The *open*EHR Archetype Model

Archetype Definition Language

Version 2 (ADL2)

Editors:{T Beale, S Heard}[1]

Revision: 2.1.0

Pages: 47

Keywords: EHR, health records, modelling, constraints, software

1. Ocean Informatics Australia

The *open*EHR foundation

is an independent, non-profit community, facilitating the creation and sharing of health records by consumers and clinicians via open-source, standards-based implementations.

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

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

Copyright Notice

Amendment Record

Issue	Details	Raiser	Completed
RELEASE 1.0.1			
2.1.0	**CR-000203**: Release 1.0 explanatory text improvements. Improve Archetype slot explanation. DOCUMENT NOW ONLY SHOWS INCLUDES DIFFERENCES FROM ADL 1.x **CR-000208**: Improve ADL grammar for assertion expressions. **CR-000160**: Duration constraints. Added ISO 8601 patterns for duration in cADL. **CR-000213**: Correct ADL grammar for date/times to be properly ISO8601 compliant. Include 'T' in cADL patterns and dADL and cADL Date/time, Time and Duration values. **CR-000216**: Allow mixture of W, D etc in ISO8601 Duration (deviation from standard). **CR-000200**: Correct Release 1.0 typographical errors. **CR-000225**: Allow generic type names in ADL. **CR-000226**: Rename C_CODED_TEXT to C_CODE_PHRASE	T Beale T Beale S Heard T Beale S Heard A Patterson M Forss T Beale	20 Jan 2007
RELEASE 1.0			
2.0.0	**CR-000136**. Add validity rules to ADL document. **CR-000153**. Synchronise ADL and AOM attribute naming. **CR-000154**. Convert ADL archetypes to dADL documents. **CR-000171**. Add validity check for cardinality & occurrences	T Beale T Beale T Beale A Maldondo	18 Jan 2006
RELEASE 0.96			
1.3.0	**CR-000141**. Allow point intervals in ADL. Updated atomic types part of cADL section and dADL grammar section. **CR-000142**. Update dADL grammar to support assumed values. **CR-000143**. Add partial date/time values to dADL syntax. **CR-000149**. Add URIs to dADL and remove query() syntax. **CR-000156**. Update documentation of container types. **CR-000138**. Archetype-level assertions.	S Heard T Beale T Beale T Beale T Beale T Beale	18 Jun 2005
RELEASE 0.95			
1.2.1	**CR-000125**. C_QUANTITY example in ADL manual uses old dADL syntax. **CR-000115**. Correct "/[xxx]" path grammar error in ADL. Create new section describing ADL path syntax. **CR-000127**. Restructure archetype specifications. Remove clinical constraint types section of document.	T Beale T Beale T Beale	11 Feb 2005

Issue	Details	Raiser	Completed
1.2	**CR-000110**. Update ADL document and create AOM document. Added explanatory material; added domain type support; rewrote of most dADL sections. Added section on assumed values, "controlled" flag, nested container structures. Change language handling. Rewrote OWL section based on input from: - University of Manchester, UK, - University Seville, Spain. Various changes to assertions due to input from the DSTC. Detailed review from Clinical Information Project, Australia. **Remove UML models to "Archetype Object Model" document.** Detailed review from UCL. **CR-000103**. Redevelop archetype UML model, add new keywords: allow_archetype, include, exclude. CR-000104. Fix ordering bug when use_node used. Required parser rules for identifiers to make class and attribute identifiers distinct. Added grammars for all parts of ADL, as well as new UML diagrams.	T Beale A Rector R Qamar I Román Martínez A Goodchild Z Z Tun E Browne T Beale T Austin T Beale K Atalag	15 Nov 2004
RELEASE 0.9			
1.1	**CR-000079**. Change interval syntax in ADL.	T Beale	24 Jan 2004
1.0	**CR-000077**. Add cADL date/time pattern constraints. **CR-000078**. Add predefined clinical types. Better explanation of cardinality, occurrences and existence.	S Heard, T Beale	14 Jan 2004
0.9.9	**CR-000073**. Allow lists of Reals and Integers in cADL. **CR-000075**. Add predefined clinical types library to ADL. Added cADL and dADL object models.	T Beale, S Heard	28 Dec 2003
0.9.8	**CR-000070**. Create Archetype System Description. Moved Archetype Identification Section to new Archetype System document. Copyright Assgined by Ocean Informatics P/L Australia to The *open*EHR Foundation.	T Beale, S Heard	29 Nov 2003
0.9.7	Added simple value list continuation (",..."). Changed path syntax so that trailing '/' required for object paths. Remove ranges with excluded limits. Added terms and term lists to dADL leaf types.	T Beale	01 Nov 2003
0.9.6	Additions during HL7 WGM Memphis Sept 2003	T Beale	09 Sep 2003
0.9.5	Added comparison to other formalisms. Renamed CDL to cADL and dDL to dADL. Changed path syntax to conform (nearly) to Xpath. Numerous small changes.	T Beale	03 Sep 2003
0.9	Rewritten with sections on cADL and dDL.	T Beale	28 July 2003
0.8.1	Added basic type constraints, re-arranged sections.	T Beale	15 July 2003
0.8	Initial Writing	T Beale	10 July 2003

Trademarks

Acknowledgements

Sebastian Garde, Central Qld University, Australia, for german translations.

Table of Contents

1 Introduction

1.1 Purpose

This document describes the design basis and syntax of the Archetype Definition Language (ADL). It is intended for software developers, technically-oriented domain specialists and subject matter experts (SMEs). Although ADL is primarily intended to be read and written by tools, it is quite readable by humans and ADL archetypes can be hand-edited using a normal text editor.

The intended audience includes:

- Standards bodies producing health informatics standards;
- Software development organisations using *open*EHR;
- Academic groups using *open*EHR;
- The open source healthcare community;
- 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 Object Model (AOM)
- The *open*EHR Archetype Profile (oAP)

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’.

1.4 Status

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

This document is available at http://svn.openehr.org/specification/TAGS/Release-1.0/publishing/architecture/am/adl2.pdf.

The latest version of this document can be found at http://svn.openehr.org/specification/TRUNK/publishing/architecture/am/adl2.pdf.

Blue text indicates sections under active development.

1.5 Peer review

Known omissions or questions are indicated in the text with a “to be determined” paragraph, as follows:

TBD_1: (example To Be Determined paragraph)

Areas where more analysis or explanation is required are indicated with “to be continued” paragraphs like the following:

To Be Continued: more work required

Reviewers are encouraged to comment on and/or advise on these paragraphs as well as the main content. Please send requests for information to info@openEHR.org. Feedback should preferably be provided on the mailing list openehr-technical@openehr.org, or by private email.

1.6 Conformance

Conformance of a data or software artifact to an *open*EHR Reference Model specification is determined by a formal test of that artifact against the relevant *open*EHR Implementation Technology Specification(s) (ITSs), such as an IDL interface or an XML-schema. Since ITSs are formal, automated derivations from the Reference Model, ITS conformance indicates RM conformance.

2 Overview

2.1 What is ADL?

Archetype Definition Language (ADL) is a formal language for expressing archetypes, which are constraint-based models of domain content. The archetype concept is described by Beale [1], [2]. The *open*EHR Archetype Object Model [3] describes the definitive semantic model of archetypes, in the form of an object model. The ADL syntax is one possible serialisation of an archetype.

The *open*EHR archetype framework is described in terms of Archetype Definitions and Principles [6] and an Archetype System [7]. Other semantic formalisms which were considered in the course of archetype, and some which remain relevant are described in detailed in section 11 on page 43.

To describe constraints on data which are instances of some information model (e.g. expressed in UML), ADL archetypes use three syntaxes: dADL (a data expression syntax), cADL (a constraint expression syntax), and a version of first-order predicate logic (FOPL). It is most useful when very generic information models are used for describing the data in a system, for example, where the logical concepts `PATIENT`, `DOCTOR` and `HOSPITAL` might all be represented using a small number of classes such as `PARTY` and `ADDRESS`. In such cases, archetypes are used to constrain the *valid* structures of instances of these generic classes to represent the desired domain concepts. In this way future-proof information systems can be built - relatively simple information models and database schemas can be defined, and archetypes supply the semantic modelling, completely outside the software. ADL can thus be used to write archetypes for any domain where formal object model(s) exist which describe data instances.

When archetypes are used at runtime in particular contexts, they are *composed* into larger constraint structures, with local or specialist constraints added, via the use of *templates*. The formalism of templates is dADL. Archetypes can be *specialised* by creating an archetypes that reference existing archetypes as parents; such archetypes can only make certain changes while remaining compatible with the parent.

Another major function of archetypes is to connect information structures to formal terminologies. Archetypes are language-neutral, and can be authored in and translated into any language.

Finally, archetypes are completely path-addressable in a manner similar to XML data, using path expressions that are directly convertible to Xpath expressions.

2.1.1 Structure

An ADL archetype is syntactically a dADL document, containing a definition section expression in the cADL syntax. The structure of an ADL archetype is shown in FIGURE 1.

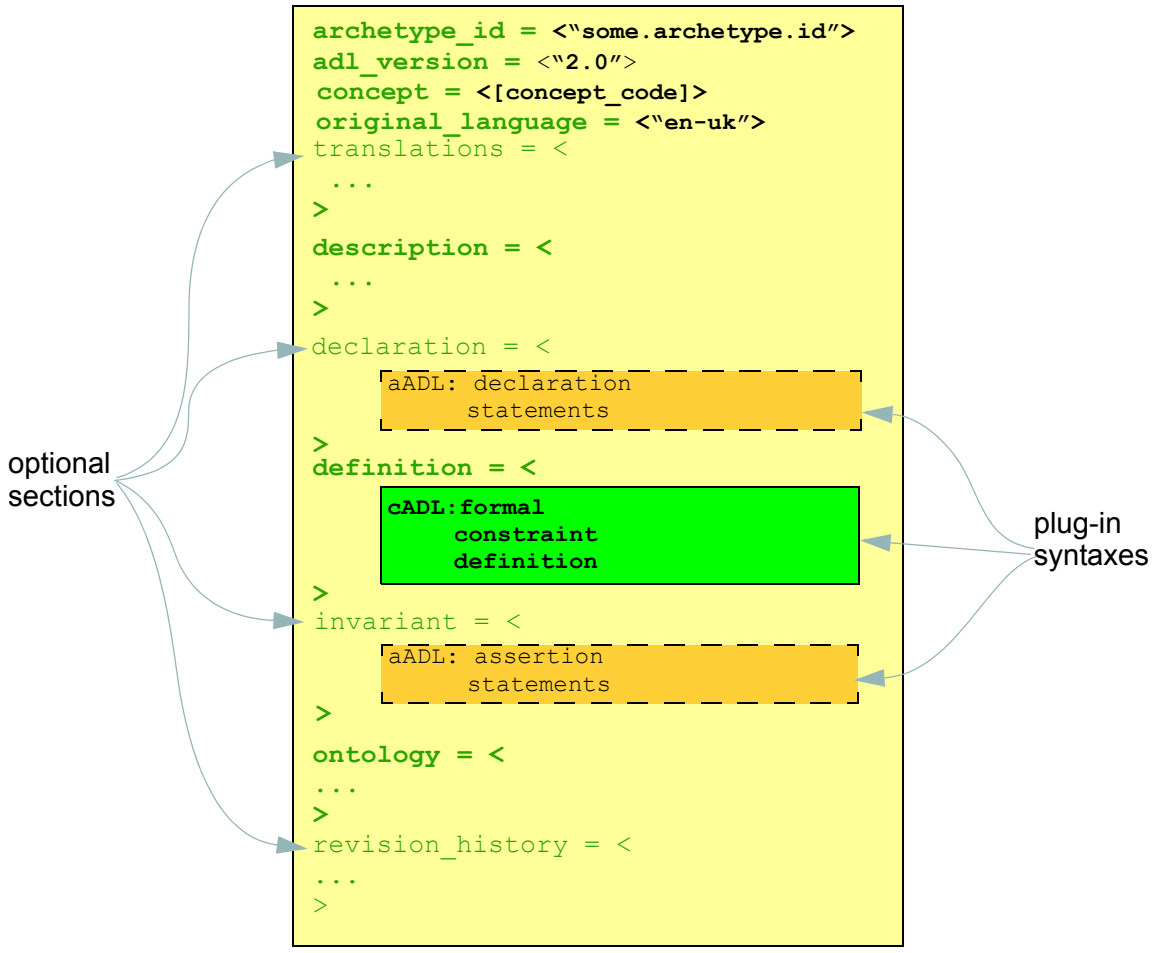

FIGURE 1 ADL Archetype Structure

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

2.1.2 An Example

The following is an example of a very simple archetype, giving a feel for the syntax. The main point to glean from the example is that the notion of 'guitar' is defined in terms of *constraints* on a *generic* model of the concept `INSTRUMENT`. The names mentioned down the left-hand side of the definition section ("INSTRUMENT", "size" etc) are alternately class and attribute names from an object model. Each block of braces encloses a specification for some particular set of instances that conform to a specific concept, such as 'guitar' or 'neck', defined in terms of constraints on types from a generic class model. The leaf pairs of braces enclose constraints on primitive types such as Integer, String, Boolean and so on.

```
archetype_id = <"adl-test-instrument.guitar.draft">
concept = <[at0000]> -- guitar
original_language = <"en">
definition = (cadl) <#
    INSTRUMENT[at0000] matches {
        size matches {|60..120|}                    -- size in cm
        date_of_manufacture matches {yyyy-mm-??}    -- year & month ok
```

```
            parts cardinality matches {0..*} matches {
               PART[at0001] matches {                    -- neck
                  material matches {[local::at0003,}     -- timber
                                    at0004]}             -- timber or nickel alloy
               }
               PART[at0002] matches {                    -- body
                  material matches {[local::at0003]}     -- timber
               }
            }
         }
      #>
      ontology = <
         term_definitions = <
            ["en"] = <
               ["at0000"] = <
                  text = <"guitar">;
                  description = <"stringed instrument">
               >
               ["at0001"] = <
                  text = <"neck">;
                  description = <"neck of guitar">
               >
               ["at0002"] = <
                  text = <"body">;
                  description = <"body of guitar">
               >
               ["at0003"] = <
                  text = <"timber">;
                  description = <"straight, seasoned timber">
               >
               ["at0004"] = <
                  text = <"nickel alloy">;
                  description = <"material for frets">
               >
            >
         >
      >
```

2.1.3 Semantics

As a parsable syntax, ADL has a formal relationship with structural models such as those expressed in UML, according to the scheme of FIGURE 2. Here we can see that ADL documents are parsed into a network of objects (often known as a 'parse tree') which are themselves defined by a formal, abstract object model (see The openEHR Archetype Object Model (AOM)). Such a model can in turn be re-expressed as any number of concrete syntaxes, such as in a programming language, XML-schema or OMG IDL.

While ADL syntax remains the primary abstract formalism for expressing archetypes, the AOM defines the semantics of an archetype, in particular relationships which must hold true between the parts of an archetype for it to be valid as a whole.

2.2 Computational Context

An archetypes is a structured model of domain content, such as "blood pressure". Archetypes sit between knowledge resources in a computing environment, such as terminologies and ontologies, and runtime data in production systems. Their primary purpose is to provide a reusable, interoperable way

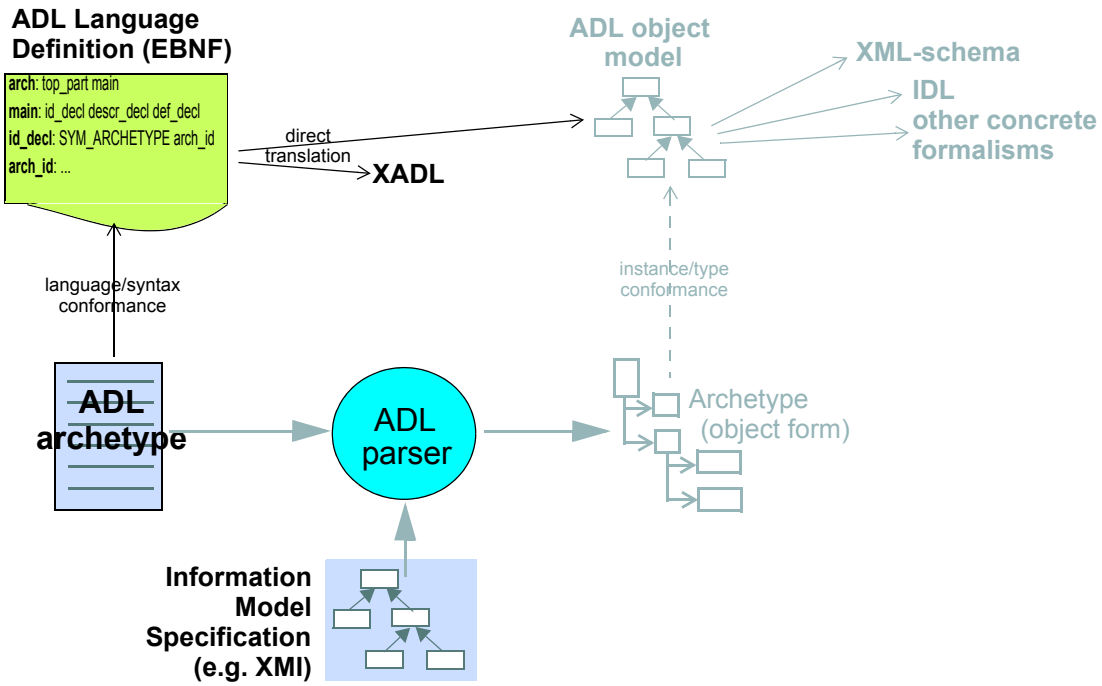

FIGURE 2 Relationship of ADL with Object Models

of managing data creation, validation and querying, by ensuring that data conform to particular structures and semantic constraints. Every ADL archetype is written with respect to a particular information model, often known as a "reference model", if it is a shared, public specification.

Archetypes are applied to data via the use of *templates*, which are defined at a local level. Templates generally correspond closely to screen forms, and may be re-usable at a local or regional level. Templates do not introduce any new semantics to archetypes, they simply specify the use of particular archetypes in particular hierarchical compositions, as well as default data values.

A third artifact that governs the functioning of archetypes and templates at runtime is a local *palette*, which specifies which natural language(s) and terminologies are in use in the locale. The use of a palette removes irrelevant languages and terminology bindings from archetypes, retaining only those relevant to actual use. FIGURE 3 illustrates the overall environment in which archetypes, templates, and a locale palette exist.

2.3 XML form of Archetypes

With ADL parsing tools it is possible to convert ADL to any number of forms, including various XML formats. XML instance can be generated from the object form of an archetype in memory. An XML-schema corresponding to the ADL Object Model is published on *open*EHR.org.

2.4 Changes From Previous Versions

For existing users of ADL or archetype development tools, the following provides a guide to the changes in the syntax.

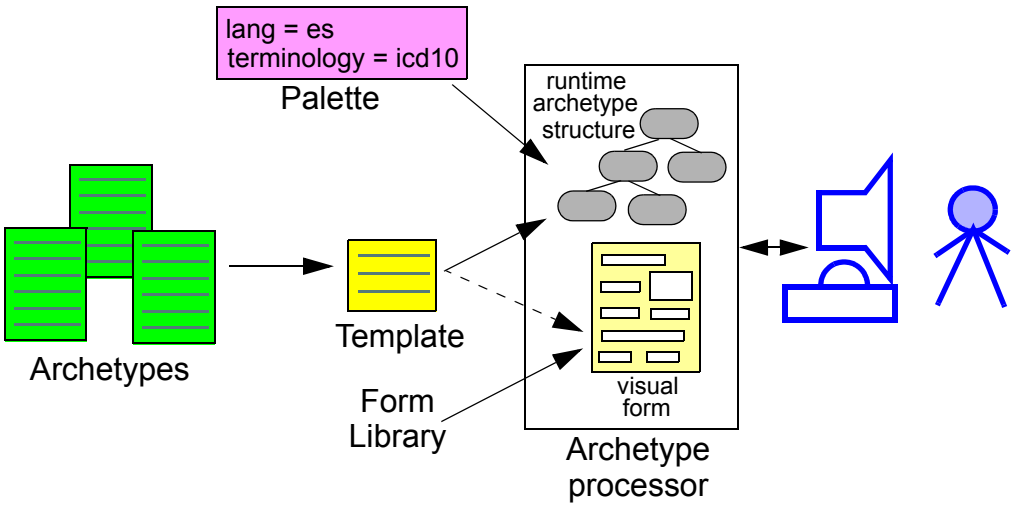

FIGURE 3 Archetypes, Templates, and Palettes

2.4.1 Version 2.0 from Version 1.3

Archetype is dADL Syntax

In this major revision of ADL, the ADL syntax is synchronised properly with the *open*EHR Archetype Object Model, although the semantics remain virtually unchanged. Accordingly, all top-level keywords in an ADL archetype are now considered the direct equivalent of an attribute of the same name in the AOM. To enable this, the entire ADL archetype is now a dADL document, with plug-in syntax sections for expressing constraints and invariants. The specific changes in this revision are:

- replace 'archetype' keyword with a dADL section named 'archetype_id' which has a String value;
- replace the ADL 1.2 adl_version syntax with a new top-level dADL section 'adl_version' which has a String value;
- replace the ADL 1.2 controlled syntax with a new top-level dADL section 'is_controlled' which has a Boolean value;
- replace 'specialise' keyword with a dADL section named 'parent_archetype_id';
- replace 'language' section with two top-level dADL sections 'original_language' and 'translations'.

These changes facilitate human understanding as well as automated processing, particularly serialisation/deserialisation into and out of object form.

A full dADL form of an archetype can also be supported, in which an archetype is a faithful dADL serialisation of instances of the Archetype Object Model (AOM), allowing archetypes to be parsed as dADL documents. This makes conversion to and from various XML formats trivial.

Archetype Slot syntax

In ADL2, two kinds of reference are allowed in archetype slots: archetype object references, e.g.

```
@archetype_id ∈ {/.*\.iso-ehr\.section\..*\..*/}
```

and archetype path references, e.g.

```
∃ /subject/relation
```

2.4.2 Version 1.3 from Version 1.2

The specific changes made in version 1.3 of ADL are as follows.

Query syntax replaced by URI data type

In version 1.2 of ADL, it was possible to include an external query, using syntax of the form:

```
attr_name = <query("some_service", "some_query_string")>
```

This is now replaced by the use of URIs, which can express queries, for example:

```
attr_name = <http://some.service.org?some%20query%20etc>
```

No assumption is made about the URI; it need not be in the form of a query - it may be any kind of URI.

Top-level Invariant Section

In this version, invariants can only be defined in a top level block, in a way similar to object-oriented class definitions, rather than on every block in the definition section, as is the case in version 1.2 of ADL. This simplifies ADL and the Archetype Object Model, and makes an archetype more comprehensible as a "type" definition.

2.4.3 Version 1.2 from Version 1.1

ADL Version

The ADL version is now optionally (for the moment) included in the first line of the archetype, as follows.

```
archetype (adl_version=1.2)
```

It is strongly recommended that all tool implementors include this information when archetypes are saved, enabling archetypes to gradually become imprinted with their correct version, for more reliable later processing. The `adl_version` indicator is likely to become mandatory in future versions of ADL.

dADL Syntax Changes

The dADL syntax for container attributes has been altered to allow paths and typing to be expressed more clearly, as part of enabling the use of Xpath-style paths. ADL 1.1 dADL had the following appearance:

```
school_schedule = <
    locations(1) = <...>
    locations(2) = <...>
    locations(3) = <...>
    subjects("philosophy:plato") = <...>
    subjects("philosophy:kant") = <...>
    subjects("art") = <...>
>
```

This has been changed to look like the following:

```
school_schedule = <
    locations = <
        [1] = <...>
        [2] = <...>
        [3] = <...>
    >
    subjects = <
        ["philosophy:plato"] = <...>
        ["philosophy:kant"] = <...>
```

```
        ["art"] = <...>
    >
```

The new appearance both corresponds more directly to the actual object structure of container types, and has the property that paths can be constructed by directly reading identifiers down the backbone of any subtree in the structure. It also allows the optional addition of typing information anywhere in the structure, as shown in the following example:

```
school_schedule = SCHEDULE <
    locations = LOCATION <
        [1] = <...>
        [2] = <...>
        [3] = ARTS_PAVILLION <...>
    >
    subjects = <
        ["philosophy:plato"] = ELECTIVE_SUBJECT <...>
        ["philosophy:kant"] = ELECTIVE_SUBJECT <...>
        ["art"] = MANDATORY_SUBJECT <...>
    >
```

These changes will affect the parsing of container structures and keys in the description and ontology parts of the archetype.

Revision History Section

Revision history is now recorded in a separate section of the archetype, both to logically separate it from the archetype descriptive details, and to facilitate automatic processing by version control systems in which archtypes may be stored. This section is included at the end of the archetype because it is in general a monotonically growing section.

Primary_language and Languages_available Sections

An attribute previously called ‘primary_language’ was required in the ontology section of an ADL 1.1 archetype. This is renamed to ‘original_language’ and is now moved to a new top level section in the archetype called ‘language’. Its value is still expressed as a dADL String attribute. The ‘languages_available’ attribute previously required in the ontology section of the archetype is renamed to ‘translations’, no longer includes the original languages, and is also moved to this new top level section.

2.5 Tools

A validating ADL parser is freely available from http://my.openehr.org/wsvn/oe_distrib/. It has been wrapped for use in Java and Microsoft .Net, and standard C/C++ environments. See the website for the latest status.

3 dADL - Data ADL

The dADL specification is unchanged from the most recent ADL 1.x specification.

4 cADL - Constraint ADL

The cADL specification is unchanged from the most recent ADL 1.x specification.

5 Assertions

The assertion specification is unchanged from the most recent ADL 1.x specification.

6 Declarations

This section for a future release of ADL

To Be Determined: main problem of variables is that they must have names, which are language-dependent; imagine if there were a mixture of variables added by authors in different languages. The only solution is to name them with terms.

To Be Determined: Variables have to be treated as term coordinations, and should be coded e.g. using ccNNNN codes ("cc" = coordinated code). Then they can be given meanings in any language.

6.1 Predefined Variables

A number of predefined variables can be referenced in ADL assertion expressions, without prior definition, including:

- `$current_date`: `Date`; returns the date whenever the archetype is evaluated
- `$current_time`: `Time`; returns time whenever the archetype is evaluated
- `$current_date_time`: `Date_Time`; returns date/time whenever the archetype is evaluated

To Be Continued: these should be coded as well, using openEHR codes

6.2 Archetype-defined Variables

Variables can also be defined inside an archetype, as part of the assertion statements in an invariant. The syntax of variable definition is as follows:

```
let $var_name = reference
```

Here, a reference can be any of the operand types listed above. 'Let' statements can come anywhere in an invariant block, but for readability, should generally come first.

The following example illustrates the use of variables in an invariant block:

```
invariant
    let $sys_bp =
       /data[at9001]/events[at9002]/data[at1000]/items[at1100]
    let $dia_bp =
       /data[at9001]/events[at9002]/data[at1000]/items[at1200]
    $sys_bp >= $dia_bp
```

To Be Continued:

7 ADL Paths

The path specification is unchanged from the most recent ADL 1.x specification.

8 ADL - Archetype Definition Language

8.1 Introduction

This section describes ADL archetypes as a whole, adding a small amount of detail to the descriptions of dADL and cADL already given. The important topic of the relationship of the cADL-encoded `definition` section and the dADL-encoded `ontology` section is discussed in detail. In this section, only standard ADL (i.e. the cADL and dADL constructs and types described so far) is assumed. Archetypes for use in particular domains can also be built with more efficient syntax and domain-specific types, as described in Customising ADL on page 41, and the succeeding sections.

An ADL archetype is a dADL document with the structure shown below:

```
archetype_id = <"some.archetype.id">
adl_version = <"2.0">
is_controlled = <True>
parent_archetype_id = <"some.other.archetype.id">
concept = <[concept_code]>
original_language = <"lang">
translations = <
    ...
>
description = <
    ...
>
definition = (cadl) <#
    cADL plug-in section
#>
invariant = (aadl) <#
    assertions plug-in section
#>
ontology = <
    ...
>
revision_history = <
    ...
>
```

In the above, all top-level attribute names are an exact match in name and type for a corresponding attribute in the *open*EHR. Archetype Object Model (AOM). Optional parts are shown unbolded. In this document, top level attributes are usually described as 'sections' of the archetype.

8.2 Basics

8.2.1 Order of Sections

As of ADL 2.0, order is no longer significant in archetypes. However, it is strongly recommended that the order above be respected, since it is the logical reading order, and some parser implementations may rely on it.

8.2.2 Keywords

ADL has no keywords of its own (i.e, distinct from the keywords in cADL and the invariant language), although depending on how parsers are built, the top-level attribute names might be regarded as special in some implementations.

8.2.3 Node Identification

In the `definition` section of an ADL archetype, a particular scheme of codes is used for node identifiers as well as for denoting constraints on textual (i.e. language dependent) items. Codes are either local to the archetype, or from an external lexicon. This means that the archetype description is the same in all languages, and is available in any language that the codes have been translated to. All term codes are shown in brackets (`[]`). Codes used as node identifiers and defined within the same archetype are prefixed with "at" and by convention have 4 digits, e.g. `[at0010]`. Codes of any length are acceptable in ADL archetypes. Specialisations of locally coded concepts have the same root, followed by "dot" extensions, e.g. `[at0010.2]`. From a terminology point of view, these codes have the implied semantics of subsumption; additionally - the "dot" structuring is used as an optimisation on node identification.

8.2.4 Local Constraint Codes

A second kind of local code is used to stand for constraints on textual items in the body of the archetype. Although these could be included in the main archetype body, because they are language- and/or terminology-sensitive, they are defined in the ontology section, and referenced by codes prefixed by "ac", e.g. `[ac0009]`. As for "at" codes, the convention used in this document is to use 4-digit "ac" codes, even though any number of digits is acceptable. The use of these codes is described in section 8.6.4

8.3 Header Sections

8.3.1 Archetype_id Section

This section introduces the archetype and must include an identifier. A typical `archetype` section is as follows:

```
archetype_id = <"mayo.openehr-ehr-entry.haematology.v1">
```

The multi-axial identifier identifies archetypes in a global space. The syntax of the identifier is described under Archetype Identification on page 9 in The openEHR Archetype System.

8.3.2 Adl_version Section

This mandatory section indicates the version of ADL being used in the archetype. Its value is one of the revision values of this specification, such as "2.0"; it typically looks as follows:

```
adl_version = <"2.0">
```

8.3.3 Is_controlled Section

This is an optional section indicating whether the archetype is change-controlled or not can be included after the version, as follows:

```
is_controlled = <False>
```

The flag may have the values `True` and `False` only, and is an aid to software. Archetypes having `is_controlled` set to True must have the `revision_history` section included, while those with the value `False`, or no flag at all, may omit the `revision_history` section. This enables archetypes to be privately edited in an early development phase without generating large revision histories of little or no value.

8.3.4 Parent_archetype_id Section

This optional section indicates that the archetype is a specialisation of some other archetype, whose identity must be given. Only one specialisation parent is allowed, i.e. an archetype cannot 'multiply-inherit' from other archetypes. An example of declaring specialisation is as follows:

```
archetype_id = <"mayo.openehr-ehr-entry.haematology-cbc.v1">
parent_archetype_id = <"mayo.openehr-ehr-entry.haematology.v1">
```

Here the identifier of the new archetype is derived from that of the parent by adding a new section to its domain concept section. See Archetype Identification on page 9 in The openEHR Archetype System.

8.3.5 Concept Section

All archetypes represent some real world concept, such as a "patient", a "blood pressure", or an "antenatal examination". The concept section describes the overall concept using the code of a concept whose definition is given in the archetype ontology. Like any coded entity, it can be displayed in any language the archetype has been translated to. A typical `concept` section is as follows:

```
concept = <[at0010]>      -- haematology result
```

In this concept definition, the term definition of `[at0010]` contains a more complete description corresponding to the notion implied by the short identifier "`haematology-cbc`" in the example `archteype_id` section shown above.

8.3.6 Original_language and Translations Sections

The `original_language` and `translations` sections include meta-data describing the original language in which the archetype was authored (essential for evaluating natural language quality), and the total list of languages available in the archetype due to translation. There can be only one `original_language`. The `translations` section is optional, but must be present if any translations are present in the `description` or `ontology` sections; if present, it must be updated every time a translation of the archetype is undertaken. The following shows a typical example.

```
original_language = <"en">
translations = <
    ["de"] = <
        author = <"Frederick Smith">
        accreditation = <"British Medical Translator id 00400595">
    >
    ["ru"] = <
        author = <"Vladimir Korotkov">
        accreditation = <"Russion Translator id 892230A">
        other_details = <
            ["email"] = <"vladimir.korotkov@acme.translators.ru">
        >
    >
>
```

Archetypes must always be translated completely or not at all to be valid. This means that when a new translation is made, every language dependent section of the `description` and `ontology` sections has to be translated into the new language, and an appropriate addition made to the `translations` list in the language section.

8.3.7 Description Section

The `description` section of an archetype contains descriptive information, or what some people think of as document "meta-data", i.e. items that can be used in repository indexes and for searching. The dADL syntax is used for the description, as in the following example.

```
description = <
    original_author = <
        ["name"] = <"Dr J Joyce">
        ["organisation"] = <"NT Health Service">
        ["date"] = <2003-08-03>
    >
    lifecycle_state = <"initial">
    archetype_package_uri =
        <"http://www.aihw.org.au/data_sets/diabetic_archetypes.html">
    details = <
        ["en"] = <
            purpose = <"archetype for diabetic patient review">
            use = <"used for all hospital or clinic-based diabetic reviews,
                including first time. Optional sections are removed according
                to the particular review">
            misuse = <"not appropriate for pre-diagnosis use">
            original_resource_uri =
                <"http://www.healthdata.org.au/data_sets/
                                diabetic_review_data_set_1.html">
            other_details = <...>
        >
        ["de"] = <
            purpose = <"Archetyp für die Untersuchung von Patienten
                        mit Diabetes">
            use = <"wird benutzt für alle Diabetes-Untersuchungen im
                  Krankenhaus, inklusive der ersten Vorstellung. Optionale
                  Abschnitte werden in Abhängigkeit von der speziellen
                  Vorstellung entfernt.">
            misuse = <"nicht geeignet für Benutzung vor Diagnosestellung">
            original_resource_uri =
                <"http://www.healthdata.org.au/data_sets/
                                diabetic_review_data_set_1.html">
            other_details = <...>
        >
    >
>
```

A number of details are worth noting here. Firstly, the free hierarchical structuring capability of dADL is exploited for expressing the "deep" structure of the `details` section and its subsections. The design of the objects in this section is specified in the AOM. Secondly, the dADL qualified list form is used to allow multiple translations of the `purpose` and `use` to be shown. Lastly, empty items such as `misuse` (structured if there is data) are shown with just one level of empty brackets. The above example shows meta-data based on the CEN MetaKnow standard, with inclusions from the HL7 Templates Proposal [5] and the meta-data of the SynEx and Australian GeHR archetypes.

Which descriptive items are required will depend on the semantic standards imposed on archetypes by health standards organisations and/or the design of archetype repositories and is not specified by ADL.

8.4 Definition Section

The `definition` section contains the main formal definition of the archetype, and is written in the Constraint Definition Language (cADL). A typical `definition` section is as follows:

```
definition = (cadl) <#
    ENTRY[at0000] ∈ {          -- blood pressure measurement
        name ∈ {               -- any synonym of BP
            CODED_TEXT ∈ {
                code ∈ {
                    CODE_PHRASE ∈ {[ac0001]}
                }
            }
        }
        data ∈ {
            HISTORY[at9001] ∈ {                          -- history
                events cardinality ∈ {1..*} ∈ {
                    EVENT[at9002] occurrences ∈ {0..1} ∈ {-- baseline
                        name ∈ {
                            CODED_TEXT ∈ {
                                code ∈ {
                                    CODE_PHRASE ∈ {[ac0002]}
                                }
                            }
                        }
                        data ∈ {
                            ITEM_LIST[at1000] ∈ {   -- systemic arterial BP
                                items cardinality ∈ {2..*} ∈ {
                                    ELEMENT[at1100] ∈ {              -- systolic BP
                                        name ∈ {       -- any synonym of 'systolic'
                                            CODED_TEXT ∈ {
                                                code ∈ {
                                                    CODE_PHRASE ∈ {[ac0002]}
                                                }
                                            }
                                        }
                                        value ∈ {
                                            QUANTITY ∈ {
                                                magnitude ∈ {|0..1000|}
                                                property ∈ {[properties::0944]}
                                                                        -- "pressure"
                                                units ∈ {[units::387]}  -- "mm[Hg]"
                                            }
                                        }
                                    }
                                    ELEMENT[at1200] ∈ {        -- diastolic BP
                                        name ∈ {       -- any synonym of 'diastolic'
                                            CODED_TEXT ∈ {
                                                code ∈ {
                                                    CODE_PHRASE ∈ {[ac0003]}
                                                }
                                            }
                                        }
                                        value ∈ {
                                            QUANTITY ∈ {
                                                magnitude ∈ {|0..1000|}
                                                property ∈ {[properties::0944]}
                                                                        -- "pressure"
```

```
                                    units ∈ {[units::387]}  -- "mm[Hg]"
                                }
                            }
                        }
                        ELEMENT[at9000] occurrences ∈ {0..*} ∈ {*}
                                                -- unknown new item
                    }
                ...
```

This definition expresses constraints on instances of the types ENTRY, HISTORY, EVENT, LIST_S, ELEMENT, QUANTITY, and CODED_TEXT so as to allow them to represent a blood pressure measurement, consisting of a history of measurement events, each consisting of at least systolic and diastolic pressures, as well as any number of other items (expressed by the [at9000] "any" node near the bottom).

8.4.1 Design-time and Run-time paths

All non-unique sibling nodes in a cADL text which correspond to nodes in data which might be referred to from elsewhere in the archetype, or might be used for querying at runtime, require a node identifier, and it is usually preferable to assign a design-time meaning, enabling paths and queries to be expressed using logical meanings rather than meaningless identifiers. When data are created according to a cADL specification, the archetype node identifiers are written into the data, providing a reliable way of finding data nodes, regardless of what other runtime names might have been chosen by the user for the node in question. There are two reasons for doing this. Firstly, querying cannot rely on runtime names of nodes (e.g. names like "sys BP", "systolic bp", "sys blood press." entered by a doctor are unreliable for querying); secondly, it allows runtime data retrieved from a persistence mechanism to be re-associated with the cADL structure which was used to create it.

An example which clearly shows the difference between design-time meanings associated with node ids and runtime names is the following, for the root node of a SECTION headings hierarchy representing the problem/SOAP headings (a simple heading structure commonly used by clinicians to record patient contacts under top-level headings corresponding to the patient's problem(s), and under each problem heading, the headings "subjective", "objective", "assessment", and "plan").

```
SECTION[at0000] matches {                -- problem
    name matches {
        CODED_TEXT matches {
            code matches {[ac0001]}
        }
    }
```

In the above, the node identifier [at0000] is assigned a meaning such as "clinical problem" in the archetype ontology section. The subsequent lines express a constraint on the runtime *name* attribute, using the internal code [ac0001]. The constraint [ac0001] is also defined in the archetype ontology section with a formal statement meaning "any clinical problem type", which could clearly evaluate to thousands of possible values, such as "diabetes", "arthritis" and so on. As a result, in the runtime data, the node identifier corresponding to "clinical problem" and the actual problem type chosen at runtime by a user, e.g. "diabetes", can both be found. This enables querying to find all nodes with meaning "problem", or all nodes describing the problem "diabetes". Internal [acNNNN] codes are described in Local Constraint Codes on page 28.

8.5 Invariant Section

The invariant section in an ADL archetype introduces assertions which relate to the entire archetype, and can be used to make statements which are not possible within the block structure of the definition section. Any constraint which relates more than one property to another is in this cate-

gory, as are most constraints containing mathematical or logical formulae. Invariants are expressed in the archetype assertion language, described in section 5 on page 23.

An invariant statement is a first order predicate logic statement which can be evaluated to a boolean result at runtime. Objects and properties are referred to using paths.

The following simple example says that the speed in kilometres of some node is related to the speed-in-miles by a factor of 1.6:

```
invariant = (aadl) <#
    validity: /speed[at0002]/kilometres/magnitude =
              /speed[at0004]/miles/magnitude * 1.6
#>
```

Note that in a well-designed archetype, the '1.6' above should be coded and included in the ontology section.

8.6 Ontology Section

8.6.1 Overview

The `ontology` section of an archetype is expressed in dADL, and is where codes representing node IDs, constraints on text or terms, and bindings to terminologies are defined. Linguistic language translations are added in the form of extra blocks keyed by the relevant language. The following example shows the layout of this section.

```
ontology = <
    terminologies_available = <"snomed_ct", ...>
    term_definitions = <
        ["en"] = <
            ["at0000"] = <...>
            ...
        >
        ["de"] = <
            ["at0000"] = <...>
            ...
        >
    >
    term_binding = <
        ["snomed_ct"] = <
            ["at0000"] = <...>
            ...
        >
        ...
    >
    constraint_definitions = <
        ["en"] = <...>
            ["ac0001"] = <...>
            ...
        ["de"] = <
            ["ac0001"] = <...>
            ...
        >
        ...
    >
    constraint_binding = <
        ["snomed_ct"] = <...>
```

```
            ...
        >
    >
```

The `term_definitions` section is mandatory, and must be defined for each translation carried out.

Each of these sections can have its own meta-data, which appears within description sub-sections, such as the one shown above providing translation details.

8.6.2 Terminologies_available Sub-section

This section provides the total list of external terminologies which are mentioned anywhere else in the ontology.

```
terminologies_available = <"snomed_ct", ...>
```

8.6.3 Term_definitions Sub-section

This section is where all archetype local terms (that is, terms of the form `[atNNNN]`) are defined. The following example shows an extract from the English and German term definitions for the archetype local terms in a problem/SOAP headings archetype. Each term is defined using a structure of name/value pairs, and mustat least include the names “text” and “description”, which are akin to the usual rubric, and full definition found in terminologies like SNOMED-CT. Each term object is then included in the appropriate language list of term definitions, as shown in the example below.

```
term_definitions = <
    ["en"] = <
        ["at0000"] = <
            text = <"problem">
            description = <"The problem experienced by the subject
                of care to which the contained information relates">
        >
        ["at0001"] = <
            text = <"problem/SOAP headings">
            description = <"SOAP heading structure for multiple problems">
        >
        ...
        ["at4000"] = <
            text = <"plan">
            description = <"The clinician's professional advice">
        >
    >
    ["de"] = <
        ["at0000"] = <
            text = <"klinisches Problem">
            description = <"Das Problem des Patienten worauf sich diese \
                    Informationen beziehen">
        >
        ["at0001"] = <
            text = <"Problem/SOAP Schema">
            description = <"SOAP-Schlagwort-Gruppierungsschema fuer
                  mehrfache Probleme">
        >
        ["at4000"] = <
            text = <"Plan">
            description = <"Klinisch-professionelle Beratung des Pflegenden">
        >
    >
>
```

In some cases, term definitions may have been lifted from existing terminologies. This is only a safe thing to do if the following conditions apply:

- if there is a lexical match between the intended term in the archetype and the preferred term or a synonym in the terminology;
- the definitions *exactly* match the need in the archetype;
- if the classification of the term in the terminology would classify real world instances (e.g. people with a certain illness) the same way as intended by the archetype design.

To indicate where definitions come from, a "provenance" tag can be used, as follows:

```
["at4000"] = <
    text = <"plan">;
    description = <"The clinician's professional advice">;
    provenance = <"ACME_terminology(v3.9a)">
>
```

Note that this does not indicate a *binding* to any term, only its origin. Bindings are described in section 8.6.5 and section 8.6.6.

8.6.4 Constraint_definitions Sub-section

The `constraint_definition` section is of exactly the same form as the `term_definition` section, and provides the definitions - i.e. the meanings - of the local constraint codes, which are of the form `[acNNNN]`. Each such code refers to some constraint such as "any term which is a subtype of 'hepatitis' in the ICD9AM terminology"; the constraint definitions do not provide the constraints themselves, but define the *meanings* of such constraints, in a manner comprehensible to human beings, and usable in GUI applications. This may seem a superfluous thing to do, but in fact it is quite important. Firstly, term constraints can only be expressed with respect to particular terminologies - a constraint for "kind of hepatitis" would be expressed in different ways for each terminology which the archetype is bound to. For this reason, the actual constraints are defined in the `constraint_binding` section. An example of a constraint term definition for the hepatitis constraint is as follows:

```
["ac1015"] = <
    text = <"type of hepatitis">
    description = <"any term which means a kind of viral hepatitis">
>
```

Note that while it often seems tempting to use classification codes, e.g. from the ICD vocabularies, these will rarely be much use in terminology or constraint definitions, because it is nearly always *descriptive*, not classificatory terms which are needed.

8.6.5 Term_binding Sub-section

This section is used to describe the equivalences between archetype local terms and terms found in external terminologies. The purpose is solely for allowing query engine software which wants to search for an instance of some external term to determine what equivalent to use in the archetype. Note that this is distinct from the process of embedding mapped terms in runtime data, which is also possible with the data models of HL7v3, openEHR, and CEN 13606.

A typical term binding section resembles the following:

```
term_binding = <
    ["umls"] = <
       ["at0000"] = <[umls::C124305]> -- apgar result
       ["at0002"] = <[umls::0000000]> -- 1-minute event
       ["at0004"] = <[umls::C234305]> -- cardiac score
```

```
            ["at0005"] = <[umls::C232405]> -- respiratory score
            ["at0006"] = <[umls::C254305]> -- muscle tone score
            ["at0007"] = <[umls::C987305]> -- reflex response score
            ["at0008"] = <[umls::C189305]> -- color score
            ["at0009"] = <[umls::C187305]> -- apgar score
            ["at0010"] = <[umls::C325305]> -- 2-minute apgar
            ["at0011"] = <[umls::C725354]> -- 5-minute apgar
            ["at0012"] = <[umls::C224305]> -- 10-minute apgar
        >
    >
```

Each entry simply indicates which term in an external terminology is equivalent to the archetype internal codes. Note that not all internal codes necessarily have equivalents: for this reason, a terminology binding is assumed to be valid even if it does not contain all of the internal codes.

8.6.6 Constraint_binding Sub-section

The last of the ontology sections formally describes text constraints from the main archetype body. They are described separately because they are terminology dependent, and because there may be more than one for a given logical constraint. A typical example follows:

```
constraint_binding = <
    ["snomed_ct"] = <
        ["ac0001"] = <http://terminology.org?terminology_id=snomed_ct&&
            has_relation=[102002];with_target=[128004]>
        ["ac0002"] = <http://terminology.org?terminology_id=snomed_ct&&
            synonym_of=[128025]>
    >
>
```

In this example, each local constraint code is formally defined to refer to the result of a query to a service, in this case, a terminology service which can interrogate the Snomed-CT terminology.

8.7 Revision_history Section

The `revision_history` section of an archetype shows the audit history of changes to the archetype, and is expressed in dADL syntax. It is optional, and is included at the end of the archetype, since it does not contain content of direct interest to archetype authors, and will monotonically grow in size. Where archetypes are stored in a version-controlled repository such as CVS or an equivalent commercial product, the revision history section would normally be regenerated each time by the authoring software, e.g. via processing of the output of the 'prs' command used with SCCS files, or 'rlog' for RCS files. The following shows a typical example, with entries in most-recent-first order (although technically speaking, the order is irrelevant to ADL).

```
revision_history = <
    ["1.57"] = <
        committer = <"Miriam Hanoosh">
        committer_organisation = <"AIHW.org.au">
        time_committed = <2004-11-02 09:31:04+1000>
        revision = <"1.2">
        reason = <"Added social history section">
        change_type = <"Modification">
    >
    -- etc
    ["1.1"] = <
        committer = <"Enrico Barrios">
        committer_organisation = <"AIHW.org.au">
        time_committed = <2004-09-24 11:57:00+1000>
```

```
            revision = <"1.1">
            reason = <"Updated HbA1C test result reference">
            change_type = <"Modification">
        >
        ["1.0"] = <
            committer = <"Enrico Barrios">
            committer_organisation = <"AIHW.org.au">
            time_committed = <2004-09-14 16:05:00+1000>
            revision = <"1.0">
            reason = <"Initial Writing">
            change_type = <"Creation">
        >
    >
```

8.8 Validity Rules

This section describes the formal (i.e. checkable) semantics of ADL archetypes. It is recommended that parsing tools use the identifiers published here in their error messages, as an aid to archetype designers.

8.8.1 Global Archetype Validity

The following validity constraints apply to an archetype as a whole. Note that the term "section" means the same as "attribute" in the following, i.e. a section called "definition" in a dADL text is a serialisation of the value for the attribute of the same name.

VARID: archetype identifier validity. The archetype must have an identifier value for the archetype_id section. The identifier must conform to the published *open*EHR specification for archetype identifiers.

VARCN: archetype concept validity. The archetype must have an archetype term value in the concept section. The term must exist in the archetype ontology.

VARDF: archetype definition validity. The archetype must have a definition section, expressed as a cADL syntax string, or in an equivalent plug-in syntax.

VARON: archetype ontology validity. The archetype must have an ontology section, expressed as a cADL syntax string, or in an equivalent plug-in syntax.

VARDT: archetype definition typename validity. The topmost typename mentioned in the archetype definition section must match the type mentioned in the type-name slot of the first segment of the archetype id.

8.8.2 Coded Term Validity

All node identifiers ('at' codes) used in the `definition` part of the archetype must be defined in the `term_definitions` part of the ontology.

VATDF: archetype term validity. Each archetype term used as a node identifier the archetype definition must be defined in the term_definitions part of the ontology.

All constraint identifiers ('ac' codes) used in the `definition` part of the archetype must be defined in the `constraint_definitions` part of the ontology.

VACDF: node identifier validity. Each constraint code used in the archetype definition part must be defined in the constraint_definitions part of the ontology.

8.8.3 Definition Section

The following constraints apply to the `definition` section of the archetype.

VDFAI: archetype identifier validity in definition. Any archetype identifier mentioned in an archetype slot in the definition section must conform to the published *open*EHR specification for archetype identifiers.

VDFPT: **path validity in definition**. Any path mentioned in the definition section must be valid syntactically, and a valid path with respect to the hierarchical structure of the definition section.

9 The ADL Parsing Process

This sectoin is unchanged from the most recent ADL 1.x specification.

10 Customising ADL

This section is unchanged from the most recent ADL 1.x specification.

11 Relationship of ADL to Other Formalisms

This section is unchanged from the most recent ADL 1.x specification.

A 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 openEHR Archetype Object Model. See http://www.openehr.org/repositories/spec-dev/latest/publishing/architecture/archetypes/archetype_model/REV_HIST.html.

4 Beale T. *A Short Review of OCL*. See http://www.deepthought.com.au/it/ocl_review.html.

5 Dolin R, Elkin P, Mead C *et al*. *HL7 Templates Proposal*. 2002. Available at http://www.hl7.org.

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

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

8 Hein J L. *Discrete Structures, Logic and Computability (2nd Ed)*. Jones and Bartlett 2002.

9 Kilov H, Ross J. *Information Modelling: an Object-Oriented Approach*. Prentice Hall 1994.

10 Gruber T R. *Toward Principles for the Design of Ontologies Used for Knowledge Sharing*. in Formal Ontology in Conceptual Analysis and Knowledge Representation. Eds Guarino N, Poli R. Kluwer Academic Publishers. 1993 (Aug revision).

11 Martin P. Translations between UML, OWL, KIF and the WebKB-2 languages (For-Taxonomy, Frame-CG, Formalized English). May/June 2003. Available at http://meganesia.int.gu.edu.au/~phmartin/WebKB/doc/model/comparisons.html as at Aug 2004.

12 Meyer B. *Eiffel the Language (2nd Ed)*. Prentice Hall, 1992.

13 Patel-Schneider P, Horrocks I, Hayes P. *OWL Web Ontology Language Semantics and Abstract Syntax*. See http://w3c.org/TR/owl-semantics/.

14 Smith G. *The Object Z Specification Language*. Kluwer Academic Publishers 2000. See http://www.itee.uq.edu.au/~smith/objectz.html.

15 Sowa J F. *Knowledge Representation: Logical, philosophical and Computational Foundations*. 2000, Brooks/Cole, California.

Resources

16 HL7 v3 RIM. See http://www.hl7.org.

17 *open*EHR. EHR Reference Model. See http://svn.openehr.org/specification/TRUNK/project_page.htm.

18 Perl Regular Expressions. See http://www.perldoc.com/perl5.6/pod/perlre.html.

19 SynEx project, UCL. http://www.chime.ucl.ac.uk/HealthI/SynEx/.

20 W3C. *OWL - the Web Ontology Language*. See http://www.w3.org/TR/2003/CR-owl-ref-20030818/.

21 W3C. XML Path Language. See http://w3c.org/TR/xpath.

END OF DOCUMENT