The openEHR Archetype Model

openEHR Archetype Profile

Editors: T Beale

Revision: 1.0.0  Pages: 25  Date of issue: 08 Apr 2007

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

© 2005-2007 The openEHR Foundation.

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

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

email: info@openEHR.org  web: http://www.openEHR.org
Copyright Notice

© Copyright openEHR Foundation 2001 - 2007
All Rights Reserved

1. This document is protected by copyright and/or database right throughout the world and is owned by the openEHR Foundation.
2. You may read and print the document for private, non-commercial use.
3. You may use this document (in whole or in part) for the purposes of making presentations and education, so long as such purposes are non-commercial and are designed to comment on, further the goals of, or inform third parties about, openEHR.
4. You must not alter, modify, add to or delete anything from the document you use (except as is permitted in paragraphs 2 and 3 above).
5. You shall, in any use of this document, include an acknowledgement in the form: "© Copyright openEHR Foundation 2001-2007. All rights reserved. www.openEHR.org"
6. This document is being provided as a service to the academic community and on a non-commercial basis. Accordingly, to the fullest extent permitted under applicable law, the openEHR Foundation accepts no liability and offers no warranties in relation to the materials and documentation and their content.
7. If you wish to commercialise, license, sell, distribute, use or otherwise copy the materials and documents on this site other than as provided for in paragraphs 1 to 6 above, you must comply with the terms and conditions of the openEHR Free Commercial Use Licence, or enter into a separate written agreement with openEHR Foundation covering such activities. The terms and conditions of the openEHR Free Commercial Use Licence can be found at http://www.openehr.org/free_commercial_use.htm
## Amendment Record

<table>
<thead>
<tr>
<th>Issue</th>
<th>Details</th>
<th>Who</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td><strong>CR-000127</strong>: Restructure archetype specifications. Initial Writing.</td>
<td>T Beale</td>
<td>05 Feb 2005</td>
</tr>
<tr>
<td></td>
<td>Original modelling work.</td>
<td>T Beale, A Goodchild, Z Tun, D Kalra, D Lloyd, N Lea, T Austin</td>
<td>June 2004</td>
</tr>
</tbody>
</table>

## Acknowledgements

The work reported in this paper has been funded by University College London and Ocean Informatics Pty Ltd, Australia.
openEHR Archetype Profile

Table of Contents

1 Introduction................................................................. 7
  1.1 Purpose........................................................................... 7
  1.2 Related Documents..................................................... 7
  1.3 Status............................................................................. 7
  1.4 Peer review..................................................................... 7

2 Overview........................................................................... 9
  2.1 Background................................................................. 9
  2.2 Package Structure......................................................... 9

3 Data_types.basic Package............................................. 11
  3.1 Class Descriptions....................................................... 11
     3.1.1 C_DV_STATE Class............................................... 11
     3.1.2 STATE_MACHINE Class...................................... 12
     3.1.3 STATE Class........................................................ 12
     3.1.4 NON_TERMINAL_STATE Class............................. 12
     3.1.5 TERMINAL_STATE Class...................................... 13
     3.1.6 TRANSITION Class................................................ 13

4 Data_types.text Package.............................................. 14
  4.1 Overview......................................................................... 14
  4.2 Design............................................................................ 14
     4.2.1 Standard ADL Approach........................................ 14
     4.2.2 Inline dADL form.................................................. 14
     4.2.3 Custom Syntax Form.............................................. 15
     4.2.4 Archetype-local Codes........................................... 15
     4.2.5 Assumed value....................................................... 15
  4.3 Class Descriptions....................................................... 16
     4.3.1 C_CODE_PHRASE Class........................................ 16

5 Data_types.quantity Package...................................... 17
  5.1 Overview......................................................................... 17
  5.2 Design - Ordinals........................................................ 17
     5.2.1 Standard ADL........................................................ 17
     5.2.2 Inline dADL Section.............................................. 18
     5.2.3 Custom Syntax....................................................... 18
     5.2.4 Assumed Value....................................................... 18
  5.3 Design - Quantities...................................................... 19
     5.3.1 Standard ADL........................................................ 19
     5.3.2 Inline dADL Section.............................................. 19
  5.4 Class Definitions........................................................ 19
     5.4.1 C_DV_ORDINAL Class Definition.......................... 20
     5.4.2 C_DV_QUANTITY Class Definition.......................... 20
     5.4.3 C_QUANTITY_ITEM Class Definition...................... 21

6 Syntax Specification.................................................... 22
  6.1 Grammar........................................................................ 22
  6.2 Symbols......................................................................... 22
1 Introduction

1.1 Purpose
This document describes the openEHR Archetype Profile (AP), which defines custom constraint classes for use with the generic archetype object model (AOM). The intended audience includes:

- Standards bodies producing health informatics standards
- Software development organisations using openEHR
- Academic groups using openEHR
- The open source healthcare community
- Clinical and domain modelling specialists.

1.2 Related Documents
Prerequisite documents for reading this document include:

- The openEHR Architecture Overview
- The openEHR Archetype Definition Language (ADL)
- The openEHR Archetype Object Model (AOM)

1.3 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-1.0.1/publishing/architecture/am/openehr_archetype_profile.pdf.

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

1.4 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.
2 Overview

2.1 Background

An underpinning architectural feature of openEHR is the use of archetypes and templates, which are formal models of domain content, and are used to control data structure and content during creation, modification and querying. The elements of this architecture are twofold.

- The openEHR Reference Model (RM), defining the structure and semantics of information in terms of information models (IMs). The RM models correspond to the ISP RM/ODP information viewpoint, and define the data of openEHR EHR systems. The information model is designed to be invariant in the long term, to minimise the need for software and schema updates.

- The openEHR Archetype Model (AM), defining the structure and semantics of archetypes and templates. The AM consists of the archetype language definition language (ADL), the Archetype Object Model (AOM) and the openEHR Archetype profile (oAP).

The purpose of the ADL is to provide an abstract syntax for textually expressing archetypes and templates. The AOM defines the object model equivalent of ADL. It is reference model-neutral, meaning that it can be used to express archetypes for any reference model in a standard syntax. ADL and the AOM are brought together in an ADL parser, i.e. any tool which can read ADL archetype texts, and whose parse-tree (resulting in-memory object representation) is instances of the AOM.

The purpose of the openEHR Archetype Profile, the subject of this document, is to define custom archetype classes and in some cases, custom syntax equivalents (essentially shorthands) that can be used instead of the AOM generic classes for archetyping certain RM classes.

2.2 Package Structure

The openEHR Archetype Profile model is defined in the package am.openehr_profile, illustrated in FIGURE 1. It is shown in the context of the openEHR am and am.archetype packages. The internal structure of the package mimics the structure of the reference model it profiles, i.e. the openEHR reference model. This is done to make software development easier, even though the package structure may be sparsely populated. Packages need only be defined where there are custom types to be defined; the only ones currently defined are in the data_types package.
FIGURE 1 openehr.am.openehr_profile Package
3 Data_types.basic Package

The am.openehr_profile.basic package, illustrated in FIGURE 2, defines custom types for constraining the RM type DV_STATE.

A example of a state machine to model the state of a medication order is illustrated in FIGURE 3. This state machine is defined by an instance of the class STATE_MACHINE. (Note that for general modelling of states of medications and other interventions, the standard state machine defined in the EHR IM should normally be used).

3.1 Class Descriptions

3.1.1 C_DV_STATE Class

<table>
<thead>
<tr>
<th>CLASS</th>
<th>C_DV_STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>Constrainer type for DV_STATE instances. The attribute c_value defines a</td>
</tr>
<tr>
<td></td>
<td>state/event table which constrains the allowed values of the attribute value</td>
</tr>
<tr>
<td></td>
<td>in a DV_STATE instance, as well as the order of transitions between values.</td>
</tr>
</tbody>
</table>
### 3.1.2 STATE_MACHINE Class

<table>
<thead>
<tr>
<th>CLASS</th>
<th>STATE_MACHINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>Definition of a state machine in terms of states, transition events and outputs, and next states.</td>
</tr>
<tr>
<td>Attributes</td>
<td>Signature</td>
</tr>
<tr>
<td>1..1</td>
<td>states: Set &lt;STATE&gt;</td>
</tr>
<tr>
<td>Invariants</td>
<td>States_valid: states /= Void and then not states.is_empty</td>
</tr>
</tbody>
</table>

### 3.1.3 STATE Class

<table>
<thead>
<tr>
<th>CLASS</th>
<th>STATE (abstract)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>Abstract definition of one state in a state machine.</td>
</tr>
<tr>
<td>Attributes</td>
<td>Signature</td>
</tr>
<tr>
<td>1..1</td>
<td>name: String</td>
</tr>
<tr>
<td>Invariants</td>
<td>Name_valid: name /= Void and then not name.is_empty</td>
</tr>
</tbody>
</table>

### 3.1.4 NON_TERMINAL_STATE Class

<table>
<thead>
<tr>
<th>CLASS</th>
<th>NON_TERMINAL_STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>Definition of a non-terminal state in a state machine, i.e. one that has transitions.</td>
</tr>
<tr>
<td>Inherit</td>
<td>STATE</td>
</tr>
<tr>
<td>Attributes</td>
<td>Signature</td>
</tr>
<tr>
<td>1..1</td>
<td>transitions: Set &lt;TRANSITION&gt;</td>
</tr>
<tr>
<td>Invariants</td>
<td>Transitions_valid: transitions /= Void and then not transitions.is_empty</td>
</tr>
</tbody>
</table>
### 3.1.5 TERMINAL_STATE Class

<table>
<thead>
<tr>
<th>CLASS</th>
<th>TERMINAL_STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>Definition of a terminal state in a state machine, i.e. a state with no exit transitions.</td>
</tr>
<tr>
<td>Inherit</td>
<td>STATE</td>
</tr>
</tbody>
</table>

#### Attributes

<table>
<thead>
<tr>
<th>Signature</th>
<th>Meaning</th>
</tr>
</thead>
</table>

#### Invariants

- Event_valid: event /= Void and then not event.is_empty
- Action_valid: action /= Void implies not action.is_empty
- Guard_valid: guard /= Void implies not guard.is_empty
- Next_state_valid: next_state /= Void

### 3.1.6 TRANSITION Class

<table>
<thead>
<tr>
<th>CLASS</th>
<th>TRANSITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>Definition of a state machine transition.</td>
</tr>
</tbody>
</table>

#### Attributes

<table>
<thead>
<tr>
<th>Signature</th>
<th>Meaning</th>
</tr>
</thead>
</table>

#### Invariants

- Event_valid: event /= Void and then not event.is_empty
- Action_valid: action /= Void implies not action.is_empty
- Guard_valid: guard /= Void implies not guard.is_empty
- Next_state_valid: next_state /= Void
4 Data_types.text Package

4.1 Overview

The am.openehr_profile.data_types.text package contains custom classes for expressing constraints on instances of the types defined in the rm.data_types.text package. Only one type is currently defined, enabling the constraining of CODE_PHRASE instances. It is illustrated in FIGURE 4.

![Diagram](image)

FIGURE 4 am.openehr_profile.data_types.text Package

4.2 Design

4.2.1 Standard ADL Approach

The generic kind of constraint that can be expressed for the DV_CODED_TEXT type can, like all standard archetype constraints, only include constraints on the attributes defined in the reference model type. This is illustrated by the following fragment of ADL:

```plaintext
DV_CODED_TEXT matches {
    defining_code matches {
        CODE_PHRASE matches {
            terminology_id matches {"xxxx"}
            code_string matches {"cccc"}
        }
    }
}
```

The standard approach allows the attributes `terminology_id` and `code_string` to be constrained independently, and would for example, allow `terminology_id` to be constrained to ICD10|Snomed-ct|LOINC, while `code_string` could be constrained to some particular fixed values. However, this makes no sense; codes only make sense within a given terminology, not across them. It also makes no sense to allow codes from more than one terminology, as terminologies generally have quite different designs - LOINC and Snomed-CT are completely different in their conception and realisation.

A more appropriate kind of constraint for CODE_PHRASE instances is for `terminology_id` to be fixed to one particular terminology, and for `code_string` to be constrained to a set of allowed codes; an empty list indicates that any code is allowed. These semantics are formalised in the class definition, shown below.

4.2.2 Inline dADL form

In an archetype, an instance of C_CODE_PHRASE can be included as inline dADL, as in the following example:

```plaintext
defining_code matches {
    ...}
```
4.2.3 Custom Syntax Form

The same constraint as above can be expressed used a custom syntax extension to ADL. This form is most usually used for expressing value-set constraints within an archetype.

```xml
defining_code matches {
    [icd10::
        F43.00, -- acute stress reaction, mild
        F43.01, -- acute stress reaction, moderate
        F32.02] -- acute stress reaction, severe
}
```

4.2.4 Archetype-local Codes

In either of the constraint forms above, the special terminology name “local” is recognised. This is used to indicate that the listed terms come from the ontology section of the archetype itself, rather than an external terminology, as in the following example:

```xml
defining_code matches {
    [local::
        at1311, -- Colo-colonic anastomosis
        at1312, -- Ileo-colonic anastomosis
        at1313, -- Colo-anal anastomosis
        at1314, -- Ileo-anal anastomosis
        at1315] -- Colostomy
}
```

4.2.5 Assumed value

The custom code syntax provides an equivalent of the assumed value notion from standard ADL by repeating the assumed value separated by the semi-colon (;) character, as in the following example:

```xml
defining_code matches {
    [local::
        at1311, -- Colo-colonic anastomosis
        at1312, -- Ileo-colonic anastomosis
        at1313, -- Colo-anal anastomosis
        at1314, -- Ileo-anal anastomosis
        at1315; -- Colostomy
        at1312] -- (assumed value)
}
```
### 4.3 Class Descriptions

#### 4.3.1 C_CODE_PHRASE Class

<table>
<thead>
<tr>
<th>CLASS</th>
<th>C_CODE_PHRASE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>Express constraints on instances of CODE_PHRASE. The terminology_id attribute may be specified on its own to indicate any term from a specified terminology; the code_list attribute may be used to limit the codes to a specific list.</td>
</tr>
<tr>
<td>Inherit</td>
<td>C_DOMAIN_TYPE</td>
</tr>
</tbody>
</table>

### Attributes

<table>
<thead>
<tr>
<th>Signature</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>terminology_id: TERMINOLOGY_ID</td>
<td>Syntax string expressing constraint on allowed primary terms</td>
</tr>
<tr>
<td>code_list: List&lt;String&gt;</td>
<td>List of allowed codes; may be empty, meaning any code in the terminology may be used.</td>
</tr>
</tbody>
</table>

### Functions

<table>
<thead>
<tr>
<th>Signature</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>any_allowed: Boolean</td>
<td>True if any CODE_PHRASE instance allowed.</td>
</tr>
<tr>
<td>List_validity: code_list /= Void implies (not code_list.is_empty and terminology_id /= Void) Any_allowed_validity: any_allowed xor terminology_id /= Void</td>
<td></td>
</tr>
</tbody>
</table>

### Invariants

List_validity: code_list /= Void implies (not code_list.is_empty and terminology_id /= Void)
Any_allowed_validity: any_allowed xor terminology_id /= Void
5 Data_types.quantity Package

5.1 Overview

The am.openehr_profile.data_types.quantity package is illustrated in FIGURE 5. Two custom types are defined: C_DV_QUANTITY and C_DV_ORDINAL.

5.2 Design - Ordinals

5.2.1 Standard ADL

An ordinal value is defined as one which is ordered without being quantified, and is represented by a symbol and an integer number. The DV_ORDINAL class can be constrained in a generic way in ADL as follows:

```adl
item matches {
    DV_ORDINAL matches {
        value matches {0}
        symbol matches {
            DV_CODED_TEXT matches {
                defining_code matches {{local::at0014}} -- no heartbeat
            }
        }
    }
    DV_ORDINAL matches {
        value matches {1}
        symbol matches {
            DV_CODED_TEXT matches {
                defining_code matches {{local::at0015}} -- less than 100 bpm
            }
        }
    }
    DV_ORDINAL matches {
        value matches {2}
        symbol matches {
            DV_CODED_TEXT matches {

```
The above says that the allowed values of the attribute value is the set of **ORDINALs** represented by three alternative constraints, each indicating what the numeric value of the ordinal in the series, as well as its symbol, which is a **CODED_TEXT**.

### 5.2.2 Inline dADL Section

The above constraint can be represented as an inline instance of the openEHR type `C_ORDINAL`, as follows:

```adl
defining_code matches {
    C_DV_ORDINAL <
    list = <
        ["1"] = <
            value = <0>
            symbol = <
                defining_code = <![local::at0014]> -- no heartbeat
            >
        >
        ["2"] = <
            value = <1>
            symbol = <
                defining_code = <![local::at0014]> -- less than 100 bpm
            >
        >
        ["3"] = <
            value = <2>
            symbol = <
                defining_code = <![local::at0014]>-- greater than 100 bpm
            >
        >
    >
}
```

### 5.2.3 Custom Syntax

A more efficient way of representing the same constraint is using the following ADL syntax:

```adl
item matches {
    0|[local::at0014], -- no heartbeat
    1|[local::at0015], -- less than 100 bpm
    2|[local::at0016]; -- greater than 100 bpm
}
```

### 5.2.4 Assumed Value

Assumed value is represented in the same way as in the custom code syntax, i.e. by adding a semicolon demarcated value at the end of the list, as follows:

```adl
item matches {
    0|[local::at0014], -- no heartbeat
    1|[local::at0015], -- less than 100 bpm
    2|[local::at0016]; -- greater than 100 bpm
    0|[local::at0014] -- (assumed value)
}
```
5.3 Design - Quantities

5.3.1 Standard ADL
A typical need in clinical and demographic data containing an age attribute is to be able to constrain it to different ranges depending on whether it is expressed in months (as is normally the case with infants) or years (for adults). If the age value is expressed using the openEHR DV_QUANTITY, this constraint can be expressed as follows:

```dadt
age matches {
    DV_QUANTITY matches {
        property matches {"time"}
        units matches {"yr"}
        magnitude matches {|0.0..200.0|}
    }
    DV_QUANTITY matches {
        property matches {"time"}
        units matches {"mth"}
        magnitude matches {|3.0..12.0|}
    }
}
```

The above says that if units matches “years”, the constraint on `DV_QUANTITY.magnitude` is 0 - 200, while if units is “months” then the `magnitude` constraint is 3 - 12. This approach is not particularly efficient or clear, since it allows multiple instances of the constraint on the `property` attribute, when in fact `property` can only sensibly be the same for all branches of the constraint.

5.3.2 Inline dADL Section
The above constraint can be represented as an inline instance of the type `C_QUANTITY`, as below. Note that an assumed value has been included as well, just using normal dADL.

```dadt
age matches {
    C_DV_QUANTITY <
        property = <[openehr::128]> -- time
        list = <
            ["1"] = <
                units = "yr"
                magnitude = <|0.0..200.0|
                precision = <|2|>
            >
            ["2"] = <
                units = "mth"
                magnitude = <|1.0..36.0|
                precision = <|2|>
            >
            assumed_value = <
                magnitude = <1.0>
                units = "yr"
            >
        >
    }
}
```

5.4 Class Definitions
### 5.4.1 C_DV_ORDINAL Class Definition

<table>
<thead>
<tr>
<th>CLASS</th>
<th>C_DV_ORDINAL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Purpose</strong></td>
<td>Class specifying constraints on instances of DV_ORDINAL. Custom constrainer type for instances of DV_ORDINAL.</td>
</tr>
<tr>
<td><strong>Inherit</strong></td>
<td>C_DOMAIN_TYPE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Signature</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1..1</td>
<td>list: Set&lt;DV_ORDINAL&gt;</td>
<td>Set of allowed DV_ORDINAL values.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Functions</th>
<th>Signature</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(effected)</td>
<td>any_allowed: Boolean</td>
<td>True if any DV_ORDINAL instance allowed.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Invariants</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Ordinals_valid: items /= Void xor any_allowed</td>
<td></td>
</tr>
<tr>
<td>Items_valid: items /= Void implies not items.is_empty</td>
<td></td>
</tr>
</tbody>
</table>

### 5.4.2 C_DV_QUANTITY Class Definition

<table>
<thead>
<tr>
<th>CLASS</th>
<th>C_DV_QUANTITY</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Purpose</strong></td>
<td>Constrain instances of DV_QUANTITY.</td>
</tr>
<tr>
<td><strong>Inherit</strong></td>
<td>C_DOMAIN_TYPE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Signature</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0..1</td>
<td>list: List&lt;C_QUANTITY_ITEM&gt;</td>
<td>List of value/units pairs.</td>
</tr>
<tr>
<td>0..1</td>
<td>property: CODE_PHRASE</td>
<td>Optional constraint on units property</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Functions</th>
<th>Signature</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(effected)</td>
<td>any_allowed: Boolean</td>
<td>True if any DV_QUANTITY instance allowed.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Invariants</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>List_valid: list /= Void implies not list.is_empty</td>
<td></td>
</tr>
<tr>
<td>Property_valid: property /= Void implies terminology(Terminology_id_openehr).has_code_for_group_id (Group_id_property, property)</td>
<td></td>
</tr>
<tr>
<td>Overall_validity: (list /= Void or property /= Void) xor any_allowed</td>
<td></td>
</tr>
</tbody>
</table>
### 5.4.3 C_QUANTITY_ITEM Class Definition

<table>
<thead>
<tr>
<th>CLASS</th>
<th>C_QUANTITY_ITEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>Constrains instances of DV_QUANTITY.</td>
</tr>
<tr>
<td>Attributes</td>
<td></td>
</tr>
<tr>
<td>Signature</td>
<td>Meaning</td>
</tr>
<tr>
<td>0..1 magnitude: Interval&lt;Real&gt;</td>
<td>Constraint on the magnitude of the DV_QUANTITY.</td>
</tr>
<tr>
<td>0..1 precision: Interval&lt;Integer&gt;</td>
<td>Constraint on the precision of the DV_QUANTITY. A value of -1 means that precision is unconstrained.</td>
</tr>
<tr>
<td>1..1 units: STRING</td>
<td>Constraint on units of the DV_QUANTITY.</td>
</tr>
<tr>
<td>Functions</td>
<td></td>
</tr>
<tr>
<td>Signature</td>
<td>Meaning</td>
</tr>
<tr>
<td>precision_unconstrained: Boolean ensure precision = -1 implies Result</td>
<td>True if no constraint on precision; True if precision = -1.</td>
</tr>
<tr>
<td>Invariants</td>
<td>units_valid: units /= Void and not units.is_empty</td>
</tr>
</tbody>
</table>
6 Syntax Specification

The syntax described in this specification require some additions to the standard cADL grammar described in the openEHR ADL specification.

The additions to the grammar and lexical specificaitons for the standard cADL syntax are shown below. The actual grammar and lexical files used in the openEHR reference ADL parser (written in Eiffel) are available at http://my.openehr.org/wsvn/ref_impl_eiffel/TRUNK/components/adl_parser/src/syntax/cadl/parser/?rev=0&sc=0. The .l and .y files can be converted for use in other yacc/lex-based programming environments. The production rules of the .y file are available as an HTML document.

6.1 Grammar

The following shows additions to the standard cADL parser production rules (yacc specification) as of revision 158 of the Eiffel reference implementation repository (http://svn.openehr.org/ref_impl_eiffel).

```
c_object:  
c_complex_object  
| archetype_internal_ref  
| archetype_slot  
| constraint_ref  
| c_code_phrase -- added  
| c_ordinal -- added  
| c_primitive_object  
| V_C_DOMAIN_TYPE  
| ERR_C_DOMAIN_TYPE  
| error

c_ordinal:  
c_ordinal_spec  
| c_ordinal_spec ; integer_value  
| c_ordinal_spec ; error

c_ordinal_spec:  
 ordinal  
| c_ordinal_spec , ordinal

ordinal:  
 integer_value SYM_INTERVAL_DELIM V_QUALIFIED_TERM_CODE_REF

c_code_phrase:  
 V_TERM_CODE_CONSTRAINT  
| V_QUALIFIED_TERM_CODE_REF
```

6.2 Symbols

The following patterns are added to the lexical specification for the standard cADL grammar.

```
---------/* V_TERM_CODE_CONSTRAINT of form */ -----------------------
-- [terminology_id::code, -- comment
--    code, -- comment
--    code] -- comment
```
-- Form with assumed value
-- [terminology_id::code, -- comment
--   code; -- comment
--   code] -- an optional assumed value
--
\{[a-zA-Z0-9\.\-_]+::[ \t\n]* -- pick up [ line

<IN_TERM_CONSTRAINT> {
  [ \t]*[a-zA-Z0-9-\._-]+[ \t]*;[ \t\n]* -- pick up , line
  [ \t]*[a-zA-Z0-9-\._-]+[ \t]*[, [ \t\n]* -- pick up ; line
  \-\-[^\,\n]*\n  -- do nothing
  [ \t]*[a-zA-Z0-9-\._-]*[ \t\n]*\} -- pick up ] line
END OF DOCUMENT