SLOVENSKI STANDARD

SIST EN 13606-2:2008

01-februar-2008

Nadomeġļa:

SIST ENV 13606-2:2003

Zdravstvena informatika - Komunikacija z elektronskimi zapisi v zdravstvenem varstvu - 2. del: Arhetipi

Health informatics - Electronic health record communication - Part 2: Archetypes interchange specification

Medizinische Informatik - Kommunikation von Patientendaten in elektronischer Form - Teil 2: Archetypen

Informatique de santé - Dossier de santé informatisé communicant - Spécification des échanges des archétypes

Ta slovenski standard je istoveten z: **EN 13606-2:2007**

ICS:

35.240.80	Uporabniške rešitve IT v zdravstveni tehniki	IT applications in health care technology

SIST EN 13606-2:2008 **en**

EUROPEAN STANDARD

NORME EUROPÉENNE

EUROPÄISCHE NORM

EN 13606-2

August 2007

ICS 35.240.80

English Version

Health informatics - Electronic health record communication - Part 2: Archetypes interchange specification

Informatique de santé - Dossier de santé informatisé communicant - Spécification des échanges des archétypes

Medizinische Informatik - Kommunikation von Patientendaten in elektronischer Form - Teil 2: Spezifikation für den Austausch von Archetypen

This European Standard was approved by CEN on 21 June 2007.

CEN members are bound to comply with the CEN/CENELEC Internal Regulations which stipulate the conditions for giving this European Standard the status of a national standard without any alteration. Up-to-date lists and bibliographical references concerning such national standards may be obtained on application to the CEN Management Centre or to any CEN member.

This European Standard exists in three official versions (English, French, German). A version in any other language made by translation under the responsibility of a CEN member into its own language and notified to the CEN Management Centre has the same status as the official versions.

CEN members are the national standards bodies of Austria, Belgium, Bulgaria, Cyprus, Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, Netherlands, Norway, Poland, Portugal, Romania, Slovakia, Slovenia, Spain, Sweden, Switzerland and United Kingdom.

EUROPEAN COMMITTEE FOR STANDARDIZATION
COMITÉ EUROPÉEN DE NORMALISATION
EUROPÄISCHES KOMITEE FÜR NORMUNG

Management Centre: rue de Stassart, 36 B-1050 Brussels

Ref. No. EN 13606-2:2007: E

Contents

Page

Figures

Foreword

This document (EN 13606-2:2007) has been prepared by Technical Committee CEN/TC 251 “Health informatics”, the secretariat of which is held by NEN.

This document shall be given the status of a national standard, either by publication of an identical text or by endorsement, at the latest by February 2008 and conflicting national standards shall be withdrawn at the latest by February 2008.

This document will supersede ENV 13606-2:2000.

This multipart standard under the general heading Health informatics – Electronic health record communication consists of the following parts:

Part 1: Reference model

Part 2: Archetypes interchange specification

Part 3: Reference archetypes and term lists

Part 4: Security

Part 5: Exchange models

According to the CEN/CENELEC Internal Regulations, the national standards organizations of the following countries are bound to implement this European Standard: Austria, Belgium, Bulgaria, Cyprus, Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, Netherlands, Norway, Poland, Portugal, Romania, Slovakia, Slovenia, Spain, Sweden, Switzerland and United Kingdom.

0 Introduction

Comprehensive, multi-enterprise and longitudinal electronic health records will often in practice be achieved through the joining up of multiple clinical applications, databases (and increasingly devices) that are each tailored to the needs of individual conditions, specialties or enterprises.

This requires that EHR data from diverse systems be capable of being mapped to and from a single comprehensive representation, which is used to underpin interfaces and messages within a distributed network (federation) of EHR systems and services. This common representation has to be sufficiently generic and rich to represent any conceivable health record data, comprising part or all of an EHR (or a set of EHRs) being communicated.

The approach adopted in this standard, underpinned by international research on the EHR, has been to define a rigorous and generic Reference Model that is suitable for all kinds of data and data structures within an EHR, and in which all labelling and context information is an integral part of each construct. An EHR Extract will contain all of the names, structure and context required for it to be interpreted faithfully on receipt even if its organisation and the nature of the clinical content have not been “agreed” in advance.

However the wide-scale sharing of health records, and their meaningful analysis across distributed sites, also requires that a consistent approach is used for the clinical (semantic) data structures that will be communicated via the Reference Model, so that equivalent clinical information is represented consistently. This is necessary in order for clinical applications and analysis tools safely to process EHR data that have come from heterogeneous sources.

0.1 Archetypes

The challenge for EHR interoperability is therefore to devise a generalised approach to representing every conceivable kind of health record data structure *in a consistent way*. This needs to cater for records arising from any profession, speciality or service, whilst recognising that the clinical data sets, value sets, templates etc. required by different health care domains will be diverse, complex and will change frequently as clinical practice and medical knowledge advance. This requirement is part of the widely acknowledged health informatics challenge of *semantic interoperability*.

The approach adopted by this standard distinguishes a Reference Model, used to represent the generic properties of health record information, and Archetypes (conforming to an Archetype Model), which are meta-data used to define patterns for the specific characteristics of the clinical data that represent the requirements of each particular profession, speciality or service.

The Reference Model is specified as an ODP Information Viewpoint model, representing the global characteristics of health record components, how they are aggregated, and the context information required to meet ethical, legal and provenance requirements. In this standard, the Reference Model is defined in Part 1. This model defines the set of classes that form the generic building blocks of the EHR. It reflects the stable characteristics of an electronic health record, and would be embedded in a distributed (federated) EHR environment as specific messages or interfaces (as specified in Part 5 of this standard).

Archetypes are effectively pre-coordinated combinations of named RECORD_COMPONENT hierarchies that are agreed within a community in order to ensure semantic interoperability, data consistency and data quality.

For an EHR_Extract as defined in Part 1 of this standard, an archetype specifies (and effectively constrains) a particular hierarchy of RECORD_COMPONENT sub-classes, defining or constraining their names and other relevant attribute values, optionality and multiplicity at any point in the hierarchy, the data types and value ranges that ELEMENT data values may take, and may include other dependency constraints. Archetype instances themselves conform to a formal model, known as an Archetype Model (which is a constraint model, also specified as an ODP Information Viewpoint Model). Although the Archetype Model is stable, individual archetype instances may be revised or succeeded by others as clinical practice evolves. Version control ensures that new revisions do not invalidate data created with previous revisions.

Archetypes may be used within EHR systems to govern the EHR data committed to a repository. However, for the purposes of this interoperability standard, no assumption is made about the use of archetypes within the EHR Provider system whenever this standard is used for EHR communication. It is assumed that the

original EHR data, if not already archetyped, may be mapped to a set of archetypes, if desired, when generating the EHR_EXTRACT.

The Reference Model defined in Part 1 of this standard has attributes that can be used to specify the archetype to which any RECORD_COMPONENT within an EHR_EXTRACT conforms. The class RECORD_COMPONENT includes an attribute *archetype_id* to identify the archetype and node to which that RECORD_COMPONENT conforms. The *meaning* attribute, in the case of archetyped data, refers to the primary concept to which the corresponding archetype node relates. However, it should be noted that Part 1 does not require that archetypes are used to govern the hierarchy of RECORD_COMPONENTS within an EHR_EXTRACT: the archetype-related attributes are optional in that model. It is recognised that the international adoption of an archetype approach will be gradual, and may take some years.

0.2 Archetype Repositories

The range of archetypes required within a shared EHR community will depend upon its range of clinical activities. The total set needed on a national basis is presently unknown, but there might eventually be several thousand archetypes globally. The ideal sources of knowledge for developing such archetype definitions will be clinical guidelines, care pathways, scientific publications and other embodiments of best practice. However, “de facto” sources of agreed clinical data structures might also include:

- the data schemata (models) of existing clinical systems;
- the lay-out of computer screen forms used by these systems for data entry and for the display of analyses performed;
- data-entry templates, pop-up lists and look-up tables used by these systems;
- shared-care data sets, messages and reports used locally and nationally;
- the structure of forms used for the documentation of clinical consultations or summaries within paper records;
- health information used in secondary data collections;
- the pre-coordinated terms in terminology systems.

Despite this list of *de facto* ways in which clinical data structures are currently represented, these formats are very rarely interoperable. The use of standardised archetypes provides an interoperable way of representing and sharing these specifications, in support of consistent (good quality) health care record-keeping and the semantic interoperability of shared EHRs.

In the longer term, it is anticipated that the involvement of national health services, academic organisations and professional bodies in the development of archetypes will enable this approach to contribute to the pursuit of quality evidence-based clinical practice. In the future regional or national public domain libraries of archetype definitions might be accessed via the Internet, and downloaded for local use within EHR systems.

0.3 Communicating Archetypes

This part standard specifies the requirements for a comprehensive and interoperable archetype representation, and defines the ODP Information Viewpoint representation for the Archetype Model and an optional archetype interchange format called Archetype Definition Language (ADL).

This standard does not require that any particular model be adopted as the internal architecture of archetype repositories, services or components used to author, store or deploy archetypes in collaboration with EHR services. It does require that these archetypes are capable of being mapped to the Archetype Model defined in this part-standard in order to support EHR communication and interoperability within an EHR-sharing community.

0.4 Overview of the Archetype Model

This section provides a general informative description of the model that is specified in Clause 7 of this part standard.

The overall archetype model consists of identifying information, a description (its meta-data), a definition (expressed in terms of constraints on instances of an object model), and an ontology. Identifying information and lifecycle state are part of the *ARCHETYPE* class. The archetype description is separated into revision history information and descriptive information about the archetype. Revision history information is concerned with the committal of the archetype to a repository, and takes the form of a list of audit trail items, while descriptive information describes the archetype itself (regardless of whether it has been committed to a repository of any kind).

The archetype definition, the `main' part of the archetype model, is an instance of a *C_COMPLEX_OBJECT*, since the root of the constraint structure of an archetype shall always take the form of a constraint on a non-primitive object type. The fourth main part of the archetype model, the ontology, is represented by its own class, and is what allows the archetypes to be natural language- and terminology-neutral.

An enumeration class, *VALIDITY_KIND* is also included in the Archetype package. This is intended to be used as the type of any attribute in this constraint model whose value is logically `mandatory', `optional', or `disallowed'. It is used in this model in the classes *C_Date*, *C_Time* and *C_Date_Time*

Archetypes contain some natural language elements, including the description and ontology definitions. Every archetype is therefore created in some original language, which is recorded in the *original_language* attribute of the *ARCHETYPE* class. An archetype is translated by doing the following:

- translating every language-dependent element to the new language;
- adding a new *TRANSLATION_DETAILS* instance to *ARCHETYPE.translations*, containing details about the translator, organisation, quality assurance and so on.

The *languages_available* function provides a complete list of languages in the archetype.

The Archetype Definition

The main definitional part of an archetype consists of alternate layers of object and attribute constraining nodes, each containing the next level of nodes. In this section, the word `attribute' refers to any data property of a class, regardless of whether regarded as a `relationship' (i.e. association, aggregation, or composition) or `primitive' (i.e. value) attribute. At the leaves are primitive object constrainer nodes constraining primitive types such as String, Integer etc. There are also nodes that represent internal references to other nodes, constraint reference nodes that refer to a text constraint in the constraint binding part of the archetype ontology, and archetype constraint nodes, which represent constraints on other archetypes allowed to appear at a given point. The full list of node types is as follows:

- C_complex_object: any interior node representing a constraint on instances of some non-primitive type, e.g. ENTRY, SECTION ;
- C_attribute: a node representing a constraint on an attribute (i.e. UML `relationship' or `primitive attribute') in an object type;
- C_primitive_object: an node representing a constraint on a primitive (built-in) object type;
- Archetype_internal_ref: a node that refers to a previously defined object node in the same archetype. The reference is made using a path;
- Constraint_ref: a node that refers to a constraint on (usually) a text or coded term entity, which appears in the ontology section of the archetype, and in ADL, is referred to with an "acNNNN" code. The constraint is expressed in terms of a query on an external entity, usually a terminology or ontology;
- Archetype_slot: a node whose statements define a constraint that determines which other archetypes may appear at that point in the current archetype. Logically it has the same semantics as a C_COMPLEX_OBJECT, except that the constraints are expressed in another archetype, not the current one.

The Archetype Description

What is normally considered the `meta-data' of an archetype, i.e. its author, date of creation, purpose, and other descriptive items, is described by the *ARCHETYPE_DESCRIPTION* and *ARCHETYPE_DESCRIPTION_ITEM* classes. The parts of this that are in natural language, and therefore may require translated versions, are represented in instances of the *ARCHETYPE_DESCRIPTION_ITEM* class. If an *ARCHETYPE_DESCRIPTION* has more than one *ARCHETYPE_DESCRIPTION_ITEM*, each of these should carry exactly the same information in a different natural language.

When an archetype is translated for use in another language environment, each ARCHETYPE_DESCRIPTION_ITEM needs to be copied and translated into the new language.

The *AUDIT_DETAILS* class is concerned with the creation and modification of the archetype in a repository. Each instance of this class in an actual archetype represents one act of committal to the repository, with the attributes documenting who, when and why.

NOTE Revision of an archetype should be limited to modification of the descriptive information, adding language translations and/or term bindings. If the definition part of an archetype is no longer valid it should instead be replaced with a new archetype to ensure that corresponding EHR data instances each conform to the same archetype specification.

The Archetype Ontology

All linguistic entities are defined in the ontology part of the archetype. There are four major parts in an archetype ontology: term definitions, constraint definitions, term bindings and constraint bindings. The former two define the meanings of various terms and textual constraints which occur in the archetype; they are indexed with unique identifiers, which are used within the archetype definition body. The latter two ontology sections describe the mappings of terms used internally to external terminologies.

Archetype Specialisation

Archetypes may be specialised: an archetype is considered a specialisation of another archetype if it specifies that archetype as its parent, and only makes changes to its definition such that its constraints are `narrower' than those of the parent. Any data created via the use of the specialised archetype shall be conformant both to it and to its parent.

Every archetype has a `specialisation depth'. Archetypes with no specialisation parent have depth 0, and specialised archetypes add one level to their depth for each step down a hierarchy required to reach them.

Archetype Composition

Archetypes may be composed to form larger structures semantically equivalent to a single large archetype. Archetype slots are the means of composition, and are themselves defined in terms of constraints.

Data types and the Support package

The model is dependent on three groups of assumed types, whose names and assumed semantics are described by ISO/IEC 11404. The first group comprises the most basic types:

- Any
- Boolean
- Character
- Integer
- Real
- Double
- String

The second is the assumed library types:

- Date
- Time
- Date_Time
- Duration

These types are supported in most implementation technologies, including XML, Java and other programming languages. They are not defined in this specification, allowing them to be mapped to the most appropriate concrete types in each implementation technology.

The third group is the Generic types:

- List<T> (ordered, duplicates allowed)
- Set<T> (unordered, no duplicates)
- Hash <T, K > (keyed list of items of any type)
- Interval <T> (interval of instances of any ordered type)

Although these types are supported in most implementation technologies, they are not yet represented in UML. The semantics of these types is therefore defined in the Generic_Types package of the UML model. The remaining necessary types are defined in the Support Package of the Archetype Model.

- ARCHETYPE_ID
- HIER_OBJECT_ID
- TERMINOLOGY_ID
- CODE_PHRASE
- CODED_TEXT

The support package also includes two enumeration classes to provide controlled datasets needed by this part standard.

The Constraint Model Package

Any archetype definition is an instance of a *C_COMPLEX_OBJECT*, which expresses constraints on a class in the underlying Reference Model (Part 1 of this standard), recorded in the attribute *rm_type_name.* A *C_COMPLEX_OBJECT* consists of attributes of type *C_ATTRIBUTE*, which are constraints on the attributes (i.e. any property, including relationships) of that Reference Model class. Accordingly, each *C_ATTRIBUTE* records the name of the constrained attribute (in *rm_attribute_name)*, the existence and cardinality expressed by the constraint (depending on whether the attribute it constrains is a multiple or single relationship), and the constraint on the object to which this *C_ATTRIBUTE* refers via its *children* attribute (according to its reference model) in the form of further *C_OBJECTs*.

The key subtypes of *C_OBJECT*, are:

- *C_COMPLEX_OBJECT*;
- *C_PRIMITIVE_OBJECT;*
- *ARCHETYPE_SLOT*.

ARCHETYPE_INTERNAL_REF and *CONSTRAINT_REF* are used to express, respectively, a `slot' where further archetypes may be used to continue describing constraints; a reference to a part of the current archetype that expresses exactly the same constraints needed at another point; and a reference to a constraint on a constraint defined in the archetype ontology, which in turn points to an external knowledge resource, such as a terminology.

The effect of the model is to create archetype description structures that are a hierarchical alternation of object and attribute constraints. The repeated object/attribute hierarchical structure of an archetype provides the basis for using *paths* to reference any node in an archetype. Archetype paths follow a syntax that is a subset of the W3C Xpath syntax.

All Node Types

All nodes in an archetype constraint structure are instances of the supertype *ARCHETYPE_CONSTRAINT*, which provides a number of important common features to all nodes.

The any_allowed Boolean, if True, indicates that any value permitted by the reference model for that attribute is allowed by the archetype; its use permits the logical idea of a completely "open" constraint to be simply expressed, avoiding the need for any further substructure.

Attribute Nodes

Constraints on attributes are represented by instances of the two subtypes of *C_ATTRIBUTE*: *C_SINGLE_ATTRIBUTE* and *C_MULTIPLE_ATTRIBUTE*. For both subtypes, the common constraint is whether the corresponding instance (defined by the *rm_attribute_name* attribute) must exist. Both subtypes have a list of children, representing constraints on the object value(s) of the attribute.

Single-valued attributes are constrained by instances of the type *C_SINGLE_ATTRIBUTE*, which uses the children to represent multiple alternative object constraints for the attribute value.

Multiply-valued attributes are constrained by an instance of *C_MULTIPLE_ATTRIBUTE*, which allows multiple co-existing member objects of the container value of the attribute to be constrained, along with a cardinality constraint, describing ordering and uniqueness of the container.

Cardinality is only required for container objects such as *List<T>*, *Set<T>* and so on, whereas *existence* is always required. If both are used, the meaning is as follows: the existence constraint says whether the container object will be there (at all), while the cardinality constraint says how many items shall be in the container, and whether it acts logically as a list, set or bag.

Primitive Types

Constraints on primitive types are defined by the classes inheriting from *C_PRIMITIVE*, namely *C_STRING*, *C_INTEGER* and so on.

Constraint References

A *CONSTRAINT_REF* is a proxy for a set of constraints on an object that would normally occur at a particular point in the archetype as a *C_COMPLEX_OBJECT*, but where the actual definition of the constraint is expressed as the binding to a query or expression into an external service (such as an ontology or terminology service) e.g.:

- a set of allowed *CODED_TERMs* e.g. the types of hepatitis;
- an *INTERVAL<QUANTITY>* forming a reference range;
- a set of units or properties or other numerical item.

Assertions

The *C_ATTRIBUTE* and subtypes of *C_OBJECT* enable constraints to be expressed in a structural fashion. In addition to this, any instance of a *C_COMPLEX_OBJECT* may include one or more *invariants*. Invariants are statements in a form of predicate logic, which may be used to state constraints on parts of an object. They are not needed to state constraints on a single attribute (since this can be done with an appropriate *C_ATTRIBUTE*), but are necessary to state constraints on more than one attribute, such as a constraint that `systolic pressure should be >= diastolic pressure' in a blood pressure measurement archetype. Such invariants may be expressed using a syntax derived from the OMG's OCL syntax.

Assertions are also used in *ARCHETYPE_SLOTs*, in order to express the `included' and `excluded' archetypes for the slot.

Assertions are modelled as a generic expression tree of unary prefix and binary infix operators.

Node_id and Paths

The node_id attribute in the class *C_OBJECT* and inherited to all subtypes has two functions:

- it allows archetype object constraint nodes to be individually identified, and in particular, guarantees sibling node unique identification;
- it is the main link between the archetype definition (i.e. the constraints) and the archetype ontology, because each *node_id* is a `term code' in the ontology.

The existence of *node_ids* in an archetype is what allows archetype paths to be created, which refer to each node.

Domain-specific Extensions

The main part of the archetype constraint model allows any type in a reference model to be archetyped - i.e. constrained - in a standard way: by a regular cascade of *C_COMPLEX_OBJECT* / *C_ATTRIBUTE* / *C_PRIMITIVE_OBJECT* objects. However, lower level logical `leaf' types may need special constraint semantics that are not conveniently achieved with the standard approach. To enable such classes to be integrated into the generic constraint model, the class *C_DOMAIN_TYPE* is included. This enables the creation of specific " *C_* " classes, inheriting from *C_DOMAIN_TYPE*, which represent custom semantics for particular reference model types. For example, a class called *C_QUANTITY* might be created which has different constraint semantics from the default effect of a *C_COMPLEX_OBJECT* / *C_ATTRIBUTE* cascade representing such constraints in the generic way (i.e. systematically based on the reference model).

Assumed Values

When archetypes are defined to have optional parts, an ability to define `assumed' values is useful. For example, an archetype for the concept `blood pressure measurement' might contain an optional fragment describing the patient position, with choices `lying', `sitting' and `standing'. Since that part of the ENTRY is optional, data could be created according to the archetype that does not contain this information. However, a blood pressure cannot be taken without the patient in some position, so it may be clinically valid to define an implied or `assumed' value. The archetype allows this to be explicitly stated so that all users/systems know what value to assume when optional items are not included in the data. Assumed values are definable at the leaf level only and may be specified in the *C_PRIMITIVE* classes.

The notion of assumed values is distinct from that of `default values': default values do appear in data, while assumed values don't.

The Assertion Package

Assertions are expressed in archetypes in typed first-order predicate logic (FOL). They are used in two places: to express archetype slot constraints, and to express invariants in complex object constraints. In both of these places, their role is to constrain something inside the archetype. Constraints on external resources such as terminologies are expressed in the constraint binding part of the archetype ontology.

The concrete syntax of assertion statements in archetypes is designed to be compatible with the OMG Object Constraint Language (OCL). Archetype assertions are statements that contain the following elements:

- variables, which are attribute names, or ADL paths terminating in attribute names (i.e. equivalent of referencing class feature in a programming language);
- manifest constants of any primitive type, plus date/time types
- arithmetic operators: +, *, -, /, ^ (exponent)
- relational operators: >, <, >=, <=, =, !=, matches
- Boolean operators: not, and, or, xor

- quantifiers applied to container variables: for_all, exists

The Primitives Package

Ultimately any archetype definition will devolve down to leaf node constraints on instances of primitive types. The primitives package defines the semantics of constraint on such types. Most of the types provide at least two alternative ways to represent the constraint; for example the *C_DATE* type allows the constraint to be expressed in the form of a pattern or an *Interval<Date>*.

The Ontology Package

All linguistic and terminological entities in an archetype are represented in the ontology part of an archetype, whose semantics are given in the Ontology package.

An archetype ontology comprises of the following things.

- A list of terms defined local to the archetype. These are identified by `atNNNN' codes, and perform the function of archetype node identifiers from which paths are created. There is one such list for each natural language in the archetype.

- A list of external constraint definitions, identified by `acNNNN' codes, for constraints defined external to the archetype, and referenced using an instance of a *CONSTRAINT_REF*. There is one such list for each natural language in the archetype.

- Optionally, a set of one or more bindings of term definitions to term codes from external terminologies.

- Optionally, a set of one or more bindings of the external constraint definitions to external resources such as terminologies.

Specialisation Depth

Any given archetype occurs at some point in a hierarchy of archetypes related by specialisation, where the depth is indicated by the *specialisation_depth* attribute. An archetype which is not a specialisation of another has a specialisation_depth of 0. Term and constraint codes introduced in the ontology of specialised archetypes (i.e. which did not exist in the ontology of the parent archetype) are defined in a strict way, using `.' (period) markers. For example, an archetype of specialisation depth 2 will use term definition codes like the following:

- `at0.0.1' - a new term introduced in this archetype, which is not a specialisation of any previous term in any of the parent archetypes;

- `at0001.0.1' - a term which specialises the `at0001' term from the top parent. An intervening `.0' is required to show that the new term is at depth 2, not depth 1;

- `at0001.1.1' - a term which specialises the term `at0001.1' from the immediate parent, which itself specialises the term `at0001' from the top parent.

This systematic definition of codes enables software to use the structure of the codes to more quickly and accurately make inferences about term definitions up and down specialisation hierarchies. Constraint codes on the other hand do not follow these rules, and exist in a flat code space instead.

Term and Constraint Definitions

Local term and constraint definitions are modelled as instances of the class *ARCHETYPE_TERM*, which is a code associated with a list of name/value pairs. For any term or constraint definition, this list shall at least include the name/value pairs for the names "text" and "description". It might also include such things as "provenance", which would be used to indicate that a term was sourced from an external terminology. The attribute term_attribute_names in *ARCHETYPE_ONTOLOGY* provides a list of attribute names used in term and constraint definitions in the archetype, including "text" and "description", as well as any others that are used in various places.

Generic Types Package

This package is included to confirm the semantics of the generic types used in this part standard. Although List<T>, Set<T>, Hash<T,K>, and Interval<T> are generic types supported by many programming environments, they are not directly supported in UML. In this package, new types such as List<String> are defined using Binding Dependencies between a new Basic Type such as List<String> and a Class (LIST in this example) that defines the minimum required semantics for all Lists.

Domain-specific Extension (Informative)

Domain-specific classes can be added to the archetype constraint model by inheriting from the class *C_DOMAIN_TYPE*. 7.12.1 (Scientific/Clinical Computing Types) shows the general approach, used to add constraint classes for commonly used concepts in scientific and clinical computing, such as `ordinal', `coded term' and `quantity'. The constraint types shown are *C_ORDINAL*, *C_CODED_TEXT* and *C_QUANTITY* which can optionally be used in archetypes to replace the default constraint semantics represented by the use of instances of *C_OBJECT* / *C_ATTRIBUTE.*

0.5 Overview of ADL

Archetype Definition Language (ADL) is a formal language for expressing archetypes. ADL uses two other syntaxes, cADL (constraint form of ADL) and dADL (data definition form of ADL) to describe constraints on data that are instances the information model specified in Clause 7 of this part-standard.

Archetypes expressed in ADL resemble programming language files, and have a defined syntax. 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 dADL syntax is used to express data, which appears in the **`language`**, **`description`**, **`ontology`**, and **`revision_history`** sections of an ADL archetype. The top-level structure of an ADL archetype is shown below.