openEHR

Release 1.0.2

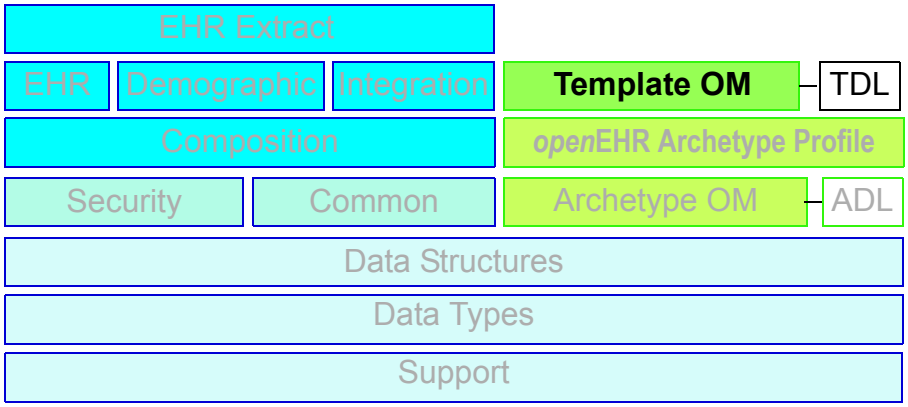
The *open*EHR Archetype Model

*open*EHR Templates

Editors: T Beale [a]		
Revision: 1.0	*Pages:* 28	*Date of issue:* 01 Feb 2010
Status: DEVELOPMENT		

a. Ocean Informatics

Keywords: EHR, ADL, health records, archetypes, templates



The *open*EHR 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

Amendment Record

Issue	Details	Raiser	Completed
RELEASE 1.1 candidate			
1.0	**SPEC-178**. Add template object model to AM.	T Beale	01 Feb 2010
RELEASE 1.0.1			
0.5	Minor content modifications.	T Beale	13 Mar 2007
RELEASE 1.0			
0.5rc1	**CR-000178**. Add Template Object Model to AM. Initial Writing	T Beale	10 Nov 2005
RELEASE 0.96			

Trademarks

Microsoft is a trademark of the Microsoft Corporation

Acknowledgements

The work reported in this document was funded by Ocean Informatics and the UK National Health Service (NHS).

Table of Contents

1 Introduction

1.1 Purpose

This document describes the semantics of *open*EHR templates. Source templates are defined as technically normal archetypes, with the only differences being to do with identification rules. The ADL and AOM 1.5 models support all aspects of template definition. A very small additional model is required to formally define the *operational* template, which is the result generated from a source templates and archetypes.

The intended audience includes:

- Standards bodies producing health informatics standards;
- Software development organisations using *open*EHR;
- Academic groups using *open*EHR;
- Medical informaticians and clinicians interested in health information;
- Health data managers.

1.2 Related Documents

Prerequisite documents for reading this document include:

- The *open*EHR Architecture Overview

Related documents include:

- The *open*EHR Archetype Definition Language (ADL)
- The *open*EHR Archetype Object Model (AOM)

1.3 Nomenclature

In this document, the term ‘attribute’ denotes any stored property of a type defined in an object model, including primitive attributes and any kind of relationship such as an association or aggregation. XML ‘attributes’ are always referred to explicitly as ‘XML attributes’.

The term ‘template’ used on its own always means an *open*EHR template definition, i.e. an instance of the AOM used for templating purposes. An operational template is always denoted as such in *open*EHR.

1.4 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/tom.pdf. New versions are announced on openehr-announce@openehr.org.

Blue text indicates sections under active development.

2 Overview

2.1 Context

Templates constitute a third layer above archetypes and the reference model in the *open*EHR application architecture shown in FIGURE 1, and provide the means of defining groupings of archetype-defined data points for particular business purposes. They support bindings to terminology subsets specific to their intended use, and can be used to generate or partly generate a number of other artefact types including screen forms and message schemas.

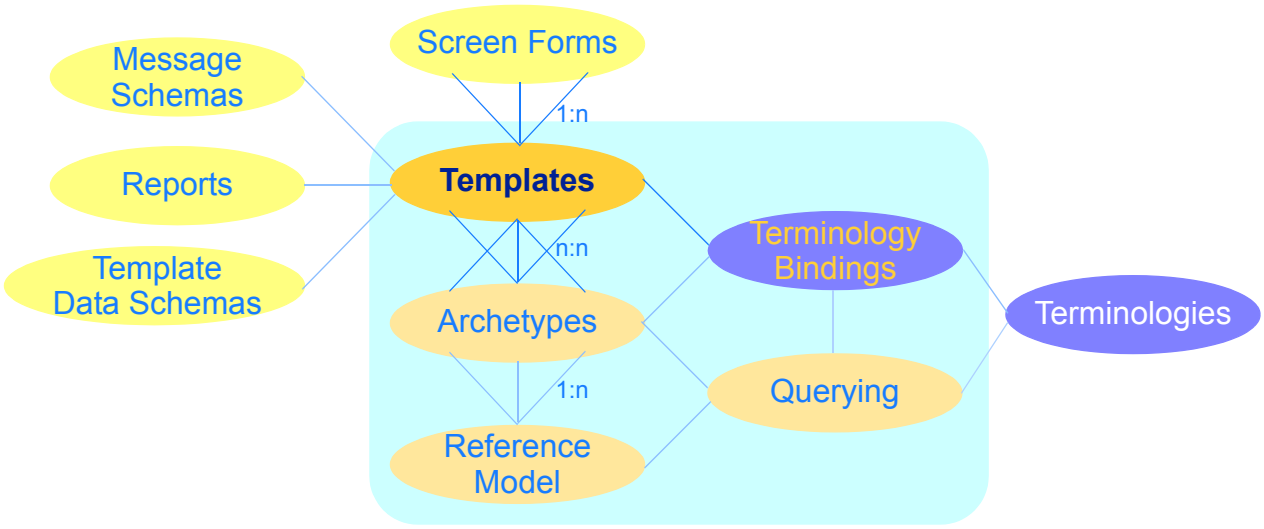


FIGURE 1 The *open*EHR Semantic Architecture

Templates are formally expressible in terms of ADL 1.5, and as instance structures, are completely defined by the AOM 1.5 specification. The operational form of a template is defined by the template package described in this specification.

TBD_1: A related artefact is the *open*EHR *palette*, which which defines the local language and terminology preferences on a per-archetype basis, also defined in this specification.

2.2 Basic Semantics

2.2.1 Purpose

An *open*EHR template is an artefact that enables the content defined in published archetypes to be used for a particular use case, i.e. business event. In health this is normally a 'health service event' such as a particular kind of encounter between a patient and a provider. Archetypes define content on the basis of topic or theme e.g. blood pressure, physical exam, report, independently of particular business events. Templates provide the way of using a particular set of archetypes, choosing a particular (often quite limited) set of nodes from each and then limiting values and/or terminology in a way specific to a particular kind of event, such as 'diabetic patient admission', 'ED discharge' and so on. Such events in an ICT environment nearly always have a logical 'form' (which may have one or more 'pages' or subforms and some workflow logic) associated with them; as a result, an *open*EHR tem-

plate is often a direct precursor to a form in the presentation layer of application software. Templates are the technical means of using archetypes in runtime systems.

This job of a template is as follows:

- aggregate archetypes into larger structures by indicating which archetypes should fill the slots of higher-level archetypes;
- remove unnecessary archetype elements ('data points');
- narrow existing constraints on archetypes or on the reference model (i.e. on parts of the RM not yet constrained by the archetypes referenced in the template);
- set default values.

A template may aggregate any number of archetypes, but choose very few data points from each, thus having the effect of defining a small data set from a very large number of data points defined in the original archetypes.

2.2.2 Formal Definition

Formally, an *open*EHR template source is simply a collection of one or more specialised archetypes, managed in a template repository. We refer to these as 't-archetypes' in order to distinguish them from locally authored archetypes or externally published archetypes. The template root is thus just a specialisation of some published archetype, as are any further constraints on referenced archetypes. The distinguishing characteristics are:

- templates are managed in a workspace separate to that of published archetypes, normally local to the organisation defining the templates;
- template identifiers, while legal archetype identifiers, follow certain patterns that make them easier to identify as templates, including by tools;
- templates tend to exclusively use the slot-filling semantic, even though it is technically valid in any archetype;
- because of slot-filling, a single logical 'template' is in fact usually made up of multiple locally specialised archetypes - it is only at the point of operational template generation that a template becomes a single artefact;
- default values, while technically possible in published archetypes are typically only stated in templates.

TBD_2: whether templates can add extra nodes like archetypes can

When a template is successfully compiled, it produces an *operational template* which is just a flat-form archetype, containing the multiple ontologies of all the constituent archetypes rather than just its own.

The fact that templates a just a kind of archetype ensure that they cannot change the semantics of published archetypes. Accordingly, *all data created due to the use of templates are guaranteed to conform to the referenced archetypes and the underlying reference model*.

2.3 Computational Environment

FIGURE 2 shows how a logical template, consisting of local archetypes, and referencing published archetypes, is used to create an operational template. The template refers to one or more archetypes and usually imposes further constraints (contained in component archetypes). A template parser converts a template into an in-memory object form described by the Template Object Model. The document form template may be expressed in TDL, or in any XML or other equivalent derived from the

Template Object Model. An operational template builder generates an in-memory Operational Template which is a standalone artefact (contains all relevant parts of its template, referenced archetypes and terminology) that can be used to generate other computational artefacts including screen forms. The builder takes a palette as an input, which has the effect of removing other languages and terminologies when generating the operational template, ensuring that each resulting artefact only contains what is needed for local use.

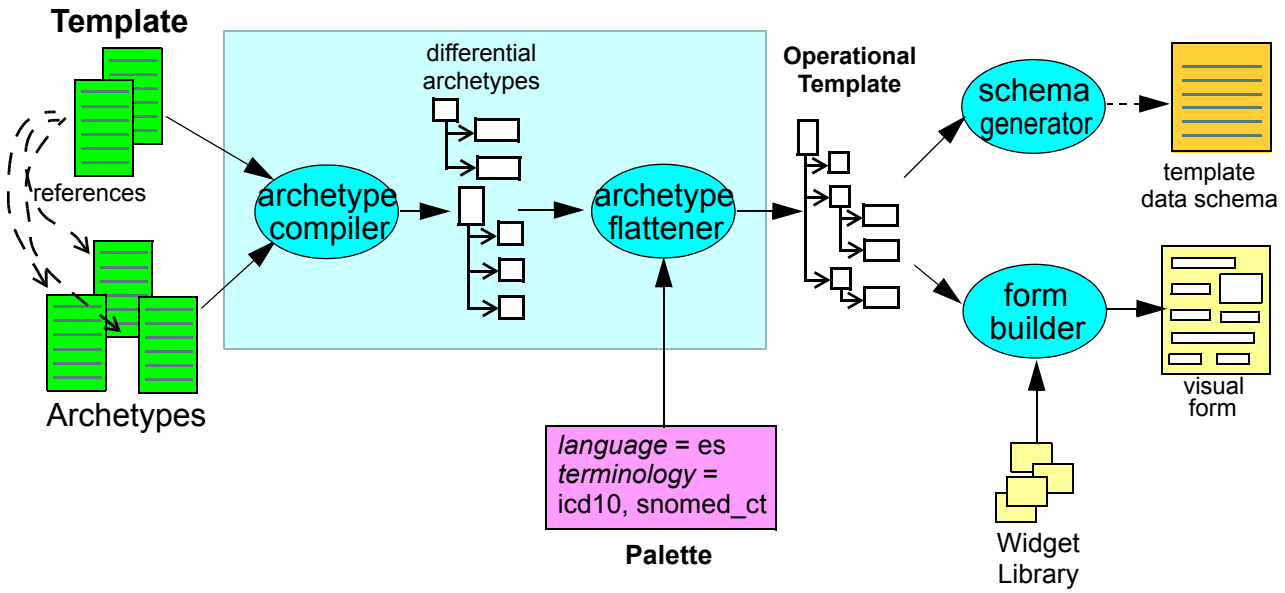


FIGURE 2 Template Tool Chain

3 Requirements

3.1 Overview

Templates need to be able to perform four main functions as follows.

Aggregation: the first is to combine archetypes into larger structures by filling archetype slots with archetypes or previously defined templates.

Element choice: the second is to choose which parts of the chosen archetypes should remain in the final structure, by removing unneeded elements, and indeed whole parts of structures specified in an archetype or the underlying reference model (where not constrained by the archetypes being used)

The third is to optionally add further constraints to the chosen archetypes or reference model, specific to the needs of the template. This may include narrowing of value ranges, and refined constraints on terminology.

The fourth function is to define default values of leaf or near-leaf structures. Default values will usually be used to pre-populate fields on a screen form, or within a template-based message.

A further general requirement on templates is of a documentary nature: how are they described, and how are they linked to existing resources to do with defining data, records and user interfaces.

The following sections describe the requirements for *open*EHR templates. Most of the requirements described here have been determined by real world use of prototype models of *open*EHR templates in health.

3.2 Identification

A globally unique identification system is required such that templates can be authored locally without reference to a central repository or identification authority, and yet still be distinguished if they are shared, or if data created by them are merged.

A human-readable structural identifier is also needed so that templates can be referred to by human authors, and template references within templates can be understood by humans.

A single kind of identifier may be used to satisfy the above purposes, or multiple identifiers may be needed.

Template versions must be included in the identification system in such a way that multiple versions of a given template can be distinguished. The versioning system should support at least 3-part version identification.

3.3 Documentary Requirements

3.3.1 Descriptive Meta-data

A similar set of meta-data as supported by archetypes should also be supported by templates, including items such as:

- purpose;
- use;
- misuse;

- audit details of creation and modification;
- references to resources used in the authoring process;
- copyright status.

Descriptive meta-data should be limited to items that directly help explain the template, rather than cross-references to every possible related item in an enterprise, which should be dealt with by other means.

3.3.2 Author Annotations

It should be possible for a template author via the use of a tool to create annotations per node of the template definition structure. Note that such annotations only apply to nodes mentioned in the template, i.e. nodes further constrained in the template. At each node, it should be possible to create one or more annotations, each with a unique tag with respect to other annotations on the same node. Annotation content is required to be normal text.

Annotations should be modifiable over time by authors. It is expected that many annotations will be meaningful only during the authoring period, but not at the time of release. A way is needed to easily remove or ignore some or all node-level annotations.

3.3.3 Institutional Meta-data and Links

There is a need to be able to create links between new artefacts like archetypes and templates, and institutional or external resources, typically documentary or design artefacts about the same topic expressed in natural language or older technologies. Within the template building environment the need is to be able to assert and persist links between such resources and templates, or parts of templates, which requires a means of referencing individual nodes in a template. It is not expected that templates themselves need to support such linking internally, rather they would be represented in an artefact external to templates.

3.4 User Interface Requirements

TBD_3: template to 'form' relationship

TBD_4: visibility of nodes on a form

3.5 Semantic Requirements

3.5.1 Relationship to Archetypes

A template is required to constrain one 'root' archetype, including any constraint redefinition and slot-filling possible within the root archetype.

3.5.2 Slot Filling

The most basic requirement is to be able to define larger structures composed from archetypes, by choosing filler archetypes for the slots defined in 'higher' (in the aggregation sense) archetypes. There are three possibilities that have to be supported:

- completely specifying the fillers for a slot;
- specifying some fillers for a slot, where remaining slot fillers may be decided at runtime;
- leaving the slot as defined by the archetype, leaving all slot fillers to be decided at runtime.

In addition, existing templates should be allowed to be used in a slot in place of an archetype. Experience with prototype templates in *open*EHR has shown the need to be able to define a template for a subtree of content that will always be constrained in the same way in all top-level objects in which it is used in the institution. Rather than redefine the same constraints in each top-level template, it is clearly preferable to be able to reuse a single definition of such content.

Typical examples are the Oximetry, Blood Pressure and Heartrate templates of the *open*EHR `OBSERVATION` class, which are reused in a number of `SECTION` and `COMPOSITION` templates.

Nesting needs to work in such a way that if the referenced template is changed, it is changed in all templates using it, i.e. a true reference. It also needs to function such a way that the referencing template can define additional constraints over and above what the referenced section defines.

3.5.3 Constraint Refinement

Templates should be able to refine the constraints defined in any included archetypes, or create new constraints on the parts of the reference model relevant to those archetypes, in the same was as for archetypes, with the following exception: template constraints cannot change existing archetype node identifiers (unlike specialised archetypes).

In particular, terminology constraints need to be redefinable as in specialised archetypes, i.e. any of the following:

- redefine open constraint on text to be set of coded terms;
- narrow coded term constraint to a subset;
- redefine an external subset (defined using an ac-code in an archetype) to a narrower external subset;
- redefine an external subset to an internal subset.

3.5.4 Default Values

Template authors need to be able to define default values for nodes defined in the template, or in an underlying archetype, even if the template does not change the node in question, and similarly for elements in the reference model for which there may or may not be constraints in the archetypes or template. A ‘default’ value is one which will be usually be displayed and will be used at runtime unless the user or application actively supplies something different.

Experience has shown that defaults need to be settable on primitive leaf types, i.e. String, Integer, Real, Boolean and the date/time types. These usually occur within subtypes of the *open*EHR type `DATA_VALUE` (e.g. `DV_QUANTITY` and `DV_CODED_TEXT`).

3.5.5 Conditional Constraints

In some cases it is needed to be able to state a constraint that is applied only if a condition is true. A typical situation is to make the existence of a subtree dependent on the existence or runtime values of data found in other subtrees. A condition is an expression combining variables and constants with boolean, arithmetic and relational operators, to generate a boolean result that can be used to decide whether to apply a constraint.

The following types of condition variables need to be supported:

- *external variable*: variables such as ‘today’s date’;
- *EHR variable*: variables such as ‘date of birth’, ‘gender’ from the EHR;

- *existence of another path*: the existence of a path in the data, which implies that at runtime an optional node was included;
- *value on other path*: the value of a leaf object in the data on a particular path.

Constants that can be used in condition expressions include the primitive types, and dates and times.

Conditions should include the facility for a natural language description of the condition so that the intent is recorded.

3.6 Optimisations

In some cases it is desirable to be able to pre-populate some terminology subsets, particularly those that are small, change rarely, or for situations where the relevant terminologies will not be available in the deployment environment at runtime. For large subsets, e.g. the set corresponding to Snomed-ct 'bacteria' (numbering in the 10s of thousands of terms), prepopulation is not desirable.

It should be possible to state within a template whether a particular subset of a specific terminology should be pre-populated.

TBD_5: if the subset was defined in the archetype, we really want to specify that a particular *binding* should be prepopulated, but if it was specified in the template, then we are choosing the binding directly?

For subsets that would normally have internal structure intact at runtime, the pre-populated version should also retain this structure, allowing the subset to be treated at runtime in the same way as if it had been available from an online terminology service. For example, a Snomed-CT subset that contains IS-A links should retain these when pre-populated.

Pre-population of templates is effected in operational templates. Consequently, when a terminology is updated, affected operational templates need to be regenerated. This requires a way of recording which templates need to be accessed in order to regenerate the corresponding operational template when changes occur in terminologies.

3.7 Tooling Requirements

TBD_6: Nested templates in a different colour

TBD_7: cut and paste of a template, template section

4 Template Definition and Development

4.1 Overview

Templates are defined in the form of specialised archetypes. This means that all the formal characteristics of a template are defined by the *open*EHR Archetype Object Model (AOM) and Archetype Definition Language (ADL) specifications. However, the mode of use of the AOM and ADL are slightly different from the typical archetype in two ways. Firstly, the following formal features are used in templates but not generally archetypes:

- slot-filling - note that specifying a slot-filler is achieved by specialisation, because a legal specialisation of a slot can be a redefinition of the slot with or without fillers;
- specifying 0..0 constraints to remove elements not needed from the referenced archetypes;
- setting default values.

Secondly, because a logical template generally implicates a number of archetypes - i.e. the root archetype plus component archetypes mentioned as slot-fillers - and also usually defines further constraints on the root and component archetypes, the typical template is expressed in the form of a *group of specialised archetypes*, with specific identifiers.

In order to better explain the template artefact structure, an example is shown in FIGURE 3. The archetype `org.openehr::openEHR-EHR-COMPOSITION.encounter_report.v1` is shown at the top left. This is templated by the template `uk.nhs.clinical::openEHR-EHR-COMPOSITION.t_encounter_report.v1`. The template performs the job of filling the at0004 slot in the root archetype by specialising it: the specialised version adds a filler object (the `C_ARCHETYPE_ROOT` instance) and also overrides the original `ARCHETYPE_SLOT` instance to close the slot to any further filling, either by further templating or at runtime. The filler object specifies in its *node_id* attribute the t-archetype being used to fill the slot, whose identifier is `uk.nhs.clinical::openEHR-EHR-SECTION.t_encounter_report-vital_signs_headings-0001.v1`. Note that this t-archetype is an artefact of the templating process, and has a generated identifier.

The same kind of redefinition occurs within the `SECTION` t-archetype, which is a specialisation of the archetype `org.openehr::openEHR-EHR-SECTION.vital_signs_headings.v1`. The at0001 slot node in this archetype is redefined by the `C_ARCHETYPE_ROOT` object in the t-archetype, which in the same way specifies the filler archetype as yet another t-archetype, this time `uk.nhs.clinical::openEHR-EHR-EVALUATION.t_encounter_report-problem_description-0004.v1`. Both of the component archetypes - the t-archetypes - also add other constraints of their own, typically removing unwanted items from the specialisation parent archetypes.

The logical template is thus constructed of three archetype artefacts, which we denote for convenience as:

- the ‘template’, or ‘template root’;
- two internal components, which we denote as ‘t-archetypes’.

It is not always the case that the components of a template must be internal; within the template environment, lower level reference model concepts such as `EVALUATION` may be templated in their own right. In this case, the outer template may contain both t-archetypes and other templates.

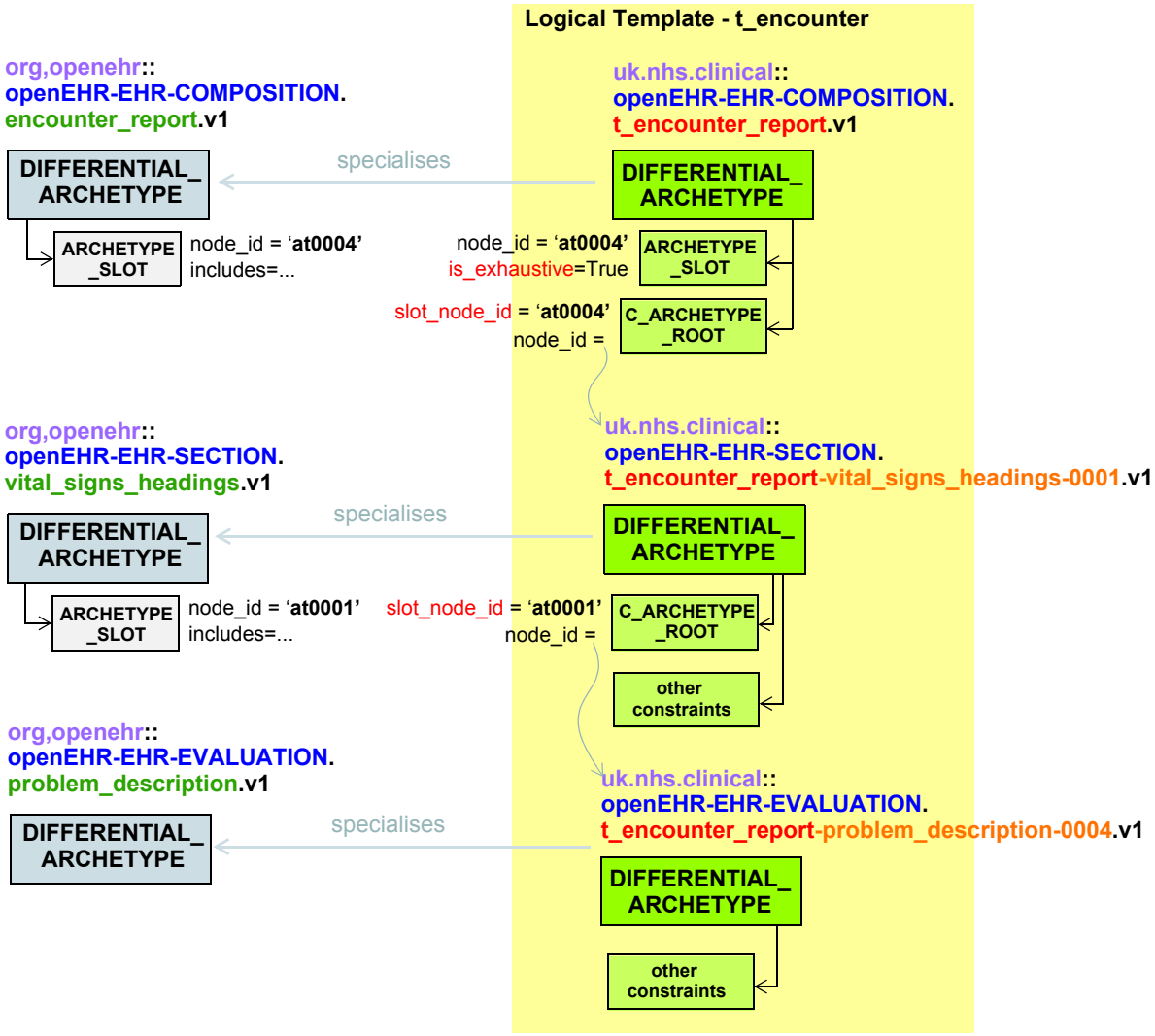


FIGURE 3 Typical template artefact structure

4.2 Design

4.2.1 Template Identification

Identification of templates is the same as for archetypes. There are two possible identifiers: the structured multi-axial form defined by the `ARCHEYTPE_ID` class, and a Guid form. These identifiers are found at the attributes `ARCHETYPE`.*archetype_id* and *uid* respectively. The structural identifier is for human readability and use in ontological classifications. Two rules, currently not formalised are strongly recommended for template identifiers. The first is to include a 't_' prefix as shown in the example. This simply enables human readers to easily spot an archetype whose design intention and use is as a template. The second is the sub-structuring of the concept_id part of the identifier of internal component t-archetypes. The recommended structure for a t-archetype ta, specialising an archetype A within a template T is as follows:

```
T:  tpl_namespace::tpl_rm_concept.tpl_domain_concept.v_tpl
A:  arch_namespace::arch_rm_concept.arch_domain_concept.v_arch
```

```
ta: tpl_namespace::arch_rm_concept.
                        tpl_domain_concept-arch_domain_concept.v_ta
```

For example:

```
Root template:
    uk.nhs.clinical::openEHR-EHR-COMPOSITION.t_encounter_report.v1
t-archetype component:
    uk.nhs.clinical::openEHR-EHR-EVALUATION.
                        t_encounter_report-problem_description-0001.v1
```

This approach defines a concept identifier which obeys the formal rule (that concept identifiers must be unique within a namespace), is human-readable, and most importantly, is tool-generatable.

Where the Guid identifier is in use, a new identifier can be immediately generated by tooling without reference to a central identification service. Each published revision of a template is assigned a new Guid, ensuring that revisions of the same logical template are distinguished. Note that the multi-axial identifier in its standard form (including only a version identifier, e.g. ‘v2’) does not do this, but does in its long form (including a full 3-part identifier ‘.v2.5.1’).

This specification uses only the multi-axial form.

4.2.2 Meta-data

Templates support the same global and node-level meta-data as archetypes, defined by the `openehr.rm.common.resource` package. The actual content will of course be oriented toward describing local template-related concerns. The language used must be one of the languages of the underlying archetypes.

4.2.3 Definition section

The definition part of a template or t-archetype is formally expressed in the same way as an archetype. In a template, the definition part usually has the following functions:

1. to declare archetype slot fillers;
2. to redefine or close slots;
3. to remove unwanted attributes and object nodes;
4. to specify local terminology, which can be done in two ways:
   a) explicit inclusion of codes within the template;
   b) the use of a subset / ref-set binding to a place-holder reference (ac-code);
5. to create template-specific ‘clones’ of existing generic nodes;
6. any other narrowing/specialisation of constraints present in the flattened parent archetype, or on otherwise unconstrained underlying reference model attributes;
7. setting template-wide rules such as conditional existence;
8. add default values, usually on leaf or near-leaf nodes.

All of these functions are performed with archetype semantics defined in the AOM, and are described below.

4.2.4 Slot-filling

A template is created as a specialisation of an archetype, usually one based on a top-level reference model concept like `COMPOSITION` or `PARTY`, but this need not be the case. Within such archetypes there are usually slots, i.e. `ARCHETYPE_SLOT` nodes (although this is not mandatory). Where there are slots within the archetype, the template can do two things in terms of filling the slots:

- fill the slots with appropriate archetypes;
- 'close' the slot for further filling at runtime.

It does this by defining a specialisation of the ARCHETYPE_SLOT found in the underlying archetype, which is in the form of any of the following:

- slot fillers in the form of one or more C_ARCHETYPE_ROOT nodes;
- a redefined version of the ARCHETYPE_SLOT object itself, either to narrow its defnition, or to 'close' it.

C_ARCHETYPE_ROOT nodes do not have sub-structure within a template source - they are a single node acting as a reference to the archetype to be used at that point in the operational template. This may be one of:

- a t-archetype, i.e. a redefinition of an archetype used privately by the current template;
- a locally pubished template, designed to be reused in other higher-level templates;
- a normal published archetype, with no further constraints imposed on it by the template.

In all cases, the identifier is a legal archetype identifier, and is stored in the *node_id* attribute, normally used for storing at-codes on all other object node types. This *node_id* is what will be used in both the operational template, and in data, at any archetype root point. In order for the C_ARCHETYPE_ROOT node to be associated with the original slot, a second attribute *slot_node_id* is used to record the at-code of that slot. The use of these two attributes can be seen in the template in FIGURE 3.

A C_ARCHETYPE_ROOT node, like any other object node, can redefine the occurrences constraint as long as:

- it is within the occurrences of the underlying slot;
- it is compatible to the cardinality constraint of the parent attribute.

4.2.5 Slot Redefinition and Closing

The ability to redefine a slot by specifying fillers, and/or redefining the slot itself, gives rise to the a number of variations in a template, including:

- it may specify the archetypes filling a slot and close the slot;
- it may specify the archetypes filling a slot and leave the slot open, possibly narrowed;
- it may specify no fillers, but redefine the slot itself, narrowing the set of archetypes that could match it;
- it may simply close the slot - this only makes sense if there are already slot fillers defined;
- it may make no statements about a slot, leaving things as they were before.

Closing a slot, is achieved by including an ARCHETYPE_SLOT node within the template, which sets the the *is_exhaustive* attribute to True. In the variants where the slot remains open, the runtime system will be able to add further t-archetypes to a slot. Such t-archetypes are either constructed on the fly, or may exist in a store of pre-validated and flattened template components, but in any case the end result must validate for the template to function properly.

4.2.6 Removal of Archetype Nodes

The most common kind of 'redefinition' specified within a template on an archetype used within the template is the removal of unwanted nodes and sub-trees. This is because most archetypes are

designed as ‘maximal data sets’, and include a large superset of data point definitions that would ever be used in a particular context. Two types of removal are required:

- removal of one or more object children of a container attribute;
- removal of the entirety of an attribute (container or single-valued).

A simple example of the first situation is the OBSERVATION.blood_pressure archetype, which defines the following data points:

- systolic blood pressure;
- diastolic blood pressure;
- mean arterial pressure (MAP);
- pulse pressure.

In any particular clinical context, the only meaningful combinations that can be used are: systolic & diastolic (typical general practice, hospital use) OR mean arterial pressure (used as the perfusion pressure by anaesthetists) OR pulse pressure (difference between systolic and diastolic, used in many modern monitoring machines). Thus, one job of the template is to choose which of the above combinations is to be used in the template. This is done by the use of exclusion constraints on the unwanted nodes, i.e. setting of the occurrences constraint to {0}, as illustrated in the top half of FIGURE 4.

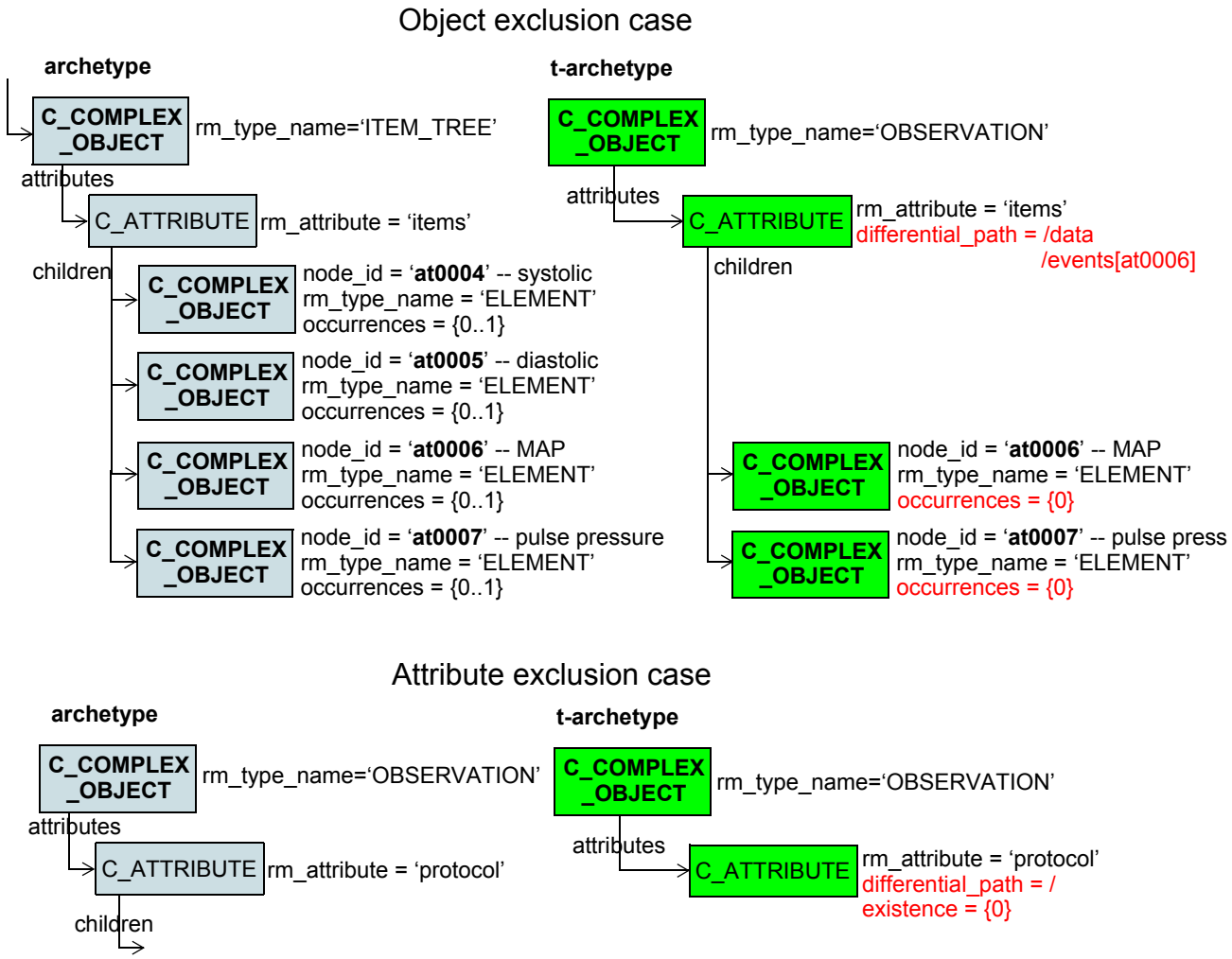


FIGURE 4 Exclusion of unwanted nodes

An example of the second situation is the removal of the OBSERVATION.*protocol* attribute from the blood pressure archetype by a template for which no protocol information (instrument type, measurement location on body etc) is needed. This is shown in the lower half of FIGURE 4.

4.2.7 Node cloning

The concept of creating a number of more specialised variants of a container attribute child object node within a template is exactly as for any normal archetype. The node in question is redefined by one or more 'clones' whose at-codes are specialisations of the at-code of the original node. Optionally, the occurrences of the original node may be reduced to {0}, meaning that no occurrences of data matching only the generic node may be created, i.e. all data instances must match one of the specialised nodes. FIGURE 5 illustrates the cloning of the at0013 ('panel item') node from a generic 'lab result' archetype into a number of specific types of children, namely LDL, HDL and total cholesteral.

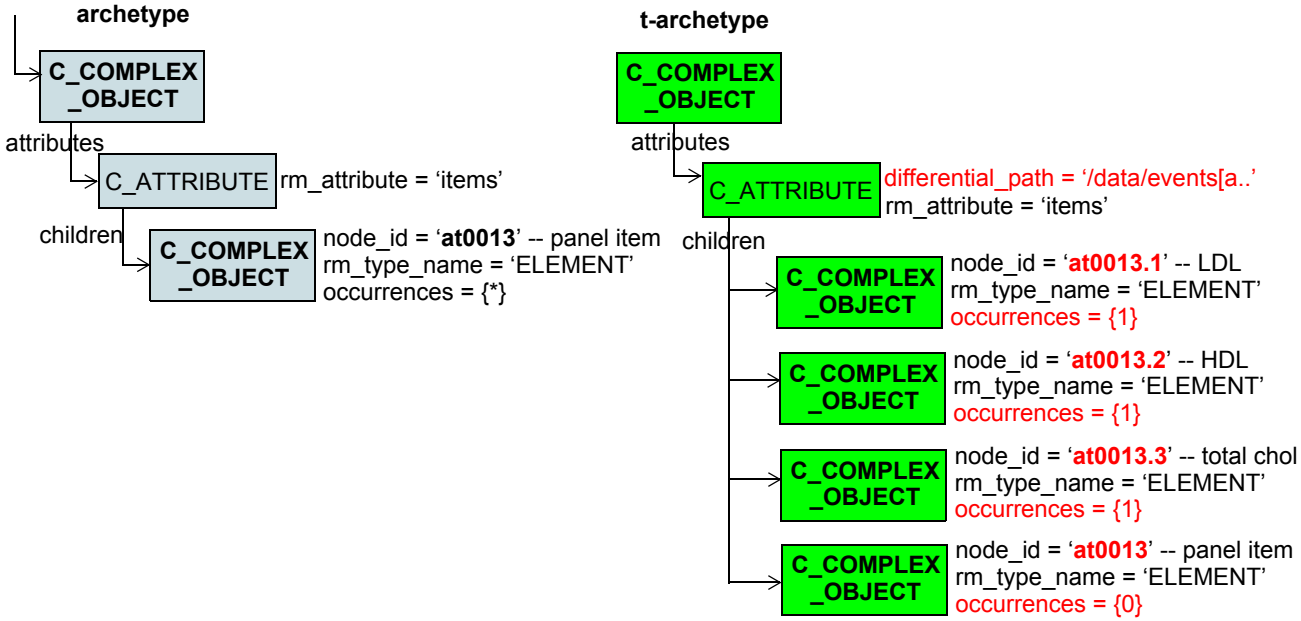


FIGURE 5 Template node cloning

Because node specialisation in templates is performed in the same way as for an archetype, the effect on querying is the same: all cloned (specialised) nodes carry a specialised version of the parent at-code, and will match a query containing the any parent at-code in a path or expression. Thus, a query including the archetype path `.../data/events[at0006]/items[at0013]` will match the at0013.1, at0013.2 and at0013.3 items above, and if at0013 instances had been allowed, these would be matched as well.

4.2.8 Terminology Subset Redefinition

The use of more constrained terminology subsets can be achieved in a template in the same way as within a specialised archetype. The following redefinitions are possible:

- a node whose values are defined by local at-codes can be defined to be a subset of the same codes: either by redefinition into a subset, or by redefinition using the !matches operator, to remove undesired codes;
- a node whose values are defined by an external terminology subset, referenced by an ac-code can have the reference set redefined into another narrower reference set;

- a node whose values are defined by an external terminology subset, referenced by an ac-code can have the reference set redefined into a set of local codes whose values are a semantic subset of those of the originally referenced external subset;
- a node can be defined to have a set of values from an external terminology, declared inline in the template.

The semantics of these redefinitions are identical with those allowed in specialised archetypes, although the use of terminology-specific constraints is generally rare in archetypes but quite common in templates.

4.2.9 Default Values

Default values are defined on the C_DEFINED_OBJECT in the AOM, and instances can be defined within a template on any descendant of this class. In practice, the use of default values in templates is normally limited to primitive types and other near-leaf complex types, including descendants of the C_DOMAIN_TYPE class. Default values are always of the reference model type constrained by the archetype object node to which they are attached.

In a template, the default value may be the only override set on a node. In terms of object structure, this will result in an empty C_COMPLEX_OBJECT or C_PRIMITIVE_OBJECT, to which the default value object is attached.

FIGURE 6 illustrates the use of a default value in a template.

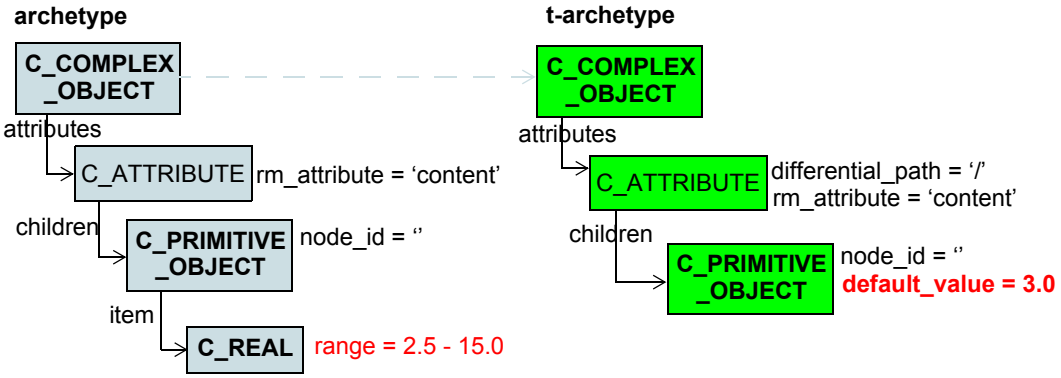


FIGURE 6 Default values in a template

4.2.10 Rules

Template rules are expressed in the form of logic statements, using the RULE_STATEMENT classes defined in the Archetype Object Model, and are attached to the template in the same way as for archetypes, i.e. via the *rules* attribute. The meaning of any such condition is that if present, and evaluated to True at runtime, the constraint to which it is attached will be considered to apply. Absence of a condition means that the constraint will always apply.

The only difference between rules expressed in a template - either in the root template or in any component t-archetype - and rules within a normal archetype is that template rules may contain paths that cross archetype boundaries. Such paths contain the archetype (i.e. sub-template or t-archetype) identifier in the place of an at-code at the archetype root point, but are otherwise the same as any other path.

4.3 Template Validity

5 Examples

5.1 Constrain *open*EHR name attribute to Text

5.2 Template adds terminology subset to DV_TEXT

Original ADL:

DV_TEXT matches {}

want to add coding constraint like

DV_CODED_TEXT matches {[ac0003]} -- but do we want to require acnnnn + binding approach

plus may want to allow default to coding with any code, i.e.

DV_CODED_TEXT matches {[snomed::]}

5.3 No coding allowed on DV_TEXT constraint

how to remove the ability to allow a DV_CODED_TEXT on a DV_TEXT constraint in the archetype - just detect in the application, since there will be no details of subset or terminology set;

TBD_8: Q how to disinquish from the case where free coding allowed?

5.4 Override EVENT with INTERNAL_EVENT

Override with INTERNAL_EVENT that has width set and limited set of codes on math_function

5.5 Condition on Age or Sex

To Be Continued:

6 Serialisation

6.1 dADL

To Be Continued:

6.2 XML

To Be Continued:

7 The Ttemplate Package

7.1 Overview

The `template` package is the fourth package within the archetype model, and is illustrated in context in FIGURE 7.

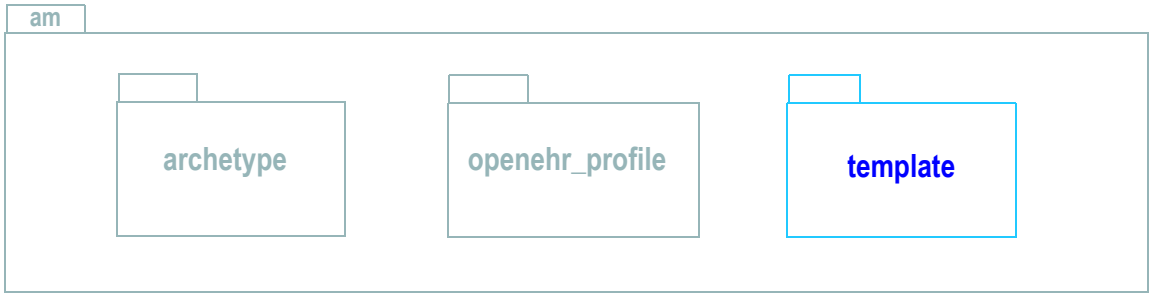


FIGURE 7 openehr.am.template Package in context

FIGURE 8 illustrates the `operational_template` package.

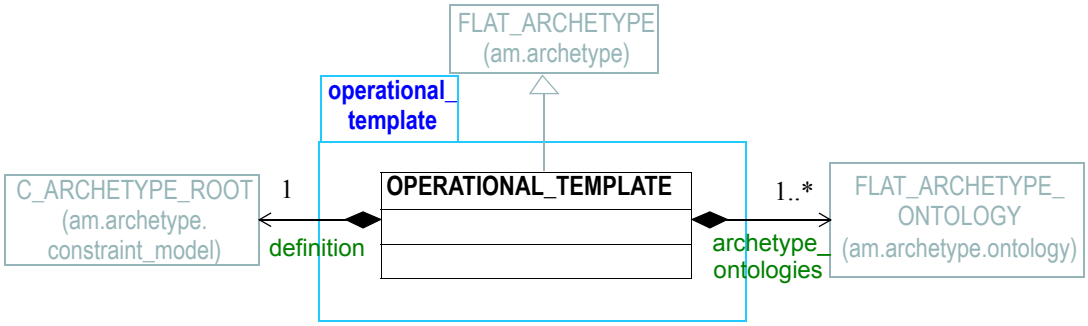


FIGURE 8 The openehr.am.template.operational_template Package

As can be seen from the model, an operational template has nearly the same form as a template definition.

The operational template can be thought of as the flat-form version of a template definition, which is a differential artefact. The `operational_template` package also includes the *open*EHR palette, which defines the languages and terminologies to be used for a particular purpose.

7.2 Design

Notes - in an operational template:

- use_nodes are expanded out to copies of the referenced nodes
- ac_code nodes remain as is, unless there is a terminology binding where we use a subtype of constraint_ref that has a terminology uri; or unless pre-populated
- unneeded languages are stripped
- unneeded terminologies are removed
- unfilled slots: an `ARCHETYPE_SLOT` object is retained

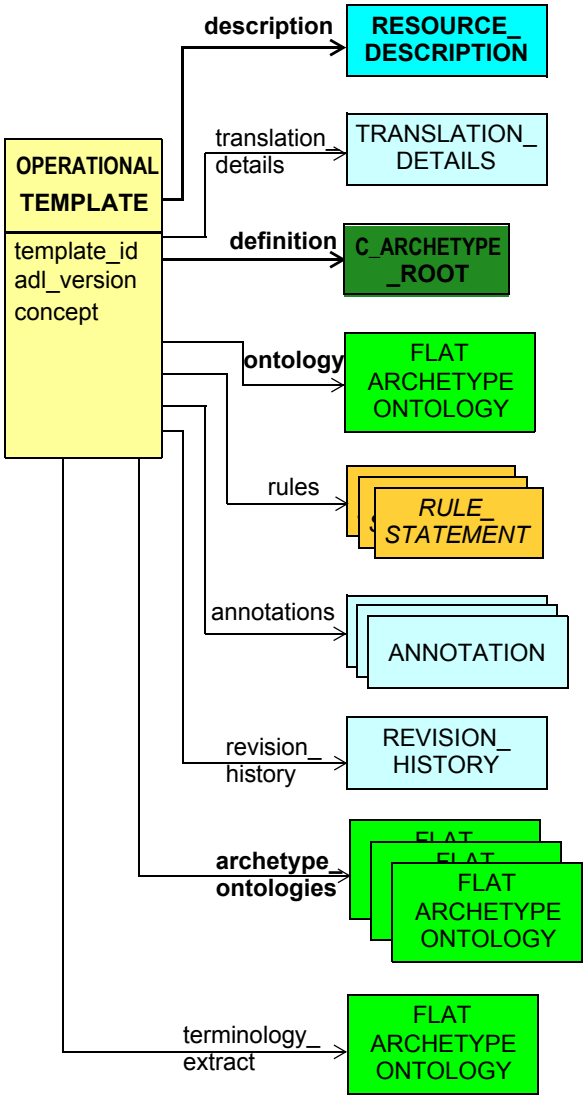


FIGURE 9 Typical Operational Template Object Structure

- filled slots - how to leave slot open for runtime addition
- ontology includes:
  - template codes section
  - section for each archetype, indexed by archetype id
- removal of all annotations

7.3 Class Definitions

7.3.1 OPERATIONAL_TEMPLATE Class

CLASS	OPERATIONAL_TEMPLATE	
Purpose	Root object of an operational template. An operational template is derived from a template definition and the archetypes mentioned by that template by a process of flattening, and potentially removal of unneeded languages and terminologies.	
Use		
Inherit	`FLAT_ARCHETYPE`	
Attributes	**Signature**	**Meaning**
1 (redefined)	**definition**: `C_ARCHETYPE_ROOT`	
1	**component_ontologies**: `Hash <FLAT_ARCHETYPE_ONTOLOGY, String>`	Compendium of flattened ontologies of all archetypes used in template, keyed by archetype identifier.
0..1	**terminology_extracts**: `Hash< FLAT_ARCHETYPE_ONTOLOGY, String>`	Extracts of external terminologies used in template, as a dictionary of ontologies, keyed by terminology identifier.
Functions	**Signature**	**Meaning**
	archetype_ontology (an_id: `String`): `FLAT_ARCHETYPE_ONTOLOGY`	Ontology for archetype with identifier *an_id*.
	terminology_extract (an_id: `String`): `FLAT_ARCHETYPE_ONTOLOGY`	Terminology extract for terminology with identifier *an_id*.
Invariant	***definition_valid***: definition /= Void ***archetype_ontologies_valid***: archetype_ontologies /= Void	

C References

Publications

1 Beale T. *Archetypes: Constraint-based Domain Models for Future-proof Information Systems*. OOPSLA 2002 workshop on behavioural semantics.
Available at http://www.deepthought.com.au/it/archetypes.html.

2 Beale T. *Archetypes: Constraint-based Domain Models for Future-proof Information Systems*. 2000.
Available at http://www.deepthought.com.au/it/archetypes.html.

3 Beale T, Heard S. The Archetype Definition Language (ADL). See http://www.openehr.org/repositories/spec-dev/latest/publishing/architecture/archetypes/language/ADL/REV_HIST.html.

4 Heard S, Beale T. Archetype Definitions and Principles. See http://www.openehr.org/repositories/spec-dev/latest/publishing/architecture/archetypes/principles/REV_HIST.html.

5 Heard S, Beale T. *The openEHR Archetype System*. See http://www.openehr.org/repositories/spec-dev/latest/publishing/architecture/archetypes/system/REV_HIST.html.

Resources

6 *open*EHR. EHR Reference Model. See http://www.openehr.org/repositories/spec-dev/latest/publishing/architecture/top.html.

END OF DOCUMENT