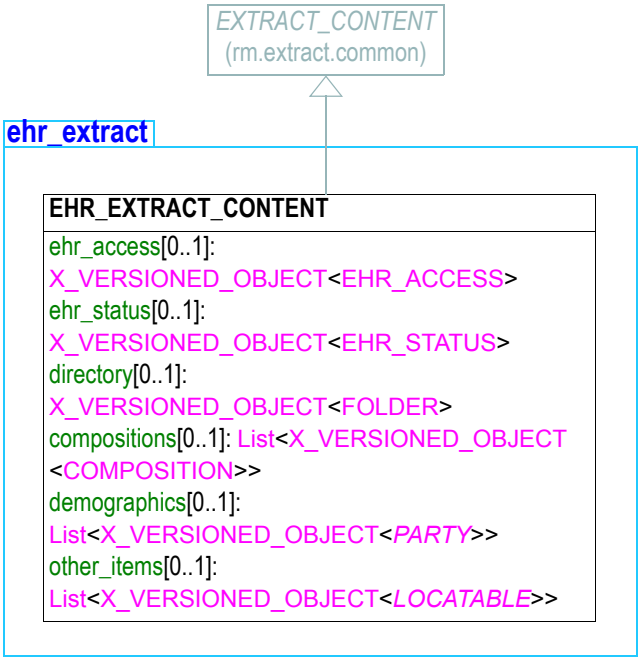


FIGURE 7 rm.extract.ehr_extract Package

5.2 Design

EHR Extract Request

The EHR_EXTRACT_REQUEST class is a specialised version of the EXTRACT_REQUEST class, whose type paremeter is bound to an EHR variant of EXTRACT_SPEC called EHR_EXTRACT_SPEC. This latter class adds only two attributes to the standard Request specification: *ehr_id* and *subject*. At least one of these must be specified. Ehr_id will be used if the requestor happens to know it, which is most likely if it is requesting updates from an EHR that was originally made as a clone of an EHR at the requestor system. However, since there is no guarantee that the identifiers will match, the subject is more likely to be specified in most cases. Another situation in which the requestor will know the EHR id will be to do with EHR split/merge remedial operations. In the Extract reply, both fields must be filled in.

IS THIS REASONABLE???

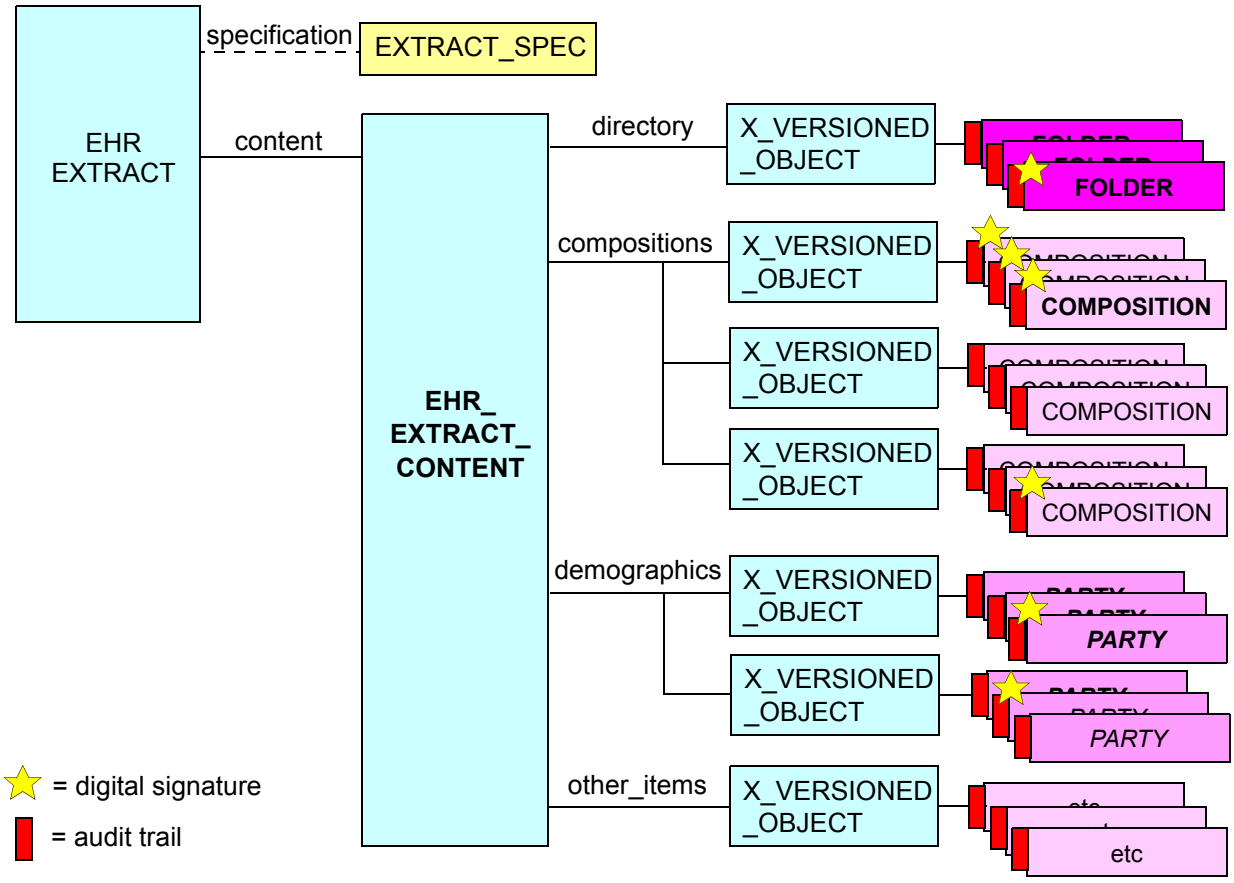


FIGURE 8 EHR Extract structure

SHOULD EHR_access be allowed? This may itself be a violation of security

should EHR_STATUS be included?

How should extract spec indicate which of directory, compositions, ehr_acces, ehr_status?

EHR Extract

`EHR_EXTRACT_CONTENT` contains items extracted from an *open*EHR EHR system according to the content and version parts of the Request specification. The items are grouped as follows:

- *directory* (copies of `EHR`.*directory* from the source system);
- *compositions*: Compositions from the source system;
- *demographics*: demographic entities from the source system demographic service, if one exists. If no demographic items are present in the Extract, the receiver must rely on identifying data in the `PARTY_IDENTFIED` objects found within `VERSIONs` and `COMPOSITIONs` within the *compositions* part of the Extract;
- *other_items*: any other items from the source system, e.g. `EHR_STATUS` object, in the case of a complete EHR move.

All of these items are included in the Extract in a serialised form of the *open*EHR `VERSIONED_OBJECT` class, called `X_VERSIONED_OBJECT`. This latter provides a standardised interoperable way to serialise all or part of `VERSIONED_OBJECTs` for lossless transmission between systems, regardless of the likely implementation differences in versioning at each end. Accordingly,

X_VERSIONED_OBJECT turns most functional properties of VERSIONED_OBJECT into data attributes. The two attributes of most interest are *revision_history*, which enables the optional inclusion of the complete revision history from the original VERSIONED_OBJECT, and *versions*, which allows any or all of the versions to be included form the original. The revision history can be requested on its own using by setting the *includes_data* flag of the version specification to False.

In most scenarios, *versions* will be included, and *revision_history* excluded. Each item in X_VERSIONED_OBJECT.*versions* consists of a wrapped copy of an ORIGINAL_VERSION from the corresponding VERSIONED_OBJECT object in the source system.

5.3 Class Descriptions

5.3.1 EHR_EXTRACT_CONTENT Class

CLASS	EHR_EXTRACT_CONTENT	
Purpose	Form of EHR Extract content containing *open*EHR serialised VERSIONED_OBJECTs.	
Inherit	EXTRACT_CONTENT	
Attributes	**Signature**	**Meaning**
0..1	**directory**: X_VERSIONED_OBJECT <FOLDER>	Folder tree from source EHR.
0..1	**compositions**: Set<X_VERSIONED_OBJECT <COMPOSITION>>	Compositions from source EHR.
0..1	**demographics**: Set<X_VERSIONED_OBJECT <*PARTY*>>	Demographic entities from source EHR.
0..1	**other_items**: Set<X_VERSIONED_OBJECT <*LOCATABLE*>>	Other items from source EHR.
Invariants	***Compositions_valid***: compositions /= Void **implies not** compositions.is_empty ***Demographics_valid***: demographics /= Void **implies not** demographics.is_empty ***Other_items_valid***: other_items /= Void **implies not** other_items.is_empty	

6 Generic_extract Package

6.1 Overview

The `rm.extract.generic_extract` package defines a generic Extract Request and Extract designed to be used by non-*open*EHR systems (including EHR/EMR systems) that want to send data to another system (which may or may not be an *open*EHR system) using *open*EHR strucures and semantics.

FIGURE 9 illustrates the `rm.extract.generic_extract` package. FIGURE 10 illustrates the instance structure of a `GENERIC_EXTRACT`..

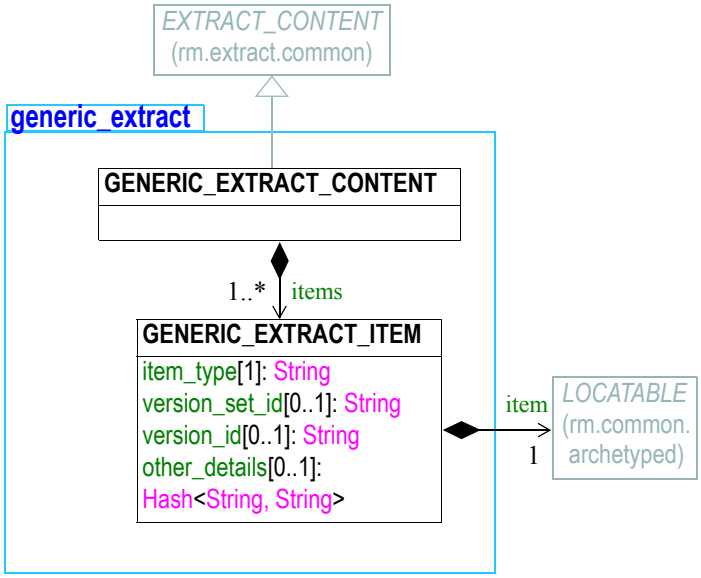


FIGURE 9 rm.extract.generic_extract Package

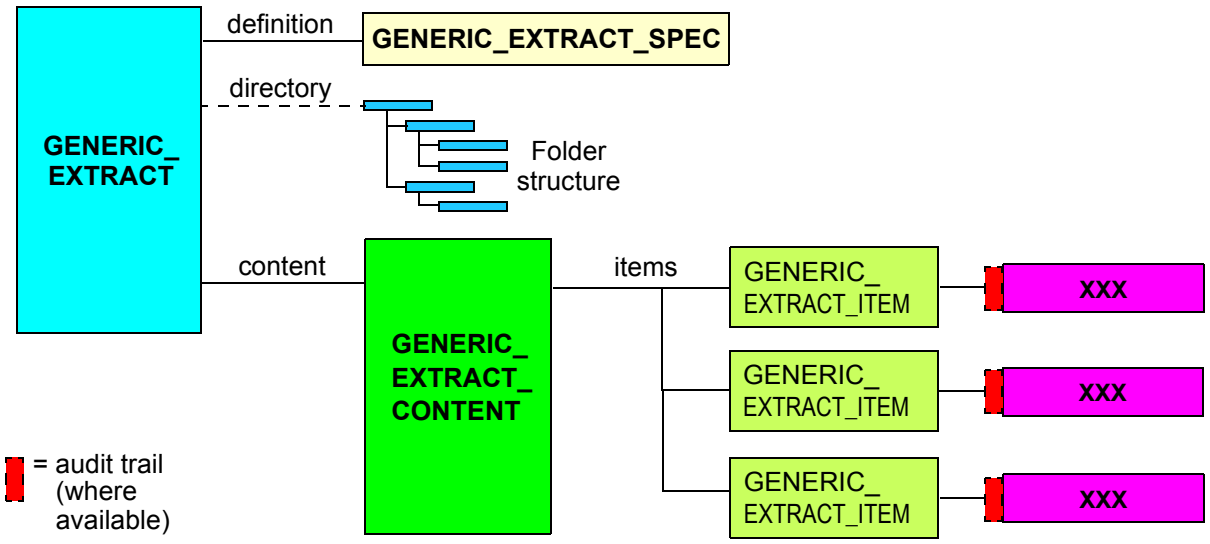


FIGURE 10 Generic Request and Extract structure

6.2 Design

Generic Extract Request

The GENERIC_EXTRACT_REQUEST class is a specialised version of the EXTRACT_REQUEST class, whose type paremeter is bound to an EHR variant of EXTRACT_SPEC called GENERIC_EXTRACT_SPEC.

This latter class adds only two attributes to the standard Request specification: *ehr_id* and *subject*. At least one of these must be specified. Ehr_id will be used if the requestor happens to know it, which is most likely if it is requesting updates from an EHR that was originally made as a clone of an EHR at the requestor system. However, since there is no guarantee that the identifiers will match, the subject is more likely to be specified in most cases. Another situation in which the requestor will know the EHR id will be to do with EHR split/merge remedial operations. In the Extract reply, both fields must be filled in.

Generic Extract

GENERIC_EXTRACT_CONTENT contains items extracted from any EHR- or EMR-like system, and converted to an *open*EHR Composition structure.

To Be Determined: should it be Compositions, or can we make it any lump of stuff, e.g. a naked Entry? I favour making it as loose as possible - that way it accommodates 13606, but also even simpler systems.

6.3 Class Descriptions

6.3.1 GENERIC_EXTRACT_CONTENT Class

CLASS	GENERIC_EXTRACT_CONTENT	
Purpose	Generic form of EHR Extract content capable of carrying any data.	
Inherit	EXTRACT_CONTENT	
Attributes	**Signature**	**Meaning**
1..1	**items**: List<GENERIC_EXTRACT_ITEM>	Items.
Invariants	***Items_valid***: items /= Void **and not** items.is_empty	

6.3.2 GENERIC_EXTRACT_ITEM Class

CLASS	GENERIC_EXTRACT_ITEM	
Purpose	Single item in generic extract.	
Inherit	EXTRACT_ITEM	
Attributes	**Signature**	**Meaning**

CLASS	GENERIC_EXTRACT_ITEM	
1..1	**item**: `LOCATABLE`	Content item.
1..1	**item_type**: `String`	Type of item.
0..1	**version_id**: `String`	Version id of this item in original system.
0..1	**version_set_id**: `String`	Version set id of this item in original system, where applicable.
0..1	**other_details**: `Hash<String, String>`	Other details about the content item.
Invariants	***Item_valid***: item /= Void ***Item_type_valid***: item_type /= Void **and then not** item_type.is_empty ***Version_id_valid***: version_id /= Void **implies not** version_id.is_empty ***Version_set_id_valid***: version_set_id /= Void **implies not** version_set_id.is_empty ***Other_details_valid***: other_details /= Void **implies not** other_details.is_empty	

7 Synchronisation Extracts

7.1 Overview

FIGURE 11 illustrates an Extract variant designed for synchronising two *open*EHR systems. The specification only allows for a list of Contributions, or Contributions since a certain Contribution; it also allows the actual versions to be included or excluded. If they are excluded, you can get just Contributions on their own - i..e find out what the other system has got.

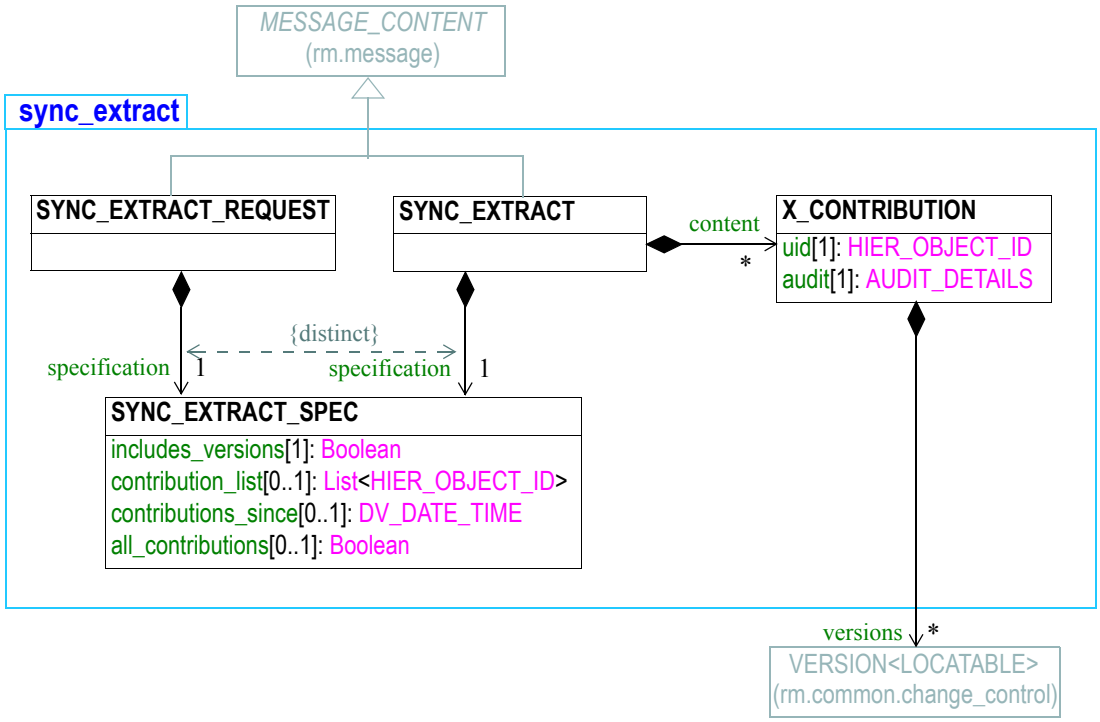


FIGURE 11 rm.extract.synchronisation_extract Package

7.2 Class Descriptions

7.2.1 SYNC_EXTRACT_REQUEST Class

CLASS	SYNC_EXTRACT_REQUEST	
Purpose	Type of request designed for synchronisation of Contributions between *open*EHR servers.	
Inherit	MESSAGE_CONTENT	
Attributes	**Signature**	**Meaning**
1..1	**specification**: SYNC_EXTRACT_SPEC	Details of specification of synchronisation request.
Invariants	***Specification_valid***: specification /= Void	

7.2.2 SYNC_EXTRACT Class

CLASS	SYNC_EXTRACT	
Purpose	Synchronisation Extract.	
Inherit	`MESSAGE_CONTENT`	
Attributes	**Signature**	**Meaning**
1..1	**specification**: `SYNC_EXTRACT_SPEC`	Details of specification of this Extract.
1..1	**content**: `Set<X_CONTRIBUTION>`	Content, in the form of a serialised Contributions.
Invariants	***Specification_valid***: specification /= Void	

7.2.3 SYNC_EXTRACT_SPEC Class

CLASS	SYNC_EXTRACT_SPEC	
Purpose	Details of specification of Extract, used in a request to specify an Extract, or in a response, to describe what is actually in the Extract.	
Attributes	**Signature**	**Meaning**
1..1	**includes_versions**: `Boolean`	True if the Versions from the Contribution are included; False if just the Contribution and its Audit are included.
0..1	**contribution_list**: `List<HIER_OJBECT_ID>`	List of Contributions to include / that are included in the Extract.
0..1	**contributions_since**: `DV_DATE_TIME`	Specify Contributions included in Extract by threshold date.
1..1	**all_contributions**: `Boolean`	True if all Contributions in the record are included.
Invariants		

7.2.4 X_CONTRIBUTION Class

CLASS	X_CONTRIBUTION	
Purpose	Serialised form of Contribution for an Extract.	
Attributes	**Signature**	**Meaning**
1..1	**uid**: `HIER_OBJECT_ID`	Uid of Contribution in source system.

CLASS	X_CONTRIBUTION	
1..1	**audit**: `AUDIT_DETAILS`	Audit of Contribution in source system.
0..1	**versions**: `List<VERSION<T>>`	Serialised Versions from Contribution in source system.
Invariants	***Uid_valid***: uid /= Void ***Audit_valid***: audit /= Void ***Versions_valid***: versions /= Void **and then not** versions.is_empty	

8 old stuff - to be rewritten

8.1 Design

The content of an EHR extract consists of the following:

- copies of a selection of Compositions, optionally within a folder structure;
- a folder structure which may include folder sub-trees from the source EHR, and/or folder trees created during the extraction process (e.g. corresponding to a discharge summary structure or similar). These folder structures may potentially be archetyped;
- copies of all entities from other services referenced from the EHR, including demographic entities (Parties) and access control entities (Access_groups);
- any extra information required for the receiver to understand the extract, potentially including terminology extracts (e.g. in the form of <key, rubric> tables).

None of the other services (particularly demographics, access control etc) in the sender's environment is assumed to be available in the receiver's environment; consequently, in general, referenced entities must be included. However, an extract can be constructed in such a way as to leave out such entities, for the case when both the receiver and sender nodes exist within the same environment and have access to the same services. In any case, it is crucial to understand that the receiver's view of the extract must include *exactly the versions of all referenced entities* as existed when the extracted information was first created. In a shared environment, the use of versioned ids (subtypes of `RM.COMMON.IDENTIFICATION.OBJECT_ID`) guarantees this.

In the case of unsolicited extracts, the structure of Folders and Compositions may be *ad hoc*, but would preferably follow archetyped models for "discharge summary", "discharge referral", "transfer of care" and other well-known documents in the health system. The structure of requested extracts is more likely to be *ad hoc*, since requests will usually be in the form of a query, such as "all compositions between date_1 and date_2", or a list of persistent compositions, such as "'current medications', 'care plan', 'therapeutic precautions'". However, it may also be structured, if archetypes are developed for repeated requests; in this case, the request will simply identify the archetype model of the extract.

Extracts are characterised by the time of creation, the parties authorising, sending and receiving, and the "initiator", i.e. who caused it - the receiver (requested) or the sender (unsolicited).

To Be Continued: describe how PARTY_IDs, other EXTERNAL_IDs work inside an EHR_EXTRACT.

9 The Message package

9.1 Requirements

In the first two EHR extract scenarios described in Requirements on page 9, extracts may be received in response to a request, or they may be unsolicited. Most transfers of care (e.g. discharge summaries and referrals) and pathology test results will generate unsolicited extracts, whereas solicited requests will usually occur due to the patient presenting him or herself in another part of the health system without an explicit transfer of care.

9.2 Design

The `message` package provides the basic abstractions for the sending and receiving of any point to point message containing a payload, of abstract type `MESSAGE`. The Message Package is illustrated in FIGURE 12.

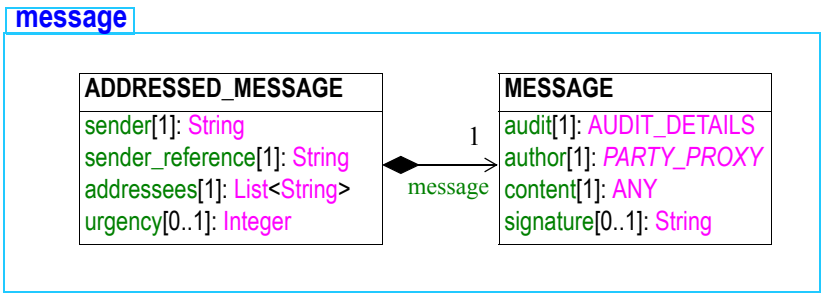


FIGURE 12 rm.message Package

A new message is required for each transmission, even if the payload was created once and is retransmitted multiple times.

Integrity and Security

The `MESSAGE` object may include a digital hash (i.e. digest or digital finderprint) of the serialised content, made for example using the SHA-1 or MD5 algorithms. The purpose of the digest is to provide an integrity check on the data. This protects against non-malicious changes to the data (e.g. due to software bugs, incorrect transaction management). Often this will be acceptable within secure, closed environments, such as a private hospital or community health network.

Protection against malicious modification can be provided by encryption.

`To Be Continued: normalised serialised expression`

9.3 Class Descriptions

9.3.1 ADDRESSED_MESSAGE Class

CLASS	ADDRESSED_MESSAGE
Purpose	The concept of a message addressed to nominated recipients.

CLASS	ADDRESSED_MESSAGE	
CEN	MESSAGE	
HL7	various message classes	
Attributes	Signature	Meaning
1..1	**sender**: String	Party sending the message.
1..1	**sender_reference**: String	Identification of message used by sender. This will be the same no matter how many times this message is sent to these recipients.
1..1	**addressees**: List<String>	Intended recipients, in the form of internet addresses.
0..1	**urgency**: Integer	Urgency with which destination should deal with message: -1 - low 0 - normal 1 - high
1..1	**message**: MESSAGE	The content of the message.
Invariants	***Sender_valid***: sender /= Void **and then not** sender.is_empty ***Addressees_valid***: addressees /= Void **and then not** addressees.is_empty ***Sender_reference_exists***: sender_reference /= Void **and then not** sender_reference.is_empty ***Message_exists***: message /= Void	

9.3.2 MESSAGE Class

CLASS	MESSAGE	
Purpose	A "message" is an authored, possibly signed, piece of content intended for one or more recipients. Since the recipient may or may not be known directly, recipients are specified in the ADDRESSED_MESSAGE class.	
Attributes	Signature	Meaning
1..1	**audit**: AUDIT_DETAILS	Details of who actually created the message-and when. This is the person who entered the data or otherwise caused the message to be created, or might be a piece of softare.
1..1	**author**: PARTY_PROXY	Party responsible for the message content, who may or may not be technically responsible for its creation (e.g. by data entry etc).

CLASS	MESSAGE	
1..1	**content**: `ANY`	Content of the message.
0..1	**signature**: `String`	Optional signature by the *author* of message content in openPGP format. The signature is created as a Hash and optional signing of the serialisation of this message object with this signature field Void.
Invariants	***Author_valid***: author /= Void ***Audit_valid***: audit /= Void ***Content_valid***: content /= Void	

10 Semantics of EHR extracts

10.1 Versioning Semantics

Although for most clinical situations, it is the latest versions of Compositions which are sent to a receiver, there are requirements for various amounts of version-related information to be included, as described in Requirements on page 9. At a minimum, Compositions always include the audit trail corresponding to the particular version which the Composition represents. In some cases, historical versions of a logical Composition are needed for some medico-legal reason. It may even be required that the receiver system wants to reconstruct a complete facsimile of the versioned object, logically identical to its form at the source (but most likely stored in a different versioning implementation).

The *open*EHR extract specification defines the simplest means of satisfying these needs, namely to include all Compositions in their whole form, including in the case where they are successive versions of a single logical Composition such as "family history", as illustrated in FIGURE 13. The main justification for this is that no assumptions should made on sender or receiver systems to do with their ability to represent or efficiently process versions. Whole Compositions can always be processed by even the simplest systems.

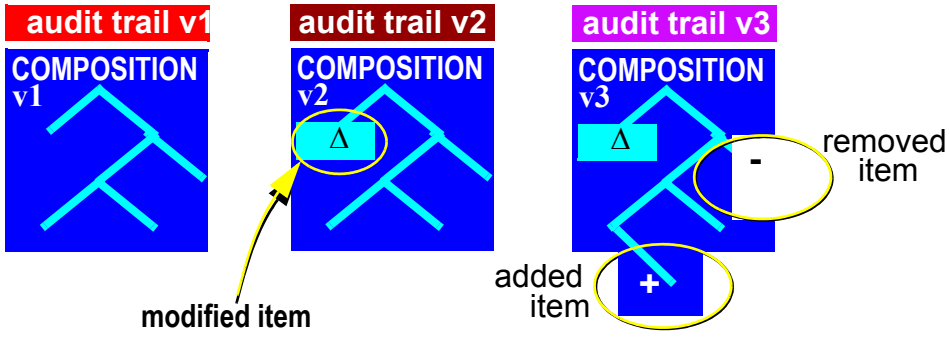


FIGURE 13 Successive Composition versions in a logical

It is assumed that any system that wants to be able to determine things such as who was responsible for changing a certain fragment of a Composition, when some part of a Composition came into being, or the differences between two particular versions of a Composition, must have version control capability locally. This usually means having some implementation of a version control model such as the one described in the *open*EHR Common Reference Model, which can do efficient versioning, differencing and so on. Supplying Compositions in their full form ensures that no assumption is made on what such an implementation might be.

The approach here is a departure from the CEN ENV 13606-4:2000 EHR Extract prestandard (although the future revision underway may change this), which defines Compositions so as to include revision history information on every node of the structure. Although it is not stated in the 13606 specification whether the "Composition" is in fact supposed to be understood as a copy of a Composition from an EHR, or as a "cumulative diff" of Composition versions in an EHR, analysis shows that only the latter can make sense because the Composition (Composition) is the unit of creation and modification, and there is logically only one audit trail for each version. Even the 100th version has associated with it only one audit trail.

This raises the question of whether a "diff" form of Compositions should be used in the *open*EHR Extract, conforming to the CEN pre-standard. The approach was not chosen for a number of reasons:

- it implies that senders can generate "diff" information structures and that receivers can process them, i.e. it makes more assumptions than necessary about
- the CEN specification appears to be in error - the sending of deleted information does not appear to be handled
- the sending of deleted information is not normally desired, and may be illegal (e.g. in Europe there are EC directives preventing the sending of statements corrected by clinicians or patients).

It is worth contemplating just how complex cumulative difference information would be. FIGURE 14 illustrates the structure generated by the accumulation of only three changes shown in the successive versions in FIGURE 13. The large numbers of changes likely in persistent Compositions will generate far more complex structures.

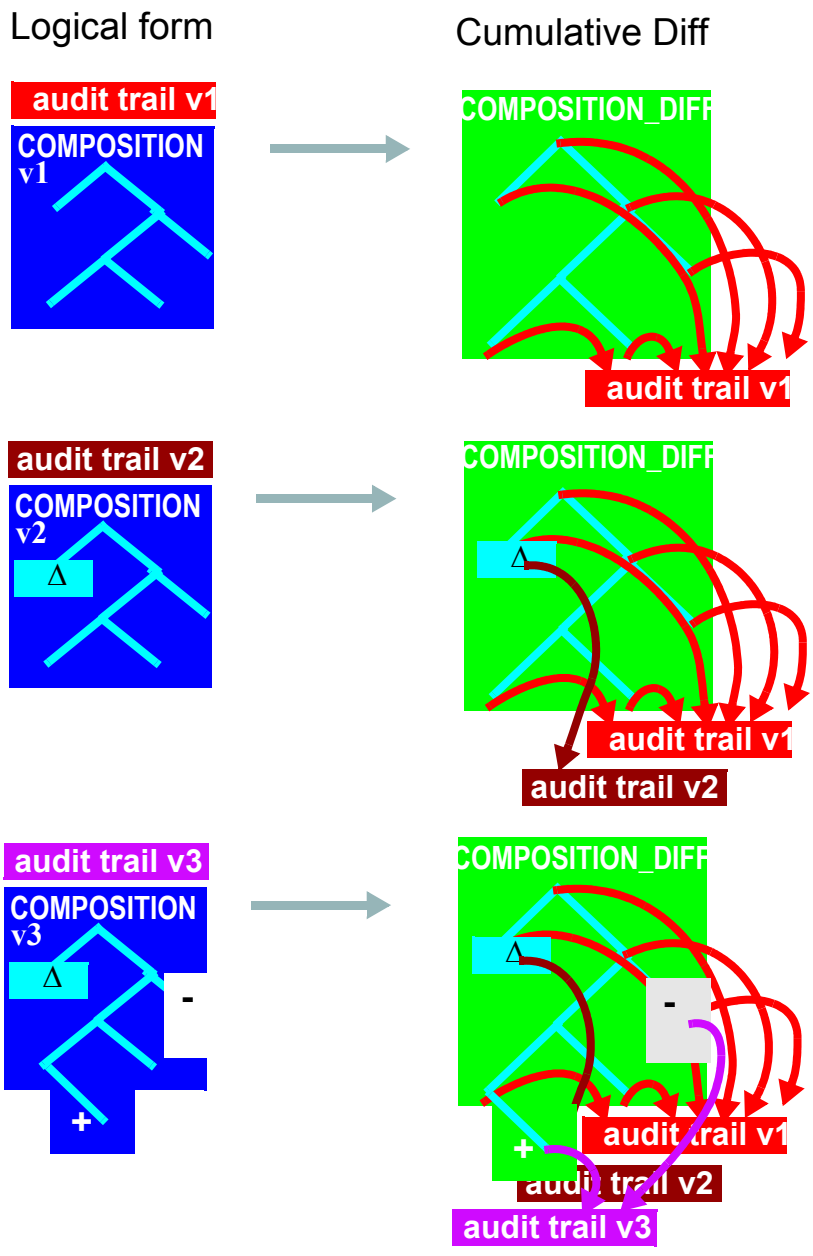


FIGURE 14 Generation of Cumulative Difference Form

In conclusion, while sending a difference form of Compositions is not out of the question in a future when most EHR nodes are capable of sophisticated version handling, it is considered too complex currently, and the controls over sending deleted information have not been sufficiently well described.

10.2 Creation Semantics

The following describes an algorithm which guarantees the correct contents of an EHR extract. The input to the algorithm is:

- the list of EHR Compositions required in the extract (the "primary" Composition set);
- optionally a folder structure in which the Compositions are to be structured in the extract;
- the *include_multimedia* flag indicating whether DV_MULTIMEDIA content is to be included inline or not;
- the *follow_links* attribute indicating to what depth DV_LINK references emanating from Compositions should be followed and the Compositions containing the link targets also included in the extract.

The algorithm is as follows.

- Create a new EHR_EXTRACT including the folder structure;
- Create a new X_DEMOGRAPHICS instance and write the demographic snapshots for each party mentioned in the EHR_EXTRACT itself into the *parties* list;
- For each Composition in the original set, do:
  - create an X_COMPOSITION, and set *is_primary*, and write the target Composition *original_path* in;
  - for each instance of OBJECT_REF encountered (e.g. PARTY_REF), obtain the target of the reference from the relevant service, and copy it to the appropriate container, i.e. *demographics*, *access_groups* tables with the key = the OBJECT_REF.*id*;
  - copy/serialise the Composition into the appropriate place in the folder structure rewriting its OBJECT_REFs so that namespace = "local"

To Be Continued: except when no parties included due to local xfer

  - for each instance of DV_MULTIMEDIA encountered, include or exclude the content referred to by the *uri* or *data* attributes, according to the *include_multimedia* flag;
  - according to the value of *follow_links*, for each instance of DV_LINK encountered (only from/to Archetyped entities):
    - follow the links recursively. For each link: create an X_COMPOSITION; set *is_primary* = False, write the path and write the target Compositions in the extract if not already there;
    - create the DV_LINK objects so that their paths refer correctly to the Compositions in the Extract;
  - TBD: do something about Access_control objects;

11 Communication Scenarios

11.1 Single Hop

To Be Continued:

11.2 Multiple Hop

To Be Continued:

11.3 Medico-legal Investigations

It is currently believed that access to prior versions will only take place for reasons of medico-legal investigations, and that this will normally occur *in situ*, i.e. at the relevant HCF, and require special legal intervention. The practical consequence of this is that only latest versions of VERSIONED_COMPOSITIONs are normally sent in EHR_EXTRACTs. However, in the case of a medico-legal investigation, earlier VERSION<COMPOSITION>s may be sent in an extract. VERSIONED_COMPOSITIONs are never sent in EHR_EXTRACTs, but might be sent in a situation where the entire EHR is changing custodianship, e.g. if the patient moves to another GP, or another country.

11.4 Transfer of Entire EHR

There are two possible ideas of transferring an "entire EHR". The first is to satisfy a request for "the latest cut" of a patient's entire EHR (or even perhaps the entire snapshot for some earlier moment in time). This scenario is simply a special case of the normal request/extract scenario in which the following conditions are true.

- The set of Compositions requested just happens to be all of the existing ones.
- There is no guarantee that all the requested compositions will be incorporated into the receiver's EHR for the patient in question - some may be discarded as irrelevant, or out of date.
- Both the sending and receiving EHR systems will continue to create and modifiy their EHRs according to independent processes.
- In general the sending and receiving systems' versions of any given EHR will diverge in time due to these processes - there is no a priori assumption that the two EHRs must remain synchronised.

Apart from the first one, these conditions are exactly the same as for the normal communication scenarios, and are dealt with by these scenarios.

The second scenario corresponds to a *change of custodianship* of an EHR, in which case the following is true:

- the whole EHR including all its versions, contributions, entire folder structure, all relevant demographic, terminological, access control and presumably all referenced patient-specific information (such as images, executing guidelines etc) stored on departmental systems is transferred to a new place;
- the old EHR is then decommissioned, archived, and possibly removed from the online system;

- the EHR in its new location (and the patient) become the responsibility of a new health care facility and/or information custodians, and is subject to what ever information governance is in place in the new location;
- all medico-legal responsibility passes to the new custodian, requiring that all previous versions in time be retained.

As far as is known, there is no solid experience showing what the generally accepted requirements for transfer-of-custodianship are. It is currently thought that the exact definition of what needs to be transferred in the second scenario could be complex and dependent upon local or regional particularities.

Further, it is thought that at the technical level, transfer of "entire EHRs" might well be accomplished by a variety of means in any particular circumstance, such as:

- physical movement of computer system;
- physical movement of binary or database files of some kind;
- low-level dump of EHR at origin and restore of EHR at receiver, assuming same/compatible database systems;
- use of binary transfer protocols e.g. CORBA, .Net etc.

At this point, it is hard to show that the operation to re-establish an EHR in its entirety in another place can be described by a clear and generally accepted set of requirements which could be formally modelled. Consequently, the specifications provided here do not claim to satisfy any particular scenario of this kind, although it is conceivable that they could be used to enact it in some particular situations, depending on the needs. Further work is required to determine what additional features might be needed in the proposed models to satisfy the EHR transfer scenarios in different countries, jurisdictions etc.

To Be Continued:

END OF DOCUMENT