



The *openEHR* Archetype Model
***openEHR* Archetype Profile**

Editors: T Beale¹

Revision: 1.0.0

Pages: 25

1. Ocean Informatics Australia

© 2005-2006 The *openEHR* Foundation

The *openEHR* 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
Patrons	To Be Announced

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

Issue	Details	Who	Date
R E L E A S E 1.0.1			
1.0.0	CR-000200: Correct Release 1.0 typographical errors. Global changes to this document. Fix invariants in C_QUANTITY classes. Correct C_QUANTITY. <i>property</i> to CODE_PHRASE. Correct invariants for C_CODED_TEXT; correct inheritance for C_DV_ORDERED. Corrected C_QUANTITY_ITEM class. Corrected errors in DV_STATE model by adding 2 new classes. CR-000219: Use constants instead of literals to refer to terminology in RM. CR-000224: Relax semantics of C_QUANTITY etc to allow no constraint. CR-000226: Rename C_CODED_TEXT to C_CODE_PHRASE.	T Beale, D Lloyd, R Chen A Patterson M Forss R Chen S Heard T Beale	22 Jan 2007
R E L E A S E 1.0			
R E L E A S E 0.95			
0.5	CR-000127. Restructure archetype specifications. Initial Writing.	T Beale	05 Feb 2005

Acknowledgements

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

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	20
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

Prerequisite documents for reading this document include:

- 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/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 archotyping 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.

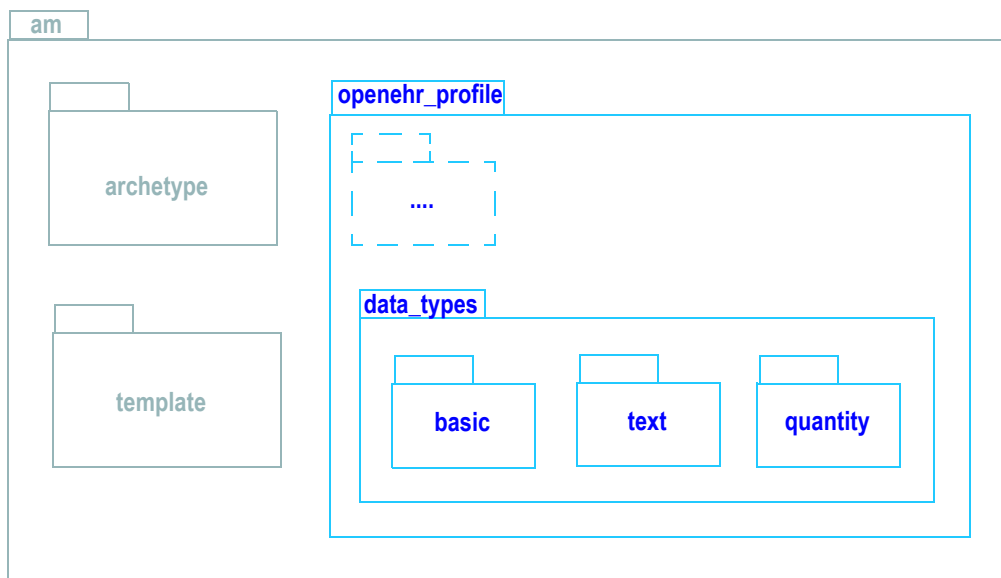


FIGURE 1 openehr.am.openehr_profile Package

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`.

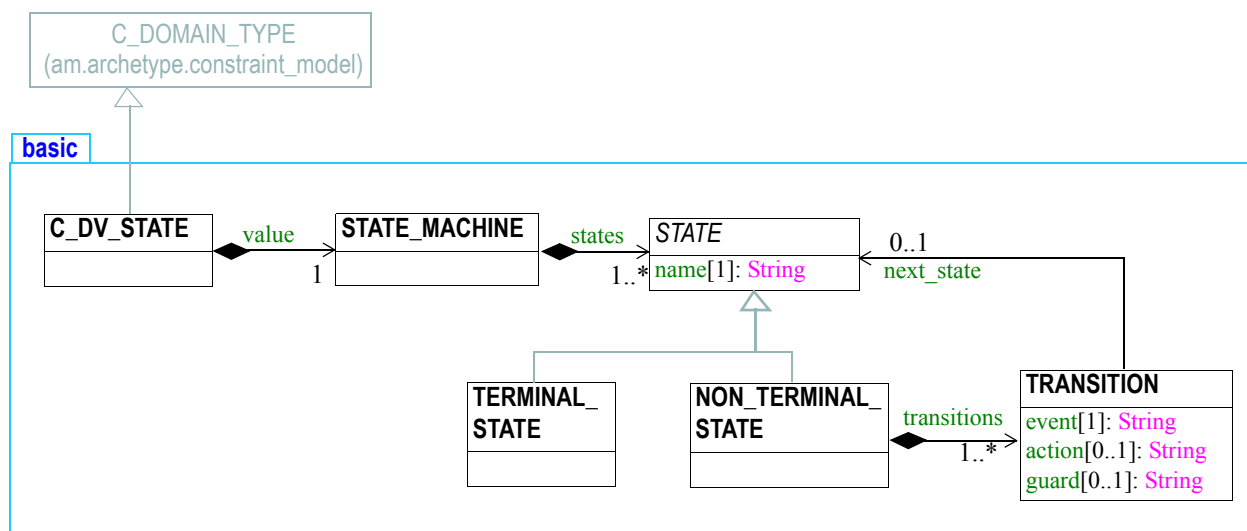


FIGURE 2 `am.openehr_profile.data_types.basic` Package

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).

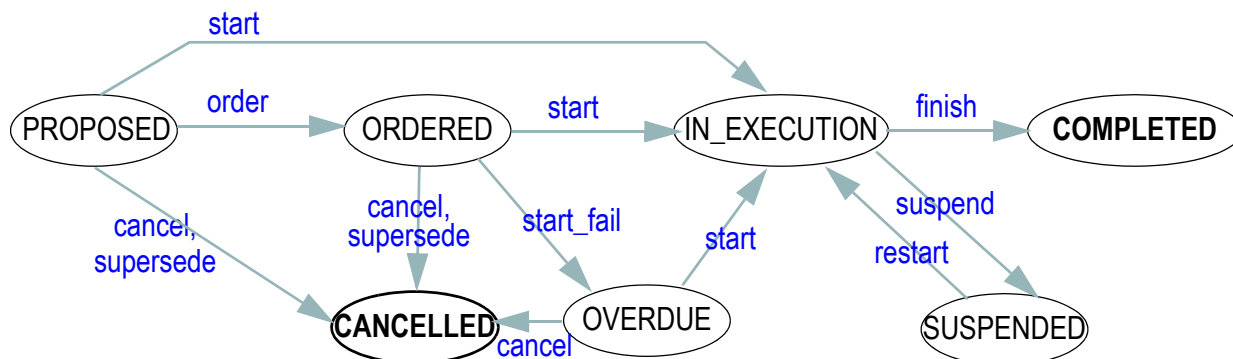


FIGURE 3 Example State Machine for Medication Orders

3.1 Class Descriptions

3.1.1 C_DV_STATE Class

CLASS	C_DV_STATE
Purpose	Constrainer type for <code>DV_STATE</code> instances. The attribute <code>c_value</code> defines a state/event table which constrains the allowed values of the attribute <code>value</code> in a <code>DV_STATE</code> instance, as well as the order of transitions between values.

CLASS	C_DV_STATE	
Inherit	C_DOMAIN_TYPE	
Attributes	Signature	Meaning
1..1	value: STATE_MACHINE	
Invariants	value_exists: c_value != Void	

3.1.2 STATE_MACHINE Class

CLASS	STATE_MACHINE	
Purpose	Definition of a state machine in terms of states, transition events and outputs, and next states.	
Attributes	Signature	Meaning
1..1	states: Set <STATE>	
Invariants	States_valid: states != Void and then not states.is_empty	

3.1.3 STATE Class

CLASS	STATE (<i>abstract</i>)	
Purpose	Abstract definition of one state in a state machine.	
Attributes	Signature	Meaning
1..1	name: String	name of this state
Invariants	Name_valid: name != Void and then not name.is_empty	

3.1.4 NON_TERMINAL_STATE Class

CLASS	NON_TERMINAL_STATE	
Purpose	Definition of a non-terminal state in a state machine, i.e. one that has transitions.	
Inherit	STATE	
Attributes	Signature	Meaning
1..1	transitions: Set <TRANSITION>	
Invariants	Transitions_valid: transitions != Void and then not transitions.is_empty	

3.1.5 TERMINAL_STATE Class

CLASS	TERMINAL_STATE	
Purpose	Definition of a terminal state in a state machine, i.e. a state with no exit transitions.	
Inherit	STATE	
Attributes	Signature	Meaning
Invariants		

3.1.6 TRANSITION Class

CLASS	TRANSITION	
Purpose	Definition of a state machine transition.	
Attributes	Signature	Meaning
1..1	event: String	Event which fires this transition
0..1	guard: String	Guard condition which must be true for this transition to fire
0..1	action: String	Side-effect action to execute during the firing of this transition
1..1	next_state: STATE	Target state of transition
Invariants	<i>Event_valid:</i> event != Void and then not event.is_empty <i>Action_valid:</i> action != Void implies not action.is_empty <i>Guard_valid:</i> guard != Void implies not guard.is_empty <i>Next_state_valid:</i> 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 `CODEPhrase` instances. It is illustrated in FIGURE 4.

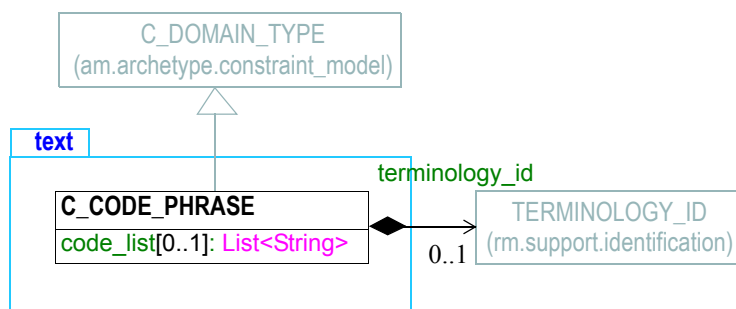


FIGURE 4 `am.openehr_profile.data_types.text` Package

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:

```

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 make 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 `CODEPhrase` 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:

```

defining_code matches {

```

```

C_CODE_PHRASE <
  terminology_id = <
    value = <"icd10">
  >
  code_list = <
    [1] = <"F43.00">      -- acute stress reaction, mild
    [2] = <"F43.01">      -- acute stress reaction, moderate
    [3] = <"F32.02">      -- acute stress reaction, severe
  >
>
}

```

4.2.3 Custom Syntax Form

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

```

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:

```

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:

```

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

CLASS	C_CODE_PHRASE	
Purpose	Express constraints on instances of <code>CODE_PHRASE</code> . The <i>terminology_id</i> attribute may be specified on its own to indicate any term from a specified terminology; the <i>code_list</i> attribute may be used to limit the codes to a specific list.	
Inherit	C_DOMAIN_TYPE	
Attributes	Signature	Meaning
0..1 (cond)	terminology_id: TERMINOLOGY_ID	Syntax string expressing constraint on allowed primary terms
0..1 (cond)	code_list: List<String>	List of allowed codes; may be empty, meaning any code in the terminology may be used.
Invariants	<p>List_validity: code_list != Void implies (not code_list.is_empty and terminology_id != Void)</p> <p>Any_allowed_validity: any_allowed = (terminology_id = Void and code_list = Void)</p>	

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`.

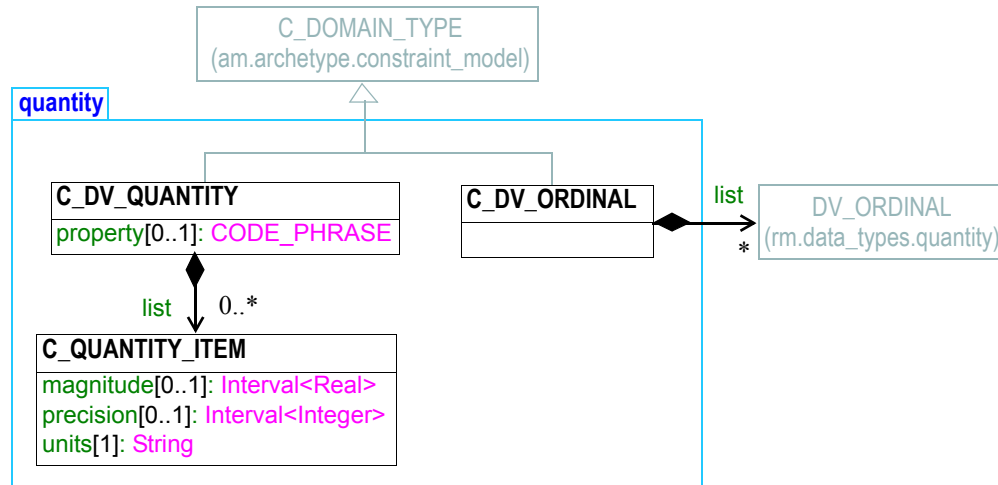


FIGURE 5 `am.openehr_profile.data_types.quantity` Package

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:

```

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 {

```

```

    bpm
    defining_code matches { [local::at0016] } -- greater than 100
  }
}

```

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:

```

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:

```

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 semi-colon demarcated value at the end of the list, as follows:

```

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:

```
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.

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

5.4 Class Definitions

5.4.1 C_DV_ORDINAL Class Definition

CLASS	C_DV_ORDINAL	
Purpose	Class specifying constraints on instances of DV_ORDINAL. Custom constrainer type for instances of DV_ORDINAL.	
Inherit	C_DOMAIN_TYPE	
Attributes	Signature	Meaning
1..1	list: Set<DV_ORDINAL>	Set of allowed DV_ORDINAL values.
Invariants	<i>Ordinals_valid:</i> items /= Void xor any_allowed <i>Items_valid:</i> items /= Void implies not items.is_empty	

5.4.2 C_DV_QUANTITY Class Definition

CLASS	C_DV_QUANTITY	
Purpose	Constrain instances of DV_QUANTITY.	
Inherit	C_DOMAIN_TYPE	
Attributes	Signature	Meaning
0..1	list: List<C_QUANTITY_ITEM>	List of value/units pairs.
0..1	property: CODE_PHRASE	Optional constraint on units property
Invariants	<i>List_valid:</i> list /= Void implies not list.is_empty <i>Property_valid:</i> property /= Void implies terminology(Terminology_id_openehr).has_code_for_group_id (Group_id_measurable_properties, property) <i>Overall_validity:</i> (list /= Void or property /= Void) xor any_allowed	

5.4.3 C_QUANTITY_ITEM Class Definition

CLASS	C_QUANTITY_ITEM	
Purpose	Constrain instances of DV_QUANTITY.	
Attributes	Signature	Meaning
0..1	magnitude: Interval<Real>	Constraint on the <i>magnitude</i> of the DV_QUANTITY.
0..1	precision: Interval<Integer>	Constraint on the <i>precision</i> of the DV_QUANTITY.

CLASS	C_QUANTITY_ITEM	
1..1	units : STRING	Constraint on <i>units</i> of the DV_QUANTITY.
Invariants	<i>units_valid</i> : units /= Void and not units.is_empty	

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 specificatoin 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