



Knowledge Artefact Identification

<i>Editors:</i> T Beale ^a		
<i>Revision:</i> 0.1	<i>Pages:</i> 21	<i>Date of issue:</i> 15 May 2009
<i>Status:</i> DEVELOPMENT		

a. Ocean Informatics

Keywords: EHR, health records, repository, governance

© 2009 The *openEHR* Foundation

The *openEHR* Foundation is an independent, non-profit community, facilitating the 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

© Copyright openEHR Foundation 2001 - 2009
All Rights Reserved

1. This document is protected by copyright and/or database right throughout the world and is owned by the openEHR Foundation.
2. You may read and print the document for private, non-commercial use.
3. You may use this document (in whole or in part) for the purposes of making presentations and education, so long as such purposes are non-commercial and are designed to comment on, further the goals of, or inform third parties about, openEHR.
4. You must not alter, modify, add to or delete anything from the document you use (except as is permitted in paragraphs 2 and 3 above).
5. You shall, in any use of this document, include an acknowledgement in the form: "© Copyright openEHR Foundation 2001-2009. All rights reserved. www.openEHR.org"
6. This document is being provided as a service to the academic community and on a non-commercial basis. Accordingly, to the fullest extent permitted under applicable law, the openEHR Foundation accepts no liability and offers no warranties in relation to the materials and documentation and their content.
7. If you wish to commercialise, license, sell, distribute, use or otherwise copy the materials and documents on this site other than as provided for in paragraphs 1 to 6 above, you must comply with the terms and conditions of the openEHR Free Commercial Use Licence, or enter into a separate written agreement with openEHR Foundation covering such activities. The terms and conditions of the openEHR Free Commercial Use Licence can be found at http://www.openehr.org/free_commercial_use.htm

Amendment Record

Issue	Details	Who	Completed
0.1	Initial Writing	T Beale	15 May 2009

1	Introduction.....	5
1.1	Purpose	5
1.2	Related Documents.....	5
1.3	Status	5
1.4	Changes to openEHR Release 1.0.2	5
2	Introduction.....	6
2.1	The Problem	6
2.2	Ontological versus Machine Identifiers.....	6
3	Identification Model	7
3.1	Overview	7
3.2	Formal Model	7
3.3	Library and Package Identification	7
3.4	Ontological Identifier	8
3.4.1	Archetype Identifier	8
3.4.2	Template Identifier	9
3.4.3	Subset Identifier	9
3.4.4	Query Identifier	10
3.5	Artefact-level Revisions	10
3.5.1	Version, Revision and Commit Numbering Rules	10
3.6	File System Artefact Identification	11
3.6.1	Source Development	12
3.6.2	Operational Artefacts	13
4	Referencing.....	14
4.1	Source Artefacts	14
4.1.1	Between Archetypes - Slots	14
4.1.2	Between Specialised Archetypes	14
4.1.3	Between Archetype Historical Name-spaces	15
4.1.4	Template to Archetype References	15
4.1.5	Templates to Templates	15
4.2	Operational Artefacts.....	15
4.3	References from Data	17
5	Artefact Authentication.....	19
5.1	Integrity Check	19
5.2	Authentication	19
5.3	Canonical Form – Archetype 'semantic view'	20

1 Introduction

1.1 Purpose

The purpose of this document is to describe an identification system for *openEHR* knowledge artefacts, i.e. archetype, template and terminology subsets. This includes such artefacts created by the *openEHR* Foundation itself, as well as any other organisation. The system can be applied to any artefact type founded on the principle of two-level modelling.

Unless otherwise stated, in this document, the term 'artefact' refers specifically to these artefact types.

1.2 Related Documents

This document is part of a framework of documents for which the core document is the following:

- Distributed Development and Governance 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://www.openehr.org/svn/specification/TRUNK/publishing/architecture/am/id_system.pdf. New versions are announced on openehr-announce@openehr.org.

1.4 Changes to *openEHR* Release 1.0.2

The changes required to *openEHR* Release 1.0.2 (ADL/AOM 1.4) to enable the globalised identification system described here include:

- augmented form of `ARCHETYPE_ID` to include organisational / package namespace, e.g.
`org.openehr.ehr::openEHR-EHR-EVALUATION.problem.v1`
- concept name section of archetype identifiers (middle part of `ARCHETYPE_ID`) now relaxed to no longer require structure based on specialisation parents, e.g. 'problem-diagnosis' can now just be 'diagnosis', or any name preferred by designers;
- addition of `commit_id` sub-section archetype description section;
- addition of `id_history` sub-section to description section, e.g.

```
id_history = <"se.skl.epj::openEHR-EHR-EVALUATION.problem.v1">
```

In addition the following features are required in forthcoming ADL/AOM 1.5 specifications:

- an optional `configuration` section is defined for operational archetypes and templates, allowing the exact list of source archetypes including version, revision and commit to be included.

2 Introduction

2.1 The Problem

The main requirement of an identification system for *openEHR* knowledge artefacts is to ensure that artefacts are correctly distinguishable both at the point of use as well as within the production environment. This entails not only defining how artefacts are identified, but how they are referenced from other artefacts and from data. Identification is needed for source as well as operational (built) artefacts. Revisions need to be handled properly, as well as qualification for global uniqueness.

2.2 Ontological versus Machine Identifiers

There are two general approaches to identification. The first is the one used in software development: the ontological, human-readable identifier approach, described in the *openEHR* Distributed Development and Governance Model. The second is the use of machine processable identifiers such as ISO Oids and GUIDs with accompanying dereferencing mechanisms. The two approaches are not mutually exclusive, but neither are they equivalent.

An ontological identification scheme supports hierarchy, multi-dimensional concept spaces, flexible versioning, and formally reflects the artefact authors' and users' understanding of the concept space being modelled. Ontological identification supports many types of computational processing. A typical ontological software artefact identifier is Within the software world, ontological style identifiers are used for both source artefacts and built components such as libraries and executables, although the details of the respective types of identifier may differ.

Machine identifiers on the other hand are not human-readable, do not directly support versioning, but do enable various useful kinds of computation. They require mapping to convert to ontological identifiers.

It is possible to define an identification scheme in which either or both ontological and machine identifiers are used. If machine identification only is used, all human artefact 'identification' is relegated to meta-data description, such as names, purpose, and so on. One problem with such schemes is that meta-data characteristics are informal, and therefore can clash – preventing any formalisation of the ontological space occupied by the artefacts. Discovery of overlaps and in fact any comparative feature of artefacts cannot be formalised, and therefore cannot be made properly computable.

In this document it is assumed that the primary identification scheme is based on ontological identifiers.

3 Identification Model

3.1 Overview

The general approach for identifying source artefacts is with an ontological identifier, prefixed by a name-space identifier if the artefact is managed within a Publishing Organisation. The general pattern of controlled artefact identifiers is as follows:

```
name_space_id '::' ontological_id
```

Different types of `ontological_id` are used for archetypes, templates and terminology subsets. The following sections describe the formal details of this identification scheme, and how it supports referencing between artefacts.

3.2 Formal Model

Both controlled and uncontrolled knowledge artefact identifiers are distinguished in *openEHR*. They are defined as follows:

```
source_artefact_id: source_controlled_id | source_uncontrolled_id
source_controlled_id: name_space_id '::' ontological_id
source_uncontrolled_id: ontological_id
name_space_id: publisher_org_id '.' library_id { '.' package_id }*
publisher_org_id: V_REVERSE_DOMAIN_NAME
library_id: V_ALPHANUMERIC_NAME
package_id: V_ALPHANUMERIC_NAME
```

```
V_ALPHANUMERIC_NAME: [a-zA-Z][a-zA-Z0-9_]+
V_REVERSE_DOMAIN_NAME:
```

All archetypes and templates should be identified with this style of identifier. Any archetype or template missing the `name_space_id` part is deemed to be an uncontrolled artefact of unknown quality. An artefact carrying a namespace identifier is assumed to be under control of the referenced publishing organisation and located within one of its Libraries. A method of authenticating this is described below. In other words:

- archetypes defined within a Library have the name-space of that library;
- uncontrolled archetypes have no name-space.

3.3 Library and Package Identification

Libraries are top-level packages, and are identified using a string name. A fully qualified library identifier is a dot-separated identifier normally commencing with the reverse domain name of the Publishing Organisation, followed by a Library identifier. Examples:

```
org.openehr.ehr - EHR archetypes library at openEHR.org
org.openehr.demographic - demographics library at openEHR.org
uk.nhs.maternity - maternity library at UK NHS
edu.nci.cancer_studies - cancer studies library at US National Cancer Institute
```

Since packages are a hierarchical construct, a qualified package name with respect to a top-level package within a Library is a dot-separated name, such as:

```
cancer_studies.melanoma
```

A fully qualified package name includes the fully qualified PO identifier, e.g.:

edu.nci.cancer_studies.melanoma - melanoma studies at US National Cancer Institute

3.4 Ontological Identifier

The existing *openEHR* `ARCHETYPE_ID` is understood as the ontological identifier of an archetype within the concept space in which it was authored. The same kind of identifier has been proposed for `TEMPLATE_ID` and would serve the same purpose for templates. For both archetypes and templates, a notion of 'version' is incorporated in the identifier, meaning that a new 'version' is a new archetype or template. Therefore, when we speak of an 'artefact', we are speaking of a 'version'. This notion of version can be thought of as a 'major version' concept, or 'variant' idea. Three types of ontological identifier are defined:

```
ontological_id: archetype_id
                | template_id
                | subset_id
                | query_id
```

3.4.1 Archetype Identifier

The archetype ontological identifier is defined by the following grammar, which is a slightly simplified version of the grammar for the `ARCHETYPE_ID` type defined in the *openEHR* Support IM specification:

```
archetype_id: qualified_rm_entity '.' domain_concept '.' version_id
qualified_rm_entity: rm_terminator '-' rm_name '-' rm_entity
rm_terminator: V_ALPHANUMERIC_NAME
rm_name: V_ALPHANUMERIC_NAME
rm_entity: V_ALPHANUMERIC_NAME
domain_concept: V_SEGMENTED_ALPHANUMERIC_NAME
version_id: 'v' V_NONZERO_DIGIT [ V_NUMBER ]

V_SEGMENTED_ALPHANUMERIC_NAME: [a-zA-Z][a-zA-Z0-9_-]+
V_NONZERO_DIGIT: [1-9]
V_NUMBER: [0-9]+
```

The field meanings are as follows:

rm_terminator: id of organisation originating the reference model on which this archetype is based;
rm_name: id of the reference model on which the archetype is based;
rm_entity: ontological level in the reference model;
domain_concept: the domain concept name, including any specialisations;
version_id: numeric version identifier.

NB: this is a simplified but backwardly compatible form of the grammar in Release 1.0.2 *openEHR*, Support IM specification.

The basic form of the structure is 2-part - an identifier for a reference model class followed by an identifier of the domain concept defined by the archetype, based on the class. The first part takes the form of a 3-part identifier, such as:

openEHR-EHR-EVALUATION

This historically has been used in *openEHR* to identify the reference model class on which an archetype is based. It includes the publisher of the reference model (e.g. "*openEHR*"), which reference model is being referred to (in case there are more than one), and finally which class. A more modern

approach might have been to use the fully qualified class identifier used in *openEHR* and in normal software, e.g.

```
org.openehr.rm.composition.content.entry.EVALUATION
```

However, this appeared unnecessarily long given that in most if not all reference models that could conceivably be used for archotyping, class names would be unique. As no problems have appeared with the short form used historically within *openEHR*, it is retained here.

The second part of the identifier is the domain concept identifier. Domain concept identifiers are natural language words or phrases, often shortened to a mnemonic form, e.g. “bp_measurement”. For specialised archetypes, they currently take the form of a series of such words, separated by '-' characters, where each segment corresponds to a parent archetype concept. This, the concept 'problem-diagnosis' is a 'diagnosis' specialisation of a 'problem' archetype.

If rigidly applied (as in the past), this system of concept naming creates the problem of how to identify two 'diagnosis' specialisations of two 'problem' archetypes, from different publishers, e.g. having the identifiers:

```
uk.nhs.clinical::openEHR-EHR-EVALUATION.problem.v1
```

and

```
org.openehr.clinical::openEHR-EHR-EVALUATION.problem.v1
```

If the UK NHS wanted to create 'diagnosis' specialisations of both of these, the identifier would be:

```
uk.nhs.ehr::openEHR-EHR-EVALUATION.problem-diagnosis.v1
```

for both, which is clearly not useful.

An approach that will avoid this problem is to consider the domain concept identifier as a single level identifier that does not try to reference any parent identifier, nor imply any particular level of specialisation. The two specialisations would then be named according to their purpose or contents, e.g.

```
uk.nhs.clinical::openEHR-EHR-EVALUATION.diagnosis.v1
```

and

```
org.openehr.clinical::openEHR-EHR-EVALUATION.cancer_diagnosis.v1
```

To allow for the fact that legacy specialised archetypes do in fact include the '-' style of separated domain concept identifier, the '-' character would still be allowed, but would have no semantic significance. The consequence is that for archetypes in the future, the level of specialisation cannot be determined solely from the identifier.

3.4.2 Template Identifier

Within a given development/publishing space, template ontological identifiers are defined the same way as archetype identifiers, i.e.:

```
template_id: qualified_rm_entity '.' domain_concept '.' version_id
```

3.4.3 Subset Identifier

Terminology subsets are a relatively new type of artefact, both in *openEHR* and in the ICT industry in general. An initial proposal for a subset identifier following the same precepts as for archetypes and templates described in this specification is as follows:

```
subset_id: qualified_terminology_id '.' domain_concept '.' version_id
qualified_terminology_id: terminology_originator '-' terminology_name
terminology_originator: V_REVERSE_DOMAIN_NAME
terminology_name: V_ALPHANUMERIC_NAME
```

This leads to identifiers like the following:

```
org.ihtsdo-snomed_ct.blood_phenotype.v2 -- Snomed Blood type subset
int.who-icd10.bacterial_infections.v13 -- ICD10 bacterial infections subset
```

An alternative identification approach within the SNOMED-CT terminology is likely to be a SNOMED code for each subset. However, the identification system used in *openEHR* must work for all terminologies, including those published within government or local organisations.

3.4.4 Query Identifier

There has been little experience with identification of queries as a designed artefact, mainly because queries in most systems are written in SQL and are not portable to any other system, being based on the local database structure. *openEHR* data queries, written in AQL or a similar formalism are portable across systems, and therefore do not need to be re-designed for each environment.

TBD_1: ontological id for queries

3.5 Artefact-level Revisions

Unlike software artefacts in most modern versioning systems, *openEHR* knowledge artefacts (i.e. particular 'versions' as defined above) are individually version-controlled. This is because an archetype or a template can in general make sense and be used on its own. Therefore, within a controlled Library, two extra numbers are defined for any particular copy of an archetype or template, as follows:

- revision number
- commit number

The revision number is incremented whenever a changed form of an archetype is published that contains only compatible changes according to the formal definition of 'revision' in DOC REF.

The commit number is used to track changes internally within an authoring environment, and is incremented for every change committed into the environment, no matter how small.

Neither the revision nor the commit number form part of the source artefact identifier, but they may be used in compiled artefacts, or as part of filenames on the file system.

3.5.1 Version, Revision and Commit Numbering Rules

The following rules are used to determine how to increment version, revision and commit number.

Version id rule: the version number starts at 0, and is incremented whenever an archetype is changed in a way that prevents it from being backwards compatible with data created by previous versions.

.v0 rule: all archetypes have this version on initial creation, before being accepted by the collaborative authoring environment;

revision id rule: revision number starts at 0 and is incremented whenever a backwards compatible change is made that affects the structure – by widening constraints and/or adding new nodes;

commit id rule: commit number starts at 0 and increments for any change at all to an archetype, including changes to meta-data, addition of translations and so on.

An archetype will start its life as a 'v0' artefact, and with no namespace. In this form, it can have any number of revisions and commits. It may be maintained for some time outside a Publishing Organisation, or it may be offered to a PO, where it will initially become part of an 'alpha' development area. where it will remain until its identifier and location in the package and Library structure is stabilised.

Once stable, an alpha archetype will migrate to the main ‘dev’ area, where it will be given a namespace prefix and have its version incremented to ‘v1’. At this point it could be published into the ‘release’ area, or alternatively, further development may occur before publishing. Whenever the revision is changed, due to a backwards-compatible structural change, the archetype should be re-published, enabling the community to have access to the latest form of the archetype.

During development, each change will increment the commit number. Whenever an archetype is published, the commit number is reset to 0.

NB: Revisions and commit numbers are not part of the ontological identifier of an artefact, but may be part of identifiers used for physical instances of such artefacts, e.g. files.

The following diagram illustrates the trajectory of an *openEHR* archetype through various versions, revisions and commits.

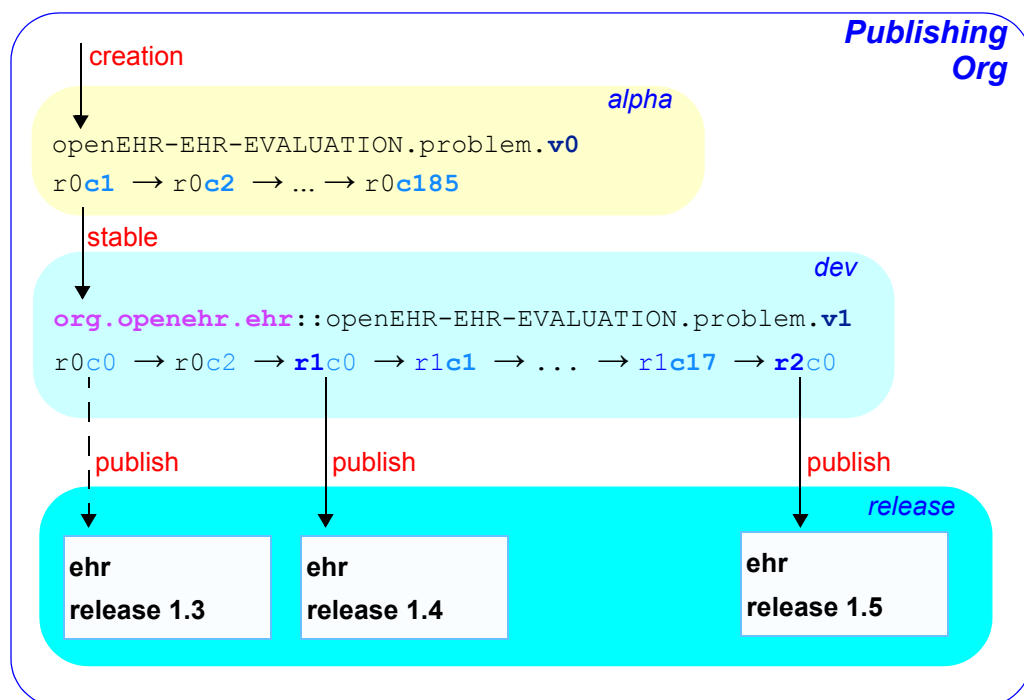


FIGURE 1 Succession of version, revision and commit changes

3.6 File System Artefact Identification

openEHR archetypes and templates can be represented on the file system as normal text files. How such files are identified depends on the intended use of the files. Two additions to the identifiers defined above are useful for file-system purposes:

- the use of *type extension*, to distinguish the various source and operational forms of the same artefact, in the usual manner of file naming;
- the use of a *directory structure* to aid human and tool use of large numbers of artefacts.

3.6.1 Source Development

Knowledge artefacts will typically be represented on the file system in source development and release areas within Publishing Organisations and also Enterprise Source Repositories, although this is not mandatory - databased approaches are equally valid.

Where the file system is used, the situation is similar to that for software, in the following ways:

- directories are used to represent namespaces and package structures;
- the current set of files represents the latest revision that has been locally created or downloaded from elsewhere. For archetypes and templates, this means that the most recent revisions and commits overwrite earlier ones.
- Publisher name-space is used as a basis for creating the top-level directory structure.

An additional structural feature stemming from the fact that archetypes and templates are based on an underlying reference model is that the package structure of the reference model can also be manifested as a directory structure. This is not mandatory, but may be helpful to tools and human users.

In a source area containing uncontrolled artefacts, the latter should be placed in a separate directory called 'uncontrolled'.

A typical directory layout of archetypes according to the above precepts is illustrated below.

```
uk.nhs                                // knowledge Publishing Org
+---clinical                          // library
|   +---org.openehr                  // RM issuer
|       +---ehr                     // RM model
|           +---composition         // RM package / class
|           +---entry               // composition.content.entry
|           |   +---action          // composition.content.entry.action
|           |   +---instruction     // etc
|           |   +---observation
|           |       +---uk.nhs.clinical~openEHR-EHR-OBSERVATION.bp_measurement.v1.adls
|           |
|           +---section
|           +---item_structure
|           +---cluster
|
|       +---demographic
|       +---party
|
+---research
    +---org.openehr
    +---ehr

org.openehr                           // knowledge Publishing Org
+---clinical                          // library
|   +---org.openehr                  // RM issuer
|       +---ehr                     // RM model
|           +---composition         // RM package / class
|           +---entry               // composition.content.entry
|           |   +---action          // composition.content.entry.action
|           |   +---observation
|           |       +---org.openehr.clinical~openEHR-EHR-OBSERVATION.heart_rate.v1.adls
```

Note that the RM package structure from *openEHR* is not religiously adhered to in this example, but used in an elided form for convenience.

In the above, the archetype filenames are a modified form of their respective fully qualified identifiers, e.g.

```
org.openehr.clinical~openEHR-EHR-OBSERVATION.heart_rate.v1.adls
```

Firstly, the '~' character replaces the '::' used in *openEHR* for namespace separation, because the Windows operating system does not allow it in filenames. Secondly, extensions are used, in this case, '.adls', meaning 'ADL source file'.

3.6.2 Operational Artefacts

In an operational context, such as a build server, there is most likely no real utility in separating artefacts into separate directories, and it may be convenient to have them in a completely flat structure, using similar filenames as above, e.g.:

```
archetypes
+--uk.nhs.clinical~openEHR-EHR-OBSERVATION.bp_measurement.v1.adl
+--org.openehr.clinical~openEHR-EHR-OBSERVATION.heart_rate.v1.adl
...
templates
+--uk.nhs.clinical.maternity~openEHR-EHR-COMPOSITION.exam.v1.opt
```

Note that here that the '.adl' and '.opt' extensions are used for operational ADL and template files respectively.

Structures such as the above would typically recur under 'tag', 'release' and 'branch' directories corresponding to specific releases and branches of a baseline of artefacts.

4 Referencing

This section describes under what circumstances, and how artefact identifiers and their component parts are used when those artefacts need to be referenced by other artefacts, data or applications.

4.1 Source Artefacts

4.1.1 Between Archetypes - Slots

Archetypes reference each other via assertions in their slot statements. These are almost always regular expressions on the allowed archetype identifiers. Such patterns can either:

- include a name-space, in which case they are limited to archetypes from that name-space;
- or include no name-space, in which case they will resolve to any matching archetypes within the local Library.

No revision or commit numbers should be used in slot references. The model of a regex pattern is any pattern based on the `artefact_id` form of identifier, i.e.:

```
[name_space_id '::'] ontological_id
TBD_2:      examples
```

4.1.2 Between Specialised Archetypes

A specialised archetype can refer to its parent in three possible ways:

- If the child is uncontrolled, with an unqualified identifier. This is always assumed to come from the current Library. If it cannot be resolved to the current library, an error is reported.
- With a qualified identifier, where the qualifier is the same as the referencing archetype. This also resolves to the latest revision of the referenced archetype, in the current Library. If not found, an error should be reported.
- With a qualified identifier where the qualifier is of Library different from that of the referencing archetype. This resolves against the latest revision of the referenced archetype in the locally available copy of the referenced Library. If not found, an error should be reported.

The form of reference used in specialised archetypes to indicate the parent is therefore the `archetype_id` form, i.e. either a controlled or uncontrolled identifier. The following figure shows a number of archetypes related by specialisation.

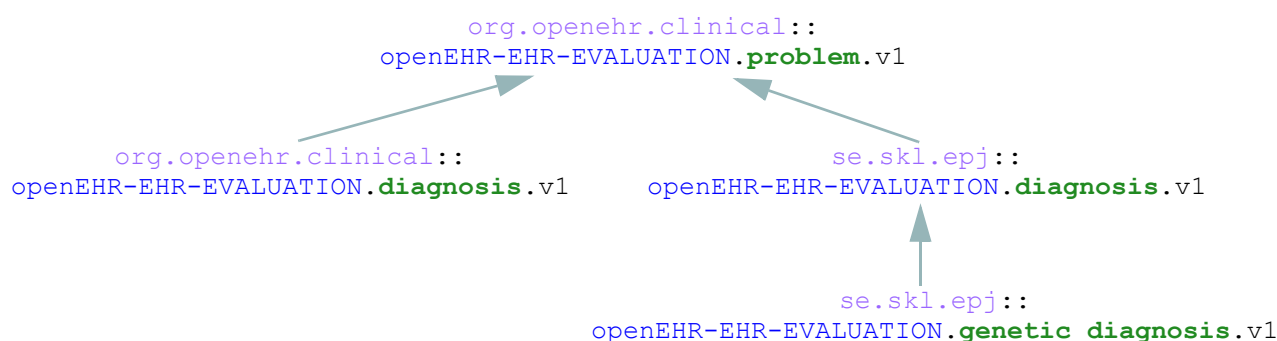


FIGURE 2 Specialisation relationships

In this figure, specialisation is indicated by a reference to the parent archetype. For example:

- `org.openehr.clinical::openEHR-EHR-EVALUATION.diagnosis.v1`
specialise
 `org.openehr.clinical::openEHR-EHR-EVALUATION.problem.v1`

One question that naturally arises to do with specialisation is what happens when the parent archetype is revised. The approach is the same as for object-oriented software: all archetypes in a given ‘check-out’ or release must always compile at any point in time to be valid. If a revised parent is introduced that invalidates a number of child archetypes in some way, revisions must be made to the children before the repository becomes valid as a whole again.

4.1.3 Between Archetype Historical Name-spaces

When an archetype is moved from its original PO (e.g. a local health authority, or a specialist peak body) to a more central authoring domain (e.g. a national library, *openEHR.org*) its name-space will be changed to the new domain, as part of the review and handover process. The archetype's semantic definition may or may not change. In order for tools to know that an archetype was not created new locally, but was moved from another PO, an explicit reference statement can be made in the archetype in the description section of an archetype as follows:

```
id_history = <"se.skl.epj::openEHR-EHR-EVALUATION.problem.v1">
TBD_3:      rules for other changes in id
TBD_4:      rules for changed level of specialisation
TBD_5:      refer to specific revision?
```

4.1.4 Template to Archetype References

As with archetype-archetype referencing, templates refer to archetypes using their identifiers. These are resolved against the current and other installed Libraries. The reference is of the same form as for archetype references, i.e. the `archetype_id`.

There appears to be no point including archetype revision information within a template source, since this creates a version management problem when multiple templates within a Library start to refer to different versions of archetypes, which are unlikely to be simultaneously available, or even mutually coherent. Instead, template references to archetypes are assumed to be resolved at compilation time, just as for archetype – archetype references.

4.1.5 Templates to Templates

Templates can refer to templates to indicate inclusion. The reference approach is the same as for template to archetype references, i.e. the use of the `template_id`.

4.2 Operational Artefacts

Operational artefacts such as operational archetypes and templates generated by the Compiler tools are built from source artefacts, including by reference resolution from within some source artefacts to others within the current configuration of the local and imported Libraries. The particular versions of reference targets are determined by the contents of the configuration, and are thus a function of version management activities, in the same way as for software development.

When an operational artefact is generated from controlled source artefacts (i.e. within a Publishing Organisation), it is possible to include the revision information from source artefacts, so that the operational form describes exactly which set of source artefacts were used to produce it. The source artefact semantic signatures can also be included. This information can be included in a configuration

section of the artefact. This would be expressed in dADL or an XML equivalent, and would list the 'configuration' of concrete artefact revisions used to generate the operational version.

The structure of a Configuration is as follows:

TBD_6: convert following to UML

```

configuration: archetype_config template_config subset_config lineage_config
    rm_release
archetype_config: { config_item }+
template_config: { config_item }*
subset_config: { config_item }*
rm_release: rm_name release_id
lineage_config: { lineage_item }*

config_item: identifier [ revision_id [ commit_id ] ] [ signature ]
lineage_item: archetype_id { archetype_id }+

signature: CHARACTER_SEQUENCE
revision_id: V_INTEGER
commit_id: V_INTEGER
release_id: V_STRING

```

An example of the configuration of an operational template in a controlled environment in dADL form is as follows:

```

configuration = <
  archetypes = <
    [1] = <
      id = <"org.openehr.ehr::openEHR-EHR-OBSERVATION.heartrate.v1">
      revision = <"3">
      commit = <"28">
      signature = <"23895yw85y0y0">
    >
    [2] = <
      id = <"se.skl.epj::openEHR-EHR-EVALUATION.genetic-diagnosis.v1">
      revision = <"2">
      commit = <"0">
      signature = <"98typrhweruhfd">
    >
    [3] = <
      id = <"org.openehr.ehr::openEHR-EHR-EVALUATION.problem.v2.4">
      revision = <"5">
      commit = <"0">
      signature = <"2rfhweiudfwieurfh">
    >
  >
  templates = <
    [1] = <
      id = <"se.skl.epj::openEHR-EHR-COMPOSITION.vital_signs.v5">
      revision = <"36">
      commit = <"0">
    >
  >
  subsets = <
    [1] = <
      id = <"org.ihtsdo.general::cardiac_diagnoses.v18">
      revision = <"1">
      commit = <"0">
    >
  >

```



```

    >
  >
  lineages = <
    ["se.skl.epj::openEHR-EHR-EVALUATION.genetic-diagnosis.v1"] = <
      "org.openehr.clinical::openEHR-EHR-EVALUATION.diagnosis.v1",
      "org.openehr.clinical::openEHR-EHR-EVALUATION.problem.v2"
    >
  >
  rm = <
    name = <"openehr">
    release = <"1.1">
  >
>

```

4.3 References from Data

There are two requirements to be considered with respect to data. Firstly, it must be possible to re-connect data with the archetypes and templates used to create them, at least in the original system (possibly not in systems receiving copies via Extracts). This means recording the archetype and template identifiers on the relevant nodes in a form that includes revision and commit numbers.

The second requirement relates to querying. It must be possible to know what archetypes, and therefore what path-set can be used for querying data. As a general principle, for a given archetype used to create data (e.g. an *openEHR* *OBSERVATION* object), the following archetypes could be used for querying:

- the same archetype, i.e. Exact same version, revision & commit;
- any previous revision of the same archetype;
- any of the specialisation parents of the archetype;
- any previous revision of any of the specialisation parents of the archetype.

For non-specialised archetypes, the allowable querying archetypes can be deduced from the archetype identifier recorded in the data. For specialised archetypes, the specialisation lineage can only be obtained from the operational form of the archetype, found in the template used to create the data. This would create a potential problem where for data imported from another site without the relevant template(s), the archetype lineage information was not available. This would prevent the query engine at the receiver system knowing how to query the data using even the more general archetypes in the lineage, that it may well have access to.

To address this situation, it may be useful to include the configuration meta-data from the operational template(s) with the data when it is transferred outside of its normal environment, e.g. in an EHR Extract.

The other possibility is to include archetype lineage information in the data itself. According to the *openEHR* RM, the archetype identifier for an archetyped subtree of data is recorded in the *LOCATABLE.archetype_details.archetype_id* attribute. This could be modified in the case of specialised archetypes to allow lineage information to be stored. The simplest form of this would be as a list of operational identifiers, e.g.

```

se.skl.epj::openEHR-EHR-EVALUATION.genetic_diagnosis.v1.12,
org.openehr.ehr::openEHR-EHR-EVALUATION.diagnosis.v1.29,
org.openehr.ehr::openEHR-EHR-EVALUATION.problem.v2.4

```

In the interests of reducing the amount of identification information in the data, a shortened form of this could be constructed, based on the assumption that the reference model class is always the same. The above example could then become:

```
se.skl.epj::openEHR-EHR-EVALUATION.genetic_diagnosis.v1.12,  
org.openehr.ehr::~diagnosis.v1.29,~problem.v2.4
```

The above syntax uses the '~' character in each identifier in the list to mean 'the missing parts are taken from the corresponding element(s) of the previous identifier in the list'. The final identifier is thus shortened to '~problem.v2.4'. In this syntax, the concrete archetype used to create the data is guaranteed to appear first and in its entirety in the list.

A form of reference catering to the above requirement is as follows:

```
archetype_data_ref: archetype_id_ref { ',' archetype_id_ref }*  
archetype_id_ref: source_controlled_id '.' revision_id [ '.' commit_id ]  
| uncontrolled_id  
compressed_archetype_data_ref: -- to be defined
```

In this reference, it is assumed that revision and commit numbers are meaningless for uncontrolled artefacts.

5 Artefact Authentication

In theory, revision information should be reliable, and no two physical knowledge artefacts should exist that are either identical but have different identifiers and/or revision information, or are different but are identified as being the same. However, in practical systems, such situations can occur due to uncontrolled artefact creation, uncontrolled copying, and errors in version management.

5.1 Integrity Check

It is therefore useful to be able to determine whether two artefacts (usually purported copies or subsequent revisions) are the same or not, regardless of revision information. This can be achieved by the use of a digital hash function (e.g. SHA-1, MD5), which generates a 'fingerprint' of the artefact. Two archetypes with the same hash value must be the same – hash functions generate a different result if even a single bit is different in the input stream. However, applying such functions to the typical file representation of an archetype or template will not usually have the desired result. This is because differences in white-space and non-significant ordering, which make no difference to the semantics – will still generate different hash values. Other semantically insignificant differences include changes to meta-data values, such as descriptions, etc may have been changed (e.g. to correct spelling, improving wording), and changes or additions to translations.

As a consequence, the input to a hashing function for the purpose of generating a semantic signature of an *openEHR* knowledge artefact must be some canonical form of the original literal artefact, that is impervious to differences of the above types while retaining differences that will affect computation with such artefacts. The integrity check process is illustrated below.

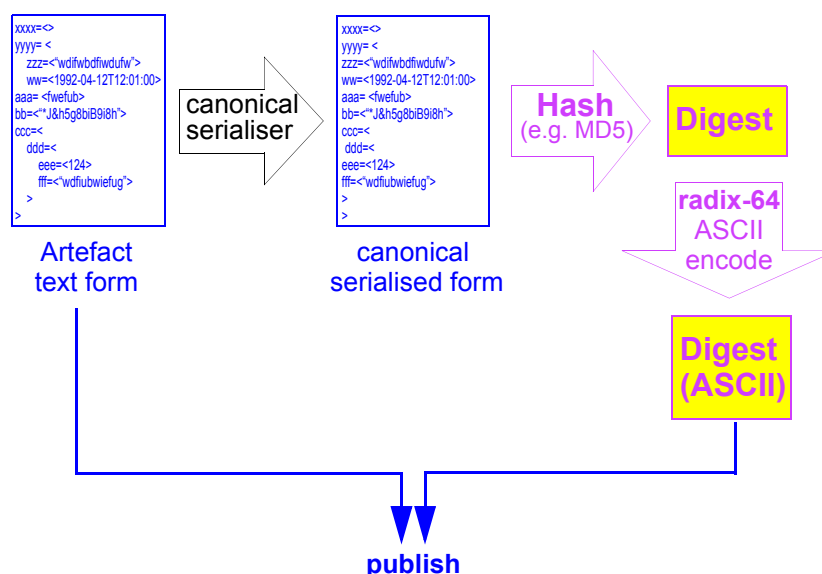


FIGURE 3 Integrity check applied to knowledge artefact

5.2 Authentication

A second need to do with validity of knowledge artefacts is establishing their authenticity, i.e. their true origin. The usual way of supporting authentication is with a digital signature. A typical scheme based on the public key infrastructure (PKI) concept is for the producer of an artefact to sign it with

their private key, and for the public key to be used by a consumer of the artefact to decrypt the signed entity.

In the case of *openEHR* knowledge artefacts, the need is to know the originating Publishing Organisation of an artefact. The PKI approach is for each PO to generate a key pair, and to provide the public key to the Central Governance Authority. Signing is then carried out using the PO private key on the hash digest already generated for an artefact. The modified process is illustrated in FIGURE 4.

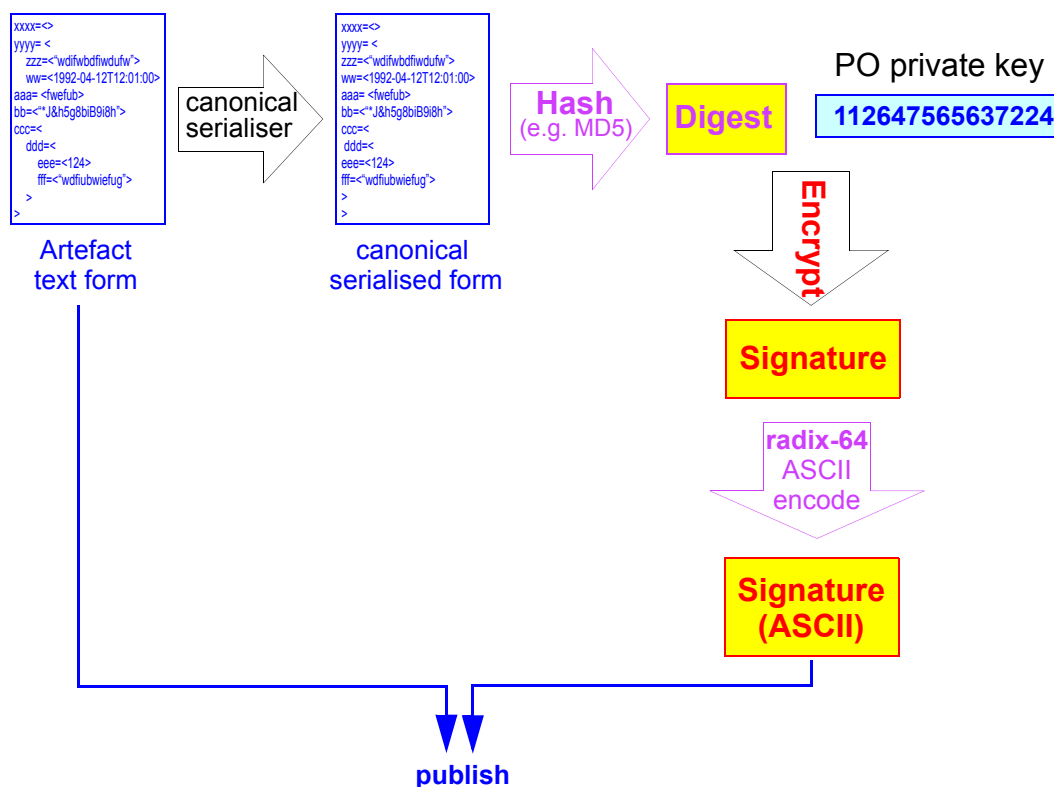


FIGURE 4 Digital signature check applied to knowledge artefact

5.3 Canonical Form – Archetype 'semantic view'

A canonical form of archetype for the purposes of hashing is based on a 'semantic view' of an archetype, analogous to the 'interface class' idea in software development. The semantic view is created from a specific serialisation of the abstract syntax tree (AST) form of an archetype, which is its computable form. The full AST form is in fact defined by the *openEHR* AOM, but this contains all textual meta-data from the description, ontology and other sections of the archetype. The semantic form of this model, suitable for generating a normalised serialisation for hashing has the following reduced form:

- the identifier;
- specialisation identifier, where present;
- concept code;
- definition section (comments stripped).

These objects would be represented in the same form as defined by the AOM. A suitable serialisation is the dADL syntax form. XML forms could be used, but they depend on which schema variant is in use, and there is no single normative *openEHR* XML-schema for the AOM.

END OF DOCUMENT