openEHR

The *open*EHR Archetype Model

Archetypes: Technical Overview

Issuer: openEHR Specification Program		
Revision: 0.8	*Pages:* 18	*Date of issue:* 10 Oct 2014
Status: DEVELOPMENT		

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.

Affiliates Australia, Brazil, Japan, New Zealand, Portugal, Sweden

Support **Issue tracker**: www.openehr.org/issues/browse/SPECPR
Web: www.openEHR.org

Amendment Record

Issue	Details	Raiser	Completed
0.8	Initial writing; taken from overview material of ADL, AOM and template specifications.	T Beale	10 Oct 2014

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).

1 Introduction

1.1 Purpose

This document provides a technical overview of the archetype formalism, its artefacts and tools.

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.

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 Introduction

2.1 The Big Picture

The archetype formalism, coupled with orthodox information models (typically object-oriented), provides a way to model information from any domain in 3 logical layers. These are as follows:

Information model: known as the 'Reference Model' (RM) here, which defines the semantics of data;

Archetypes: models defining *possible* arrangements of data that correspond to logical data points and groups for a *domain topic*; a collection of archetypes constitutes a library of reusable domain content definition elements;

Templates: models of content corresponding to *use-case specific data sets*, constituted from archetype elements.

One may ask: why not model the information of a domain solely in an information model, i.e. in just one layer, as described by many IT textbooks? The short answer is that for complex domains, including the biomedical and clinical, the massive complexity and diversity of information items leads to unmaintainable software, systems and data. Hence, in the archetype approach, the reference model level is limited to defining data elements and structures, such as Quantity, CodedConcept, and various generic containment structures. This enables stable data processing sofware to be built and deployed independently of the definition of specific domain information entities.

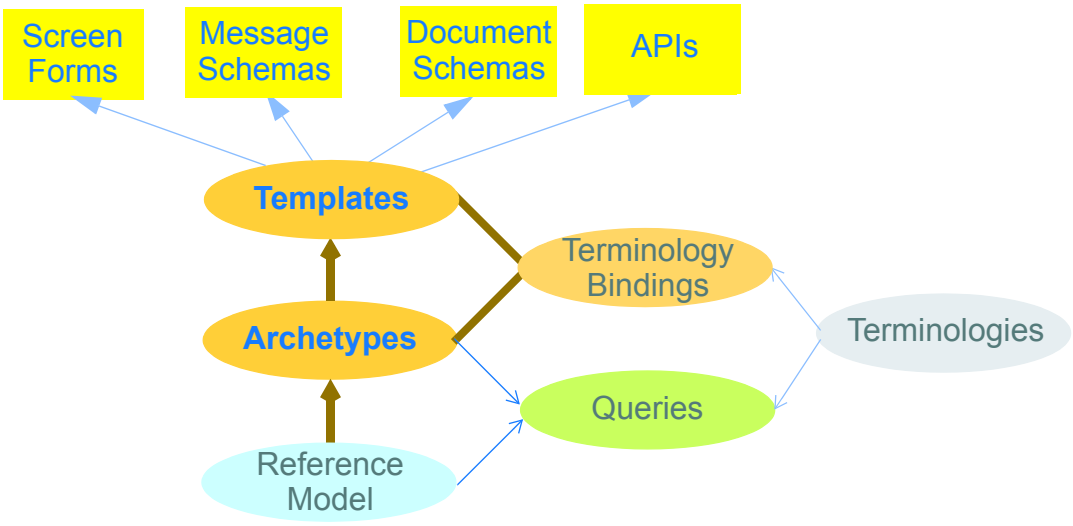

FIGURE 1 The *open*EHR Semantic Architecture

Why use two more layers instead of just one? It turns out that *reuse* and *use* need to be treated somewhat differently. A reusable information entity in the health domain consists of the various parts of a common domain data item, such as '(record of) blood pressure measurement'. The elements include 'systolic blood pressure', 'diastolic blood pressure', 'patient position', 'cuff size' and so on. These are clearly reusable across a multitude of specific environments, settings and applications. Accordingly these topic-based definitions are expressed in the Archetype layer, forming a *library of generic domain information definitions*.

When it comes to 'use', definitions of data sets are specific to a screen form, API result, or a particular message, need their own layer of models, known as *templates*. These models reuse elements from

the archetype library, combining them in such a way as to define every *data set* required by an application or system.

A template is an artefact that enables the content defined in published archetypes to be used for a particular use case or business event. In health this is often 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 often have a corresponding screen 'form' (which may have one or more 'pages' or subforms and some workflow logic) associated with them; as a result, an *open*EHR template 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.

One other semantic ingredient is crucial: that of **terminology**, with the implication of underlying ontology/ies. The RM/archetypes/templates 'stack' defines information as it is created and used - an *epistemological* view, but does not try to describe the *ontological* semantics of each information element. The latter is the business of ontologies, such as can be found in the Open Biomedical Ontology (OBO) Foundry, and terminologies, such as SNOMED CT, which define naming over an ontological framework for use in particular contexts.

The connection of the information model stack to terminology is made in archetypes and templates, via *terminology bindings*. Through these, it is possible to state within archetypes and templates the relationship between the 'names' of elements (ontologically: what the element 'is-about') with terminology and ontology entities, as well as the relationship between element values, and value domains on the terminology side.

A final ingredient is required in the semantic mix: **querying**. Under the archetype methodology, information queries are defined solely in terms of archetype elements (via paths), terminology concepts and logical reference model types, without regard to data schemas used in the persistence layer. Archetype-based queries are therefore *portable queries*, and only need to be written once for a given logical information structure.

Together, reference model, archetypes, and templates (with bound terminology) provide a powerful semantic model space. However, any model needs to be deployed to be useful. Because templates are defined as abstract artefacts, they enable single-source generation of concrete artefacts such as XML schemas, screen forms and so on. This approach means that a single definition of the data set for 'diabetic patient encounter' can be used to generate a message definition XSD and a screen form.

It is the template 'operational' form that provides the basis for tool-generation of usable downstream concrete artefacts, which embody all of the semantics of the implicated Templates in a form usable by 'normal' developers with typical expertise and skills.

Downstream artefacts when finally deployed in operational systems are what enable data to be created and queried. Artefacts created by archetype-based ecosystems enable informations systems of significantly higher quality, semantic power and maintainability to be built, because both the data and querying are model-based, and the models are underpinned by terminology and ontology.

Underlying all of this are of course formalisms and tooling - the language and tools of archetypes. This overview describes the archetype specifications and how they fit together and support tool-building as well as downstream model-based software development.

2.2 The Specifications

The semantics described above are defined in the following set of specifications:

- **ADL**: the Archetype Definition Language: a normative abstract syntax for archetypes, templates and terminology binding;
- **AOM**: the Archetype Object Model: the normative structural model of archetypes and templates;
- **AQL**: the Archetype Querying Language: the normative querying language based on archetypes and terminology;
- **The Template Guide**: an informative specification of how the templating semantics of ADL/AOM are used to create templates;
- **Knowledge Artefact Identification**: a normative specification of archetype and template model identification, versioning, referencing and lifecycle;

The Archetype Definition Language (ADL) is a formal abstract syntax for archetypes, and can be used to provide a default serial expression of archetypes. It is the **primary document for human *understanding* of the semantics of archetypes**.

The AOM is the definitive formal expression of archetype semantics, and is independent of any particular syntax. **The main purpose of the AOM specification is to specify to developers how to build archetype tools and also EHR components that use archetypes.**

The semantics defined in the AOM are used to express the object structures of source archetypes and flattened archetypes. Since in ADL 2 a template is just a kind of archetype, the AOM also describes the semantics of templates as well. The two source forms are authored by users via tools, while the two flat forms are generated by tools. The rules for how to use the AOM for each of these forms is described in details in this specification.

3 Models

This section provides an overview of archetype and template semantics.

3.1 Archetypes

Archetypes are topic- or theme-based models of domain content, expressed in terms of constraints on a reference information model. Since each archetype constitutes an encapsulation of a set of data points pertaining to a topic, it is of a manageable, limited size, and has a clear boundary. For example an 'Apgar result' archetype of the *open*EHR reference model class `OBSERVATION` contains the data points relevant to Apgar score of a newborn, while a 'blood pressure measurement' archetype contains data points relevant to the result and measurement of blood pressure. Archetypes are assembled by templates to form structures used in computational systems, such as document definitions, message definitions and so on.

3.1.1 Archetype Relationships

A 'system' of archetypes is a collection of archetypes covering all or part of a domain, such as clinical medicine. Apart from versioning, two kinds of relationship can exist between archetypes in the system: specialisation and composition.

Archetype Specialisation

An archetype can be specialised in a descendant archetype in a similar way to a subclass in an object-oriented programming language. Specialised archetypes are, like classes, expressed in a *differential* form with respect to the *flat* parent archetype, i.e. the effective archetype resulting from a flattening operation down the specialisation hierarchy. This is a necessary pre-requisite to sustainable management of specialised archetypes. An archetype is a specialisation of another archetype if it mentions that archetype as its parent, and only makes changes to its definition such that its constraints are 'narrower' than those of the flat parent. Note that this can include a specialised archetype defining constraints on a part of the reference model not constrained at all by parent archetypes - since this is still 'narrowing' or constraining. The chain of archetypes from a specialised archetype back through all its parents to the ultimate parent is known as an *archetype lineage*. For a non-specialised (i.e. top-level) archetype, the lineage is just itself.

In order for specialised archetypes to be *used*, the differential form used for authoring has to be *flattened* through the archetype lineage to create *flat-form archetypes*, i.e. the standalone equivalent of a given archetype, as if it had been constructed on its own. A flattened archetype is expressed in the same serial and object form as a differential form archetype, although there are some slight differences in the semantics, other than the use of differential paths.

Any data created via the use of an archetype conforms to the flat form of the archetype, and to the flat form of every archetype up the lineage.

Archetype Composition

In the interests of re-use, archetypes can be composed to form larger structures semantically equivalent to a single large archetype. Composition allows two things to occur: for archetypes to be defined according to natural 'levels' or encapsulations of information, and for the re-use of smaller archetypes by higher-level archetypes. There are two mechanisms for expressing composition: direct reference, and archetype *slots* which are defined in terms of constraints. The latter, unlike an object model, allows an archetype to have a composition relationship with any number of archetypes matching

some constraint pattern. Depending on what archetypes are available within the system, the archetypes matched may vary.

These semantics are described in detail in their syntax form in the ADL specification, and in structural form in the AOM specification.

3.1.2 Archetype Definition Language (ADL)

ADL uses three syntaxes, cADL (constraint form of ADL), ODIN (Object Data Instance Notation), and a version of first-order predicate logic (FOPL), to express constraints on data which are instances of an underlying information model, which may be expressed in UML, relational form, or in a programming language. ADL itself is a very simple 'glue' syntax, which uses two other syntaxes for expressing structured constraints and data, respectively. The cADL syntax is used to express the archetype `definition`, while the ODIN syntax is used to express data which appears in the `language`, `description`, `terminology`, and `revision_history` sections of an ADL archetype. The top-level structure of an ADL archetype is shown in FIGURE 2.

This main part of this document describes cADL and ADL path syntax, before going on to describe the combined ADL syntax, archetypes and domain-specific type libraries.

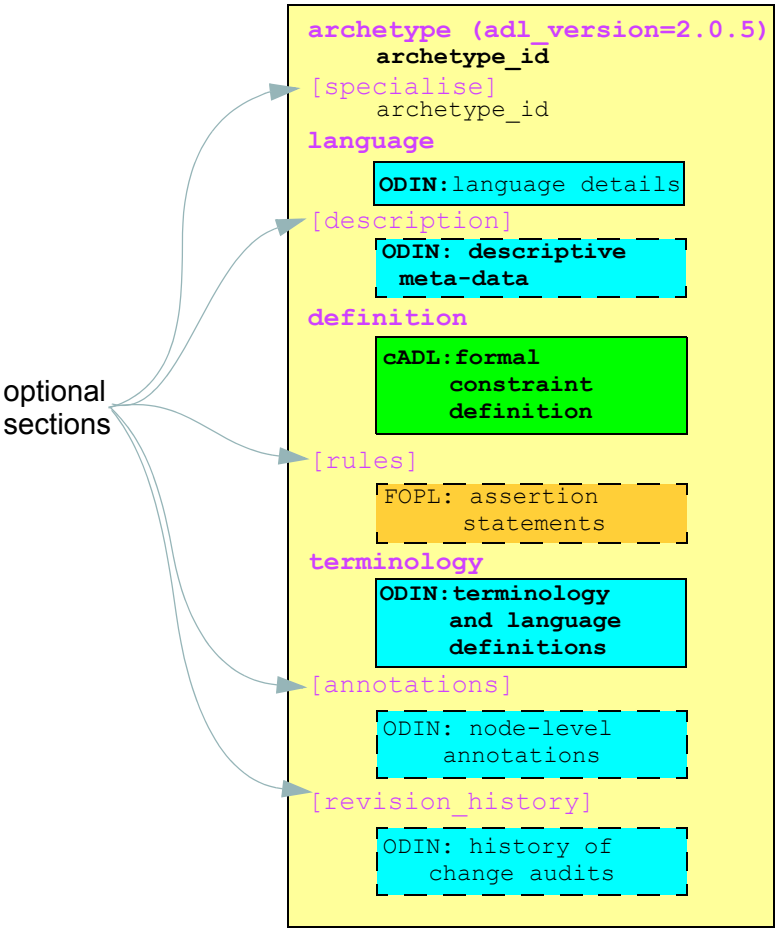

FIGURE 2 ADL Archetype Structure

3.1.3 The Role of ADL

While ADL is a normative syntax formalism for archetypes, the Archetype Object Model defines the semantics of an archetype, in particular relationships that must hold true between the parts of an

archetype for it to be valid as a whole. Other syntaxes are therefore possible. In particular, XML-schema based XML is used for many common computing purposes relating to archetypes, and may become the dominant syntax in terms of numbers of users. Nevertheless, XML is not used as the normative syntax for archetypes for a number of reasons.

- There is no single XML schema that is likely to endure. Initially published schemas are replaced by more efficient ones, and may also be replaced by alternative XML syntaxes, e.g. the XML-schema based versions may be replaced or augmented by Schematron.
- XML has little value as an explanatory syntax for use in standards, as its own underlying syntax obscures the semantics of archetypes or any other specific purpose for which it is used. Changes in acceptable XML expressions of archetypes described above may also render examples in documents such as this obsolete.

An XML schema for archetypes is available on the *open*EHR website.

3.2 Templates

In practical systems, archetypes are assembled into larger usable structures by the use of templates. A template is expressed in the same source form as a specialised archetype, although typically making use of the slot-filling mechanism. It is processed against an archetype library to product an *operational template*. The latter is like a large flat-form archetype, and is the form used for runtime validation, and also for the generation of all downstream aretefacts derived from templates. Semantically, templates perform three functions: aggregating multiple archetypes, removing elements not needed for the use case of the template, and narrowing some existing constraints, in the same way as specialised archetypes. The effect is to re-use needed elements from the archetype library, arranged in a way that corresponds directly to the use case at hand.

This job of a template is as follows:

composition: compose archetypes into larger structures by indicating which archetypes should fill the slots of higher-level archetypes;

element choice: choose which parts of the chosen archetypes should remain in the final structure, by doing one or more of the following:

removal: remove archetype nodes (‘data points’) not needed for the purpose of the template;

mandation: mark archetype nodes (‘data points’) mandatory for the runtime use of the template;

nodes may also be left optional, meaning that the corresponding data item is considered optional at runtime;

narrow constraints: narrow remaining constraints on archetype elements or on the reference model (i.e. on parts of the RM not yet constrained by the archetypes referenced in the template);

set detaults: set default values if required.

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

The archetype semantics used in templates are described in detail in the ADL specification; a detailed description of their use is given in the Template Guide specification.

The AOM describes in structural form all the semantics of templates. Note that the AOM does not distinguish between archetypes, specialised archetypes or templates other than by use of an artefact type classifier. All other differences in how archetypes and templates work are implemented in tools that may prevent or allow certain operations depending on whether the artefact being worked on is an archetype or template.

4 Artefacts

4.1 Model / Syntax Relationship

The Archetype Object Model can be considered as the model of an in-memory archetype or a template, or equivalently, a standard syntax tree for any serialised format - not just ADL. The normative *abstract* syntax form of an archetype is ADL, but an archetype may just as easily be parsed from and serialised to XML, JSON or any other format. The in-memory archetype representation may also be created by calls to a suitable AOM construction API, from an archetype or template editing tool. These relationships, and the relation between each form and its specification are shown in FIGURE 3.

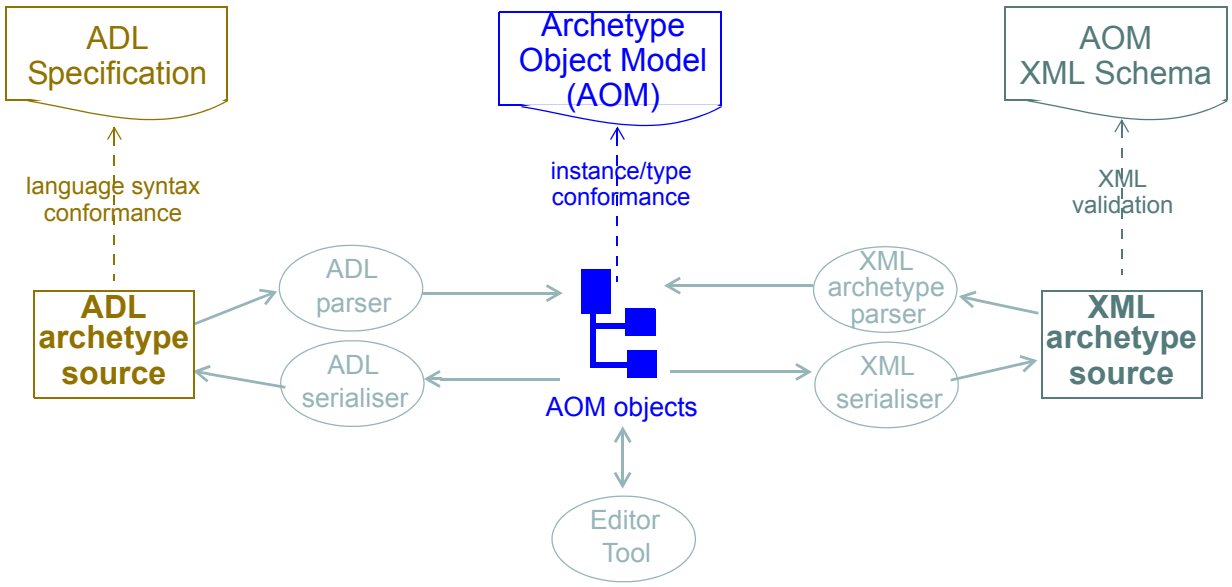

FIGURE 3 Relationship of archetype in-memory and syntax forms

The existence of source and flat form archetypes and templates, potentially in multiple serialised formats may initially appear confusing, although any given environment tends to use a single serialised form. FIGURE 4 illustrates all possible archetype and template artefact types, including file types.

4.2 The Development Process

Archetypes are authored and transformed according to a number of steps very similar to class definitions within an object-oriented programming environment. The activities in the process are as follows:

- *authoring*: creates source-form archetypes, expressed in AOM objects;
- *validation*: determine if archetype satisfies semantic validity rules; requires flat form of parent of current archetype if specialised;
- *flattening*: creates flattened archetypes#

Templates are authored the same way, although typically using only mandations, prohibitions and refinements. The final step is the generation of an operational template, which is the fully flattened and substituted form of the source definitions implicated by the template.

The tool chain for the process is illustrated in FIGURE 5. From a software development point of view, template authoring is the starting point. A template references one or more archetypes, so its

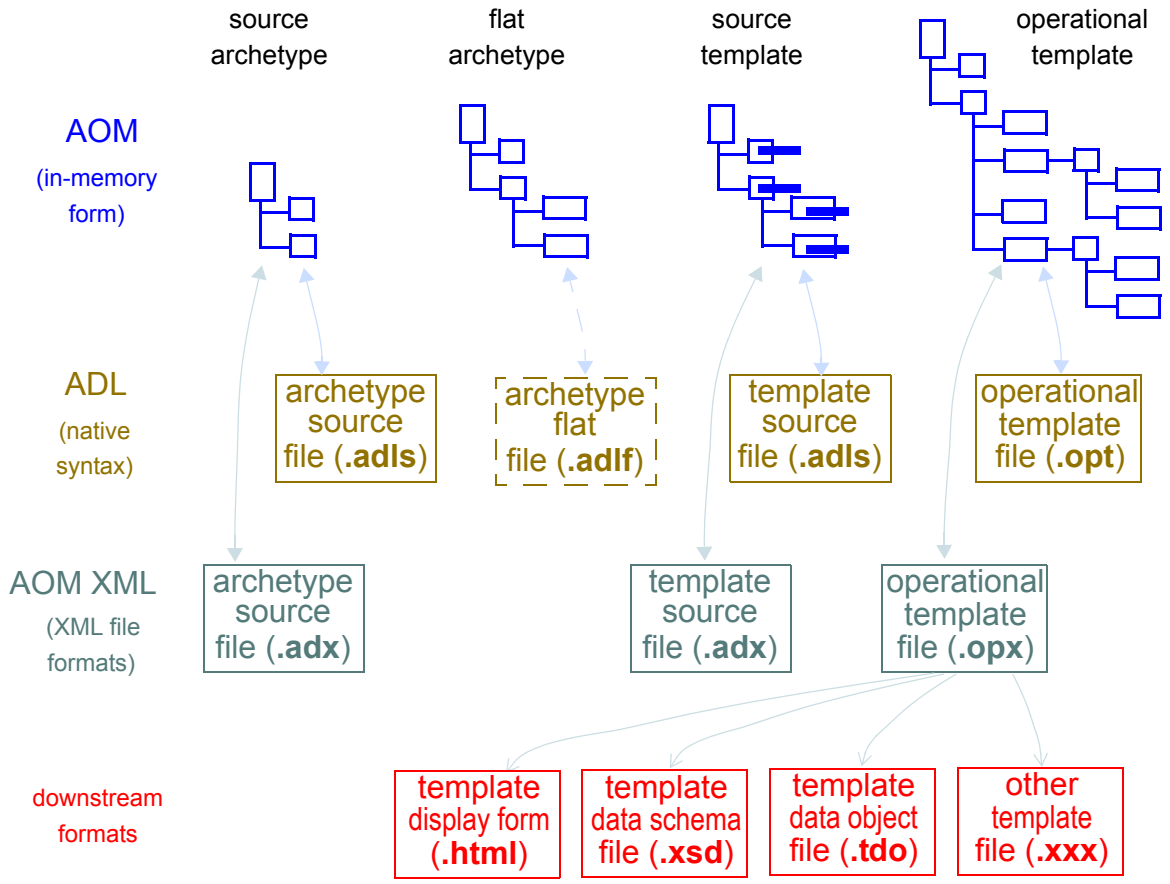

FIGURE 4 Relationship of computational artefacts and specifications

compilation (parsing, validation, flattening) involves both the template source and the validated, flattened forms of the referenced archetypes. With these as input, a template flattener can generate the final output, an operational template.

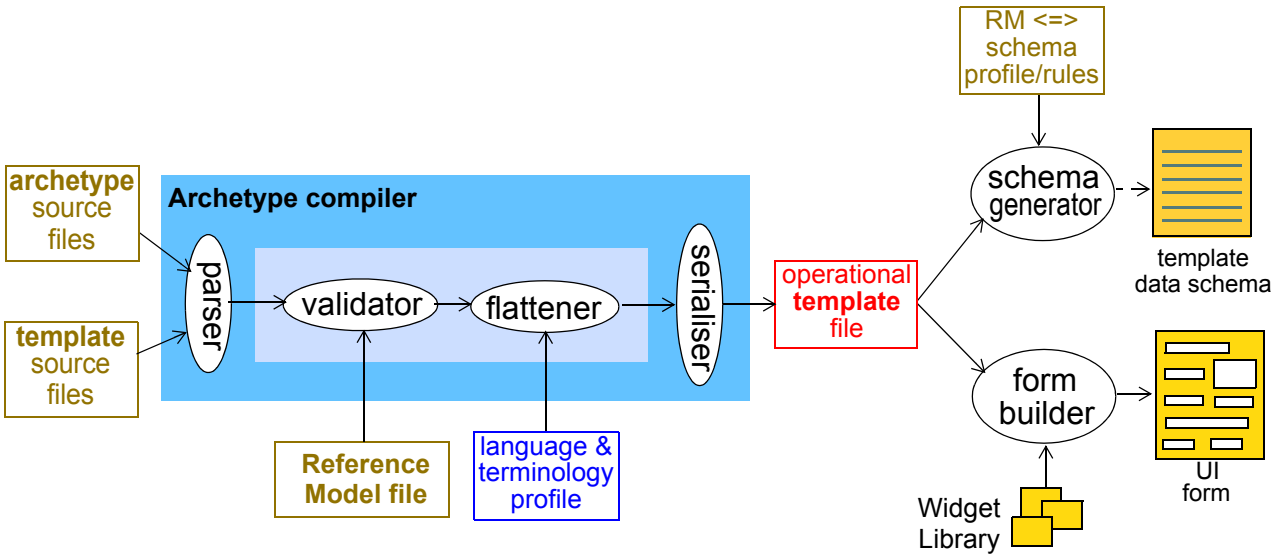

FIGURE 5 Archetype / template tool chain

4.3 Compilation

A tool that parses, validates, flattens and generates new outputs from a library of artefacts is called a *compiler*. Due to the existence of specialisation, *archetype specialisation lineages* rather than just single archetypes are processed - i.e. specialised archetypes can only be compiled in conjunction with their specialisation parents up to the top level. Each such archetype also potentially has a list of *supplier* archetypes, i.e. archetypes it references via the ADL use_reference statement. For an archetype to compile, all of its suppliers and specialisation parents must already compile.

This is exactly how object-oriented programming environments work. For any given lineage, compilation proceeds from the top-level archetype downward. Each archetype is validated, and if it passes, flattened with the parent in the chain. This continues until the archetype originally being compiled is reached. In the case of archetypes with no specialisations, compilation involves the one archetype only and its suppliers.

FIGURE 6 illustrates the object structures for an archetype lineage as created by a compilation process, with the elements corresponding to the top-level archetype bolded. Differential input file(s) are converted by the parser into differential object parse trees, shown on the left of the 'flattener' processes. The same structures would be created by an editor application.

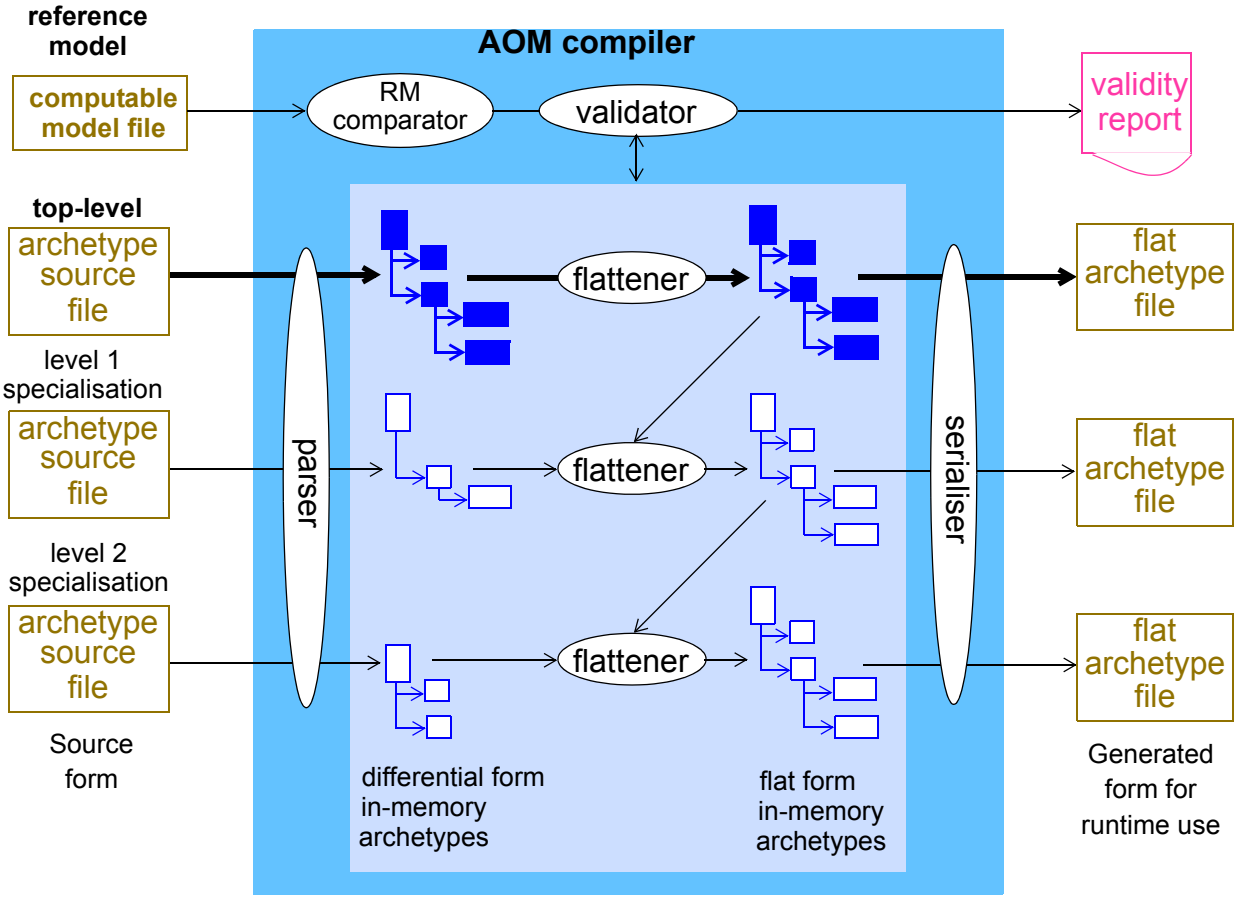

FIGURE 6 Computational model of archetype compilation

The differential in-memory representation is validated by the semantic checker, which verifies numerous things, such as that term codes referenced in the definition section are defined in the terminology section. It can also validate the classes and attributes mentioned in the archetype against a specification for the relevant reference model[1].

The results of the compilation process can be seen in the archetype visualisations in the *open*EHR ADL Workbench[1].

4.4 Optimisations

In an authoring (i.e. 'design time') environment, artefacts should always be considered 'suspect' until proven otherwise by reliable validation. This is true regardless of the original syntax - ADL, XML or something else. Once validated however, the flat form can be reserialised both in a format suitable for editor tools to use (ADL, XML, ...), and also in a format that can be regarded as a reliable pure object serialisation of the in-memory structure. The latter form is often XML-based, but can be any object representation form, such as JSON, the *open*EHR ODIN (aka dADL) syntax, YAML, a binary form, or a database structure. It will not be an abstract syntax form such as ADL, since there is an unavoidable semantic transformation required between the abstract syntax and object form.

The utility of this pure object serialisation is that it can be used as *persistence* of the validated artefact, to be converted to in-memory form using only a non-validating stream parser, rather than a multi-pass validating compiler. This allows such validated artefacts to be used in both design environments and more importantly, runtime systems with no danger of compilation errors. It is the same principle used in creating .jar files from Java source code, and .Net assemblies from C# source code.

Within *open*EHR environments, managing the authoring and persisted forms of archetypes is achieved using various mechanisms including digital signing, which are described in the *open*EHR

1. An ODIN expression of the *open*EHR reference model is available for this purpose.
1. See http://www.openehr.org/svn/ref_impl_eiffel/TRUNK/apps/doc/adl_workbench_help.htm

5 Tools

5.1 Computational Environment

FIGURE 7 shows how an archetype compiler is used to create an operational template from a source template, which contains references to one or more archetypes and usually imposes further constraints.

The parser converts the template into an in-memory object form described by the Archetype Object Model (AOM) and Template extensions defined in this specification. Later stages of the compiler validate and 'flatten' the template, generating as a result a an in-memory Operational Template (OPT) 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. The compiler can take as input a list of terminologies and languages to retain in the OPT.

The OPT may be serialised to ADL, JSON, YAML, or in XML derived from the Archetype Object Model.

Downstream artefacts such as XML schemas, screen forms and APIs may be generated from the OPT by dedicated transformation components.

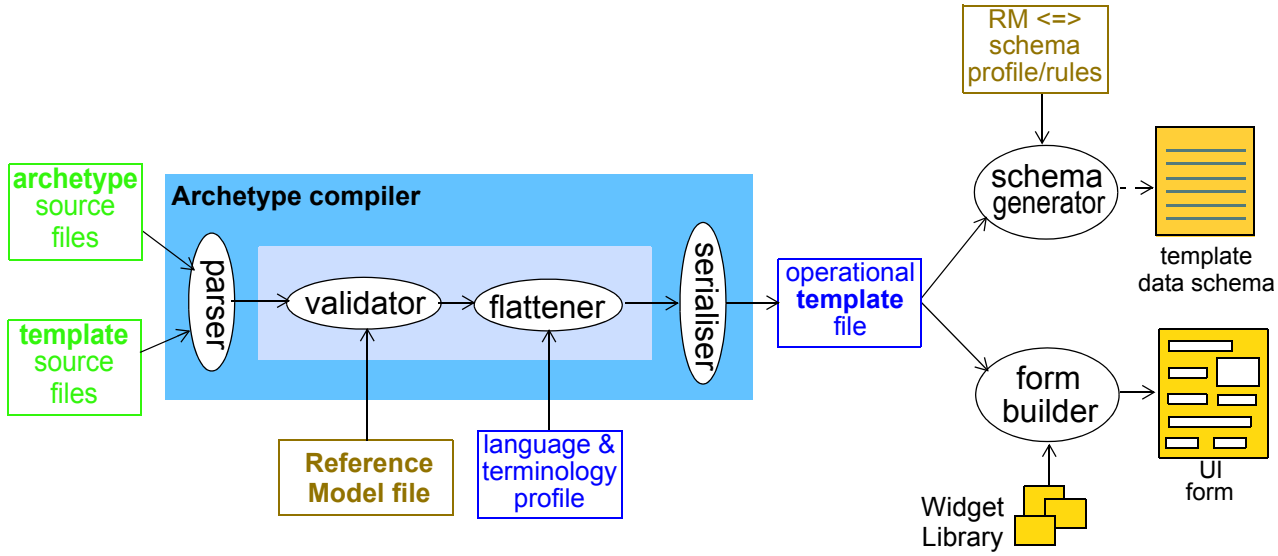

FIGURE 7 Template Tool Chain

END OF DOCUMENT