REFERENCE MODEL

The *open*EHR Data Structures Information Model

Editors:{T Beale, S Heard}[1], {D Kalra, D Lloyd}[2]

Revision: 1.4

Pages: 31

1. Ocean Informatics Australia

2. Centre for Health Informatics and Multi-professional Education, University College London

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.

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

Copyright Notice

Amendment Record

Issue	Details	Who	Completed
1.4	CR-000019. Add HISTORY & STRUCTURE supertype. CR-000028. Change name of STRUCTURE class to avoid clashes. CR-000089. Remove ITEM.*displayed*. CR-000091. Correct anomalies in use of CODE_PHRASE and DV_CODED_TEXT. CR-000067. Change EVENT class to enable math function interval measurements. Renamed EVENT.*duration* and SINGLE_EVENT.*duration* to *width*. **Formally validated using ISE Eiffel 5.4.**	T Beale H Frankel DSTC T Beale S Heard	09 Mar 2004
1.3.3	CR-000041. Visually differentiate primitive types in openEHR documents.	D Lloyd	04 Sep 2003
1.3.2	CR-000013 Rename key classes - rename COMPOUND to CLUSTER to conform with CEN 13606.	D Kalra, T Beale	20 Jun 2003
1.3.1	Improved heading layout, package naming. Made HISTORY.*is_periodic* a function.	T Beale, Z Tun	18 Mar 2003
1.3	**Formally validated using ISE Eiffel 5.2.** No changes.	T Beale	20 Feb 2003
1.2.1	Minor corrections to terminology_id invariants.	Z Tun	08 Jan 2003
1.2	Defined packages properly and moved HISTORY classes from EHR RM. No change to semantics.	T Beale	18 Dec 2002
1.1.1	Minor corrections: SINGLE_S new function.	T Beale	10 Nov 2002
1.1	Minor adjustments due to change in DV_CODED_TEXT.	T Beale	1 Nov 2002
1.0	Taken from Common RM.	T Beale	11 Oct 2002

Acknowledgements

The work reported in this paper has been funded in by a number of organisations, including The University College, London; The Cooperative Research Centres Program through the Department of the Prime Minister and Cabinet of the Commonwealth Government of Australia; Ocean Informatics, Australia.

Table of Contents

1 Introduction

1.1 Purpose

This document describes the common data structures used in *open*EHR reference models, including lists, tables, trees, and history, along with one possible data representation (hierarchical) which is compatible with the CEN 13606 EHCR standard and the HL7v3 message specification.

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 intersted in health information;
- Health data managers.

1.2 Related Documents

Prerequisite documents for reading this document include:

- The *open*EHR Modelling Guide
- The *open*EHR Data Types Reference Model

1.3 Status

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

Currently the UML diagrams are hand-produced. None of the existing tools (e.g. Rose, Objecteering), includes sufficient support of UML or has good enough visual quality to use here. However, UML tools are constantly under investigation, and this situation may change in the future.

The latest version of this document can be found in PDF and HTML formats at `http://www.openEHR.org/Doc_html/Model/Reference/data_structures_rm.htm`. New versions are announced on `openehr-announce@openehr.org`.

1.4 Peer review

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 discussed on one of the appropriate mailing lists, `openehr-technical@openehr.org` or `openehr-clinical@openehr.org`.

2 Background

2.1 Requirements

The requirements for structured data in the EHR and other systems are essentially that low-level data can be expressed in standard structures. The structures which are commonly required are as follows:

- single values, e.g. weight, height, blood sugar;
- lists of named/numbered elements, e.g. blood test results;
- tables of values with named columns and/or named rows, e.g. visual acuity results;
- trees of values, e.g. biochemistry, microbiology results;
- histories of values, each of which takes any of the above forms, e.g. a time series of blood pressures, glucose levels, or imaging data.

2.2 Design Principles

The design principle which particularly applies to the data structure models described here is the need to provide explicit specifications for logical structures using the same generic representation, such as hierarchy. The logical structures include tables, lists, trees, and the concept of history.

Regardless of whether such structures are treated as pure presentation or as having semantic significance, there are at various reasons for explicitly including the semantics of logical structures which are represented in a generic way such as hierarchy, including:

- it is essential for interoperability that a structure such as a logical table, list or linear history be encoded into the generic representation in the same way by all senders and receivers of information, otherwise there is no guarantee that any communicating party's software processes the structures in the intended fashion;
- software implementors can develop software which explicitly captures the logical structures as functional interfaces which are used as the only way of building such structures. Such interfaces (assuming they are bug-free) guarantee that all application software always creates correct structures - there is no need to rely on caller software each time making low level calls to create a table or list out of hierarchy elements;
- the use of a functional interface for such types means that application software at the receiver's end can always process incoming information in its intended form, enabling correct presentation of data on the screen.

One of the motivations for defining logical data structures explicitly is to remove the ambiguity in recording structure and time in previous EHR specifications and standards, such as CEN 13606, GEHR, GEHR Australia, and HL7 CDA specifications. The alternative in the past was to simply use generic hierarchical structures; there was no agreement in the standard about how a table might be represented, similarly, time had no standard representation. Where single values were recorded, an attribute meaning 'time of recording' was set appropriately; if a time series was required, there was no clear guideline as to how to model it. One way would have been to build a double list which is logically a two column table, whose first column was time-point data, but many other approaches are possible. The standardised approach removes all such ambiguity, and improves the quality of data and software.

3 Overview

The Data_Structure package contains two packages: the Structure package and the History package. The first descibes generic, path-addressable data structures, while the latter describes a generic notion of linear history, for recording events in past time. The package is illustrated in FIGURE 1.

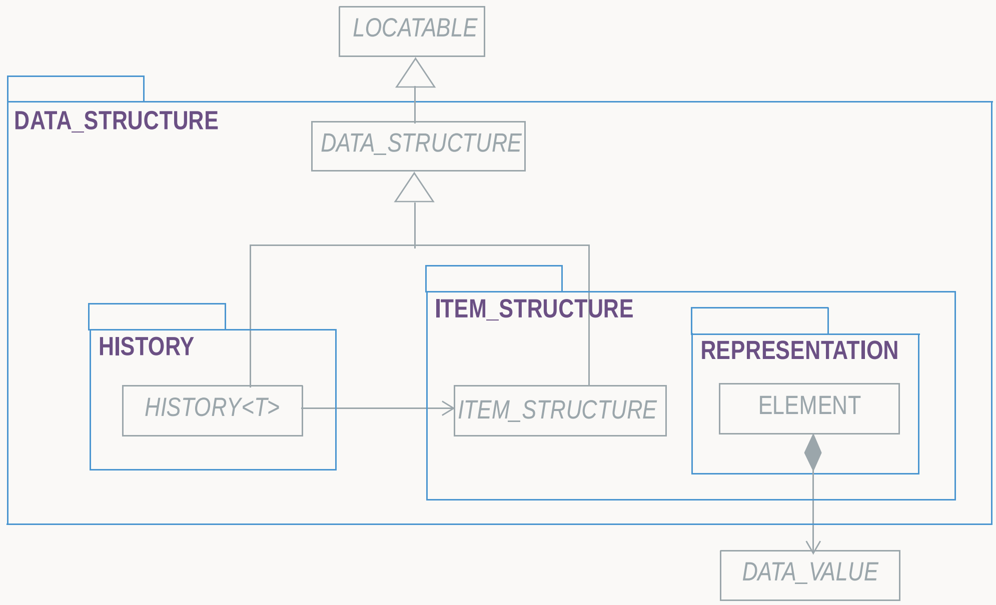


FIGURE 1 RM.DATA_STRUCTURE Package

4 RM.DATA_STRUCTURE. ITEM_STRUCTURE Package

4.1 Overview

The Item_Structure classes presented here are a formalisation of the need for generic clinical data structures, and are used by all reference models. The approach to spatially complex structures taken is to explicitly model the logical data structure types which typically occur in health record data. The CEN 13606 and GEHR approach of using one hierarchical representation for all data is used. However, the *open*EHR model treats the hierarchy part of the model as *representation* only, and puts it in its own package. A new package is added, the Item_Structure package, containing a number of classes representing logical data structures, including single value, list, table, and tree. Each of these classes defines clear semantics for its respective structural type, including rules for encoding the structure into the hierarchical representation. As a result, any system implementing these types is guaranteed to create data which represents these structures identically. The design principle RM-spatial [2] and preceding discussion describe the reasons behind this approach.

Data values are connected to the spatial model via the value *attribute* of the `ELEMENT` class of the Representation cluster. This class also carries an important attribute *interpretation*, whose value indicates how to read the value. A small domain termlist containing values such as "unknown", "not disclosed", "undetermined", etc, as described in the Flavours of Null section of the *open*EHR Data Types Information Model.

The *open*EHR class model for spatial structures is illustrated in FIGURE 2. These classes are not equivalents of similarly named classes in most data structure libraries - they also include per-node *name*, *meaning* and leaf node value and null flavour, and path capabilities.

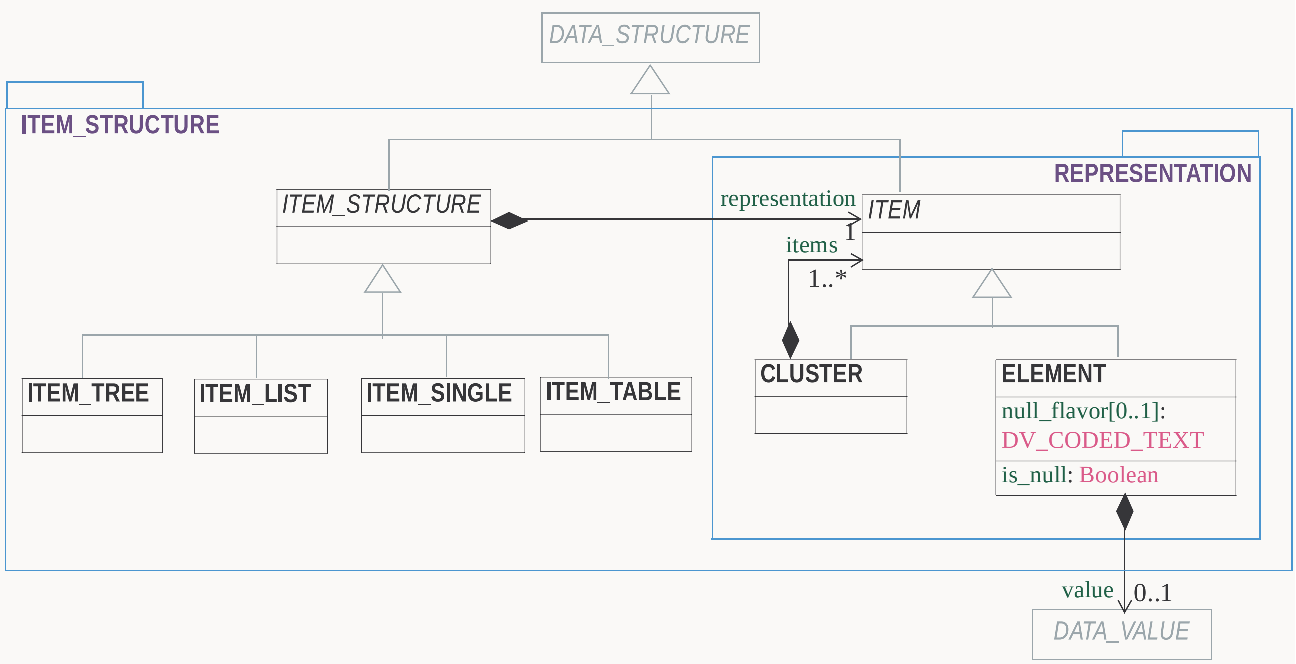


FIGURE 2 RM.DATA_STRUCTURE.STRUCTURE Package

Diagrams of typical instances of the structures are included throughout this document. Each instance of shown in both physical and logical form. The physical form shows the instances which will occur in data if the structure is implemented using the hierarchical representation package. The logical form shows the same instance in a logical form only - i.e. hiding the physical implementation. Only the latter form is used in other *open*EHR documents. In all instance diagrams, the following shorthand is used for well-known attribute names:

- “m = xxxx” - means “meaning = xxxx”, i.e. the value of the *meaning* attribute inherited from the `LOCATABLE` class.
- “n = xxxx” - means “name = xxxx”, i.e. the value of the *name* attribute inherited from the `LOCATABLE` class.
- “v = xxxx” - means “value = xxxx”, i.e. the value of the *value* attribute from the `ELEMENT` class.

4.2 Class Descriptions

4.2.1 DATA_STRUCTURE Class

CLASS	***DATA_STRUCTURE (abstract)***	
Purpose	Abstract parent class of all data structure types.	
Inherit	`LOCATABLE`	
Abstract	**Signature**	**Meaning**
Invariants		

4.2.2 ITEM_STRUCTURE Class

CLASS	***ITEM_STRUCTURE (abstract)***	
Purpose	Abstract parent class of all spatial data types.	
GEHR	`G1_HIERARCHICAL_PROPOSITION`	
HL7	CDA `Structure` abstract type.	
Inherit	`DATA_STRUCTURE`	
Abstract	**Signature**	**Meaning**
	representation: `ITEM`	
Invariants	***Representation_exists***: representation /= Void	

4.2.3 ITEM_SINGLE Class

CLASS	ITEM_SINGLE	
Purpose	Logical single value data structure.	
Use	Used to represent any data which is logically a single value, such as a person's height or weight.	
GEHR	G1_SIMPLE_PROPOSITION	
HL7	CDA `Item` type.	
Inherit	ITEM_STRUCTURE	
Attributes	**Signature**	**Meaning**
	representation: ELEMENT	
Functions	**Signature**	**Meaning**
	item: ELEMENT	Retrieve the item.
Invariants		

4.2.3.1 ITEM_SINGLE Paths

In the following path structure, the name of the ITEM_SINGLE object acts as the root-name.

- the item: "|" <ITEM_SINGLE.name>, e.g. "|weight"

4.2.3.2 ITEM_SINGLE Structural Encoding Rules

The ITEM_SINGLE encoding rules are as follows:

- The item is represented by a single ELEMENT.

4.2.3.3 ITEM_SINGLE Instance Structure

FIGURE 3 illustrates a ITEM_SINGLE instance, in both physical and logical forms.

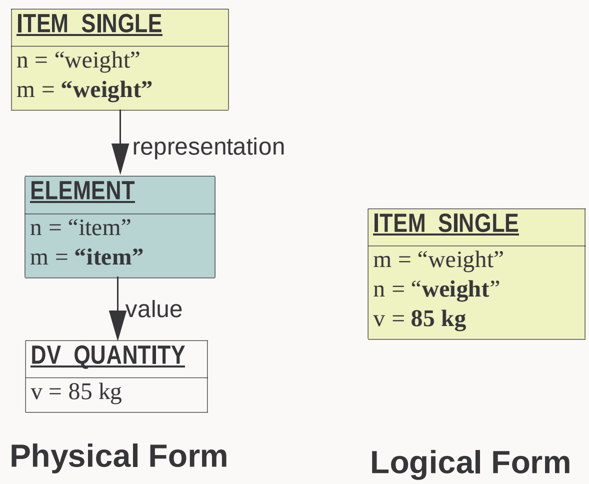


FIGURE 3 Instance Structure of ITEM_SINGLE

4.2.4 ITEM_LIST Class

CLASS	**ITEM_LIST**	
Purpose	Logical list data structure, where each item has a value and can be referred to by a name and a positional index in the list.	
Use	Used to represent any data which is logically a list of values, such as blood pressure, most protocols, many blood tests etc.	
MisUse	Not to be used for time-based lists, which should be represented with the proper temporal class, i.e. `HISTORY`.	
GEHR	`G1_LIST_PROPOSITION`	
HL7	CDA 1.0 `List Entry` type.	
Inherit	`ITEM_STRUCTURE`	
Attributes	**Signature**	**Meaning**
	representation: `CLUSTER`	
Functions	**Signature**	**Meaning**
	item_count: `Integer`	Count of all items
	items: `List<ELEMENT>`	Retrieve all items
	names: `List<DV_TEXT>`	Retrieve the names of all items

CLASS	ITEM_LIST	
	named_item(a_name:String): ELEMENT	Retrieve the item with name 'a_name'
	ith_item(i:Integer): ELEMENT	Retrieve the i-th item with name
Invariants	***Valid_structure***: representation.items.forall({ITEM}.type = "ELEMENT")	

4.2.4.1 ITEM_LIST Paths

In the following path structure for Lists, the *name* attribute of the ITEM_LIST object acts as the root-name.

- whole list: "|" <ITEM_LIST.name>, e.g. "|BP protocol"
- nth list item:"|" <ITEM_LIST.name> "|item=" <n>, e.g. "|BP protocol|item=2"
- named list item: "|" <ITEM_LIST.name> "|" <item_name>, e.g. "|BP protocol|cuff"

4.2.4.2 ITEM_LIST Structural Encoding Rules

The ITEM_LIST encoding rules are as follows:

- The list as a whole has a single CLUSTER object as the root.
- Each item is represented by an ELEMENT object, whose names is the name of the item.

4.2.4.3 ITEM_LIST Instance Structure

FIGURE 4 illustrates a typical ITEM_LIST structure, in this case for a BP protocol.

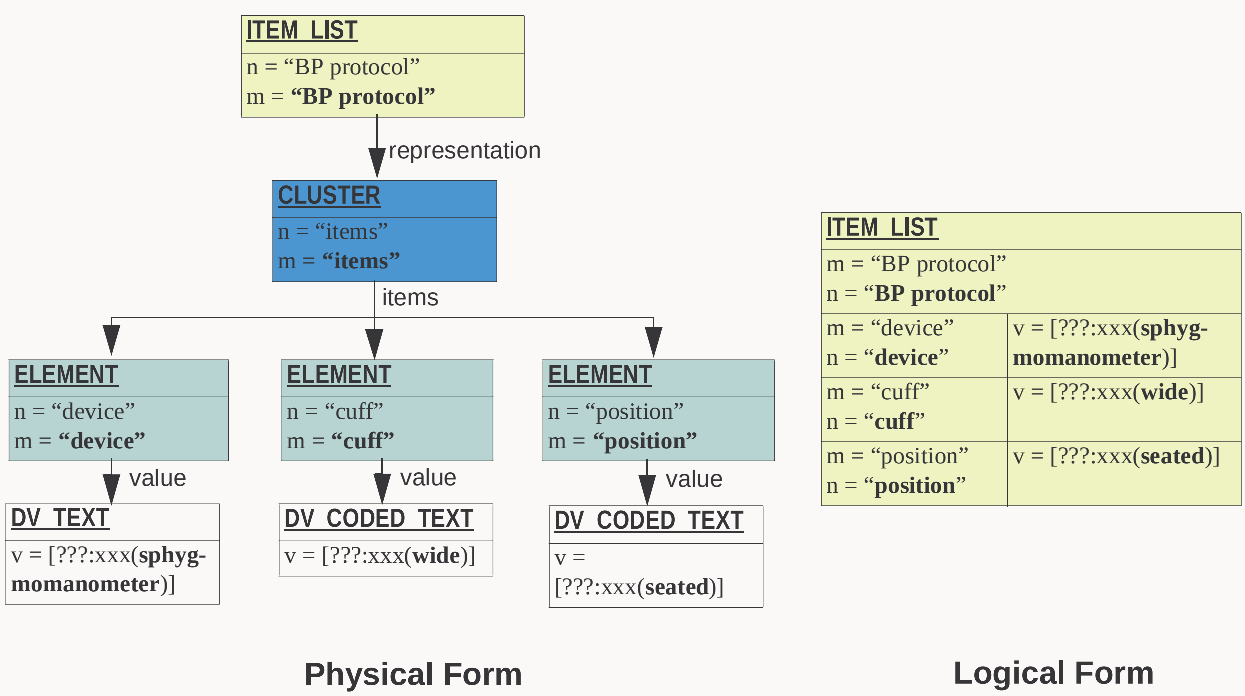


FIGURE 4 ITEM_LIST Instance Structure

4.2.5 ITEM_TABLE Class

CLASS	ITEM_TABLE	
Purpose	Logical table data structure, in which columns are named and ordered. Some columns may be designated ‘key’ columns, containing key data for each row, in the manner of relational tables. This allows row-naming, where each row represents a body site, a blood antigen etc. All values in a column have the same data type.	
Use	Used to represent any data which is logically a table of values, such as blood pressure, most protocols, many blood tests etc.	
MisUse	Not used for time-based data, which should be represented with the temporal class `HISTORY`.	
CEN	n/a	
GEHR	`G1_TABLE_PROPOSITION, G1_MATRIX_PROPOSITION`	
HL7	RIM structured types Table_structure, Table_cell, Table etc ; CDA 1.0 Table Entry type.	
Inherit	`ITEM_STRUCTURE`	
Attributes	**Signature**	**Meaning**
	representation: `CLUSTER`	
Functions	**Signature**	**Meaning**
	row_count:`Integer`	Return the number of rows
	column_count:`Integer`	Return the number of columns
	row_names: `List<DV_TEXT>`	Return the row names
	column_names: `List<DV_TEXT>`	Return the column names
	ith_row(i:`Integer`**)**: `List<ELEMENT>` ***require*** i >0 ***and*** i < row_count	Return the i-th row
	has_row_with_name(a_key:`String`**)**: `Boolean` ***require*** a_key /= Void ***and then not*** a_key.empty	True if there is a row whose first column has the name ‘a_key’
	has_column_with_name(a_key:`String`): `Boolean` ***require*** a_key /= Void ***and then not*** a_key.empty	True if there is a column with name ‘a_key’

CLASS	ITEM_TABLE	
	named_row(a_key:String): List<ELEMENT> ***require*** has_row_with_name(a_key)	Return the row whose first column has the name ‘a_key’
	has_row_with_key(keys:Set<String>): Boolean	True if there is a row whose first n columns have the names in ‘keys’
	row_with_key(key_vals:Set<String>) : List<ELEMENT> ***require*** has_row_with_key(key_vals)	Return the row whose first n columns have names equal to the values in ‘keys’
	element_at_cell_ij(i, j:Integer): ELEMENT ***require*** i >= 1 ***and*** i <= column_count j >= 1 ***and*** j <= row_count	Return the element at the column i, row j.
	element_at_named_cell (row_key, col_key:String): ELEMENT ***require*** has_row_with_name(row_key) has_column_with_name(column_key)	Return the element at the row whose first column has the name ‘row_key’ and column has the name ‘col_key’
Invariants	***Valid_structure***: representation.items.forall({ITEM}.type = “CLUSTER” ***and then*** {ITEM}.items.forall({ITEM}.type = “ELEMENT”))	

4.2.5.1 ITEM_TABLE Paths

The following path patterns are legal for tables.

- whole table: `“|” <ITEM_TABLE.name>`, e.g. `“|root”`
- column:`“|” <ITEM_TABLE.name> “|” <column-name>`, e.g. `“|vision|left eye”`
- row: `“|” <ITEM_TABLE.name> “|” “row=” <row-name>`, e.g. `“|vision|row=colour”`
- cell: `“|” <ITEM_TABLE.name> “|” “col=” <column-name> “|” “row=” <row-name>`, e.g. `“|vision|col=left eye|row=acuity w/ glasses”`

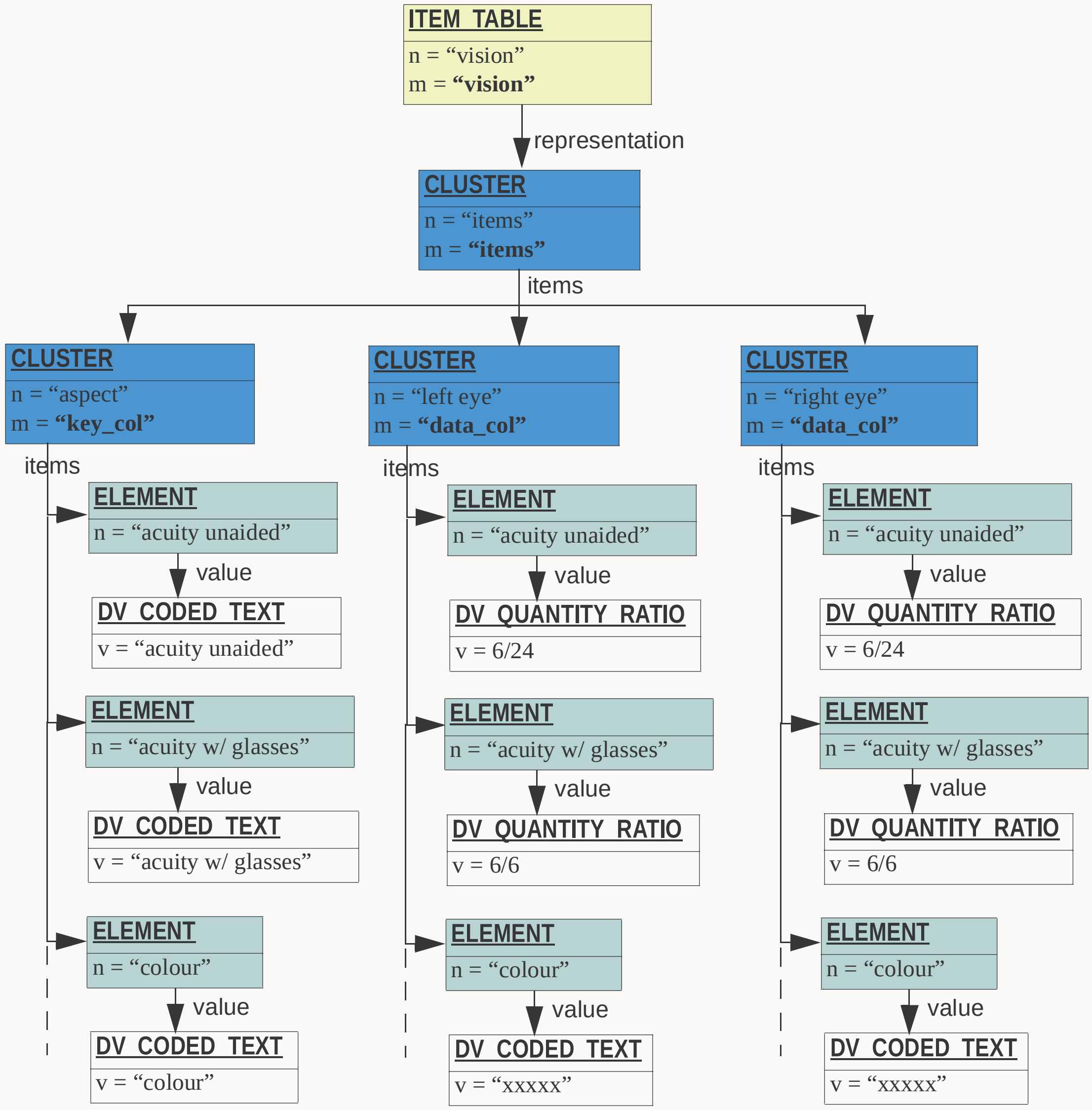
4.2.5.2 ITEM_TABLE Structural Encoding Rules

The ITEM_TABLE encoding rules are as follows:

- A CLUSTER is required as the parent of all columns.
- Each column is represented by a CLUSTER, whose name value is the name of the column.
- Each row item in a given column is represented by an ELEMENT under the relevant column CLUSTER.
- The name of each ELEMENT object is the name of its row.

4.2.5.3 ITEM_TABLE Instance Structure

FIGURE 5 illustrates a table of visual acuity test results.



Physical Form

ITEM_LIST m = “vision” n = **“vision”**		
m = “key_col” n = **“aspect”**	m = “data_col” n = [???:xxx(**left eye**)]	m = “data_col” n = [???:xxx(**right eye**)]
m = “acuity unaided” n = **“acuity unaided”**	6/24	6/24
m = “acuity with glasses” n = **“acuity with glasses”**	6/6	6/6
m = “colour” n = **“colour”**	normal	normal

Logical Form

FIGURE 5 ITEM_TABLE Instance Structure

4.2.6 ITEM_TREE Class

CLASS	ITEM_TREE	
Purpose	Logical tree data structure.	
Use	Used to represent data which are logically a tree such as audiology results, micro-biology results, biochemistry results.	
MisUse		
CEN	The CEN cluster is effectively the only data structure available in CEN, and is equivalent to the `ITEM_TREE` type.	
GEHR	`G1_TREE_PROPOSITION`	
HL7	This can be constructed with CDA 1.0 Lists. Act and Act_relationship are the closest correspondents in the HL7 RIM.	
Inherit	`ITEM_STRUCTURE`	
Attributes	**Signature**	**Meaning**
	representation: `CLUSTER`	
Functions	**Signature**	**Meaning**
	has_element_path(a_path:`String`): `Boolean`	True if path 'a_path' is a valid leaf path
	element_at_path(a_path:`String`**)**: `ELEMENT` ***require*** has_element_path(a_path)	Return the leaf element at the path 'a_path'

4.2.6.1 ITEM_TREE Paths

Tree paths are of the following form.

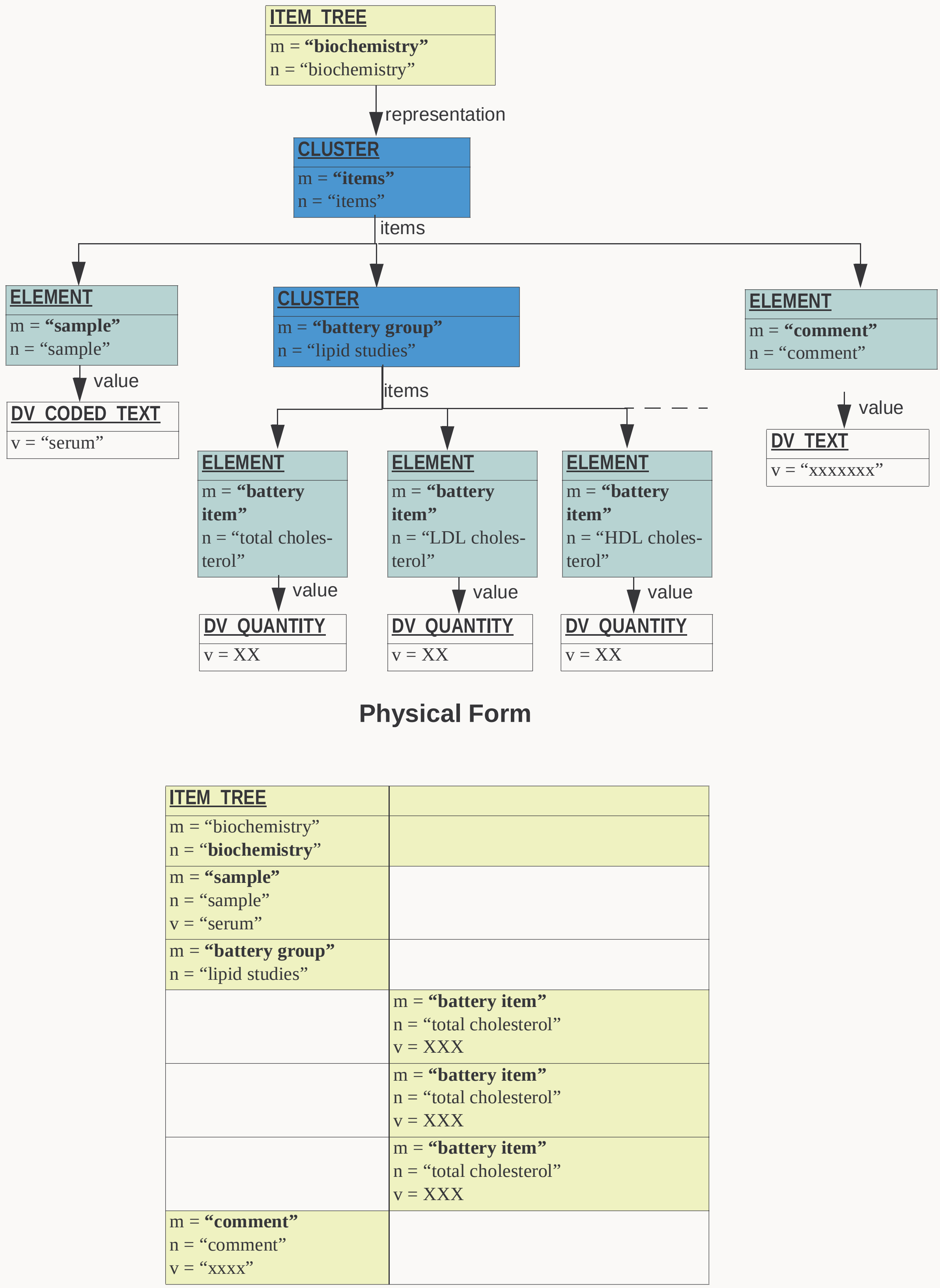
- whole tree: `"|" <ITEM_TREE.name>`, e.g. `"|biochemistry"`
- node: `"|" <ITEM_TREE.name> "|" <node-name>...<node-name>`,
  e.g. `"|biochemistry|lipid studies"`
- leaf value: `"|" <ITEM_TREE.name> "|" <node-name>...<node-name> "|" <leaf-name>"`, e.g. `"|biochemistry|lipid studies|LDL cholesterol"`

4.2.6.2 ITEM_TREE Structural Encoding Rules

Logically hierarchical data is encoded directly into tree structures without structral transformation.

4.2.6.3 ITEM_TREE Instance Structure

FIGURE 6 illustrates te logical and physical form of an example `ITEM_TREE` instance, representing a biochemistry result.



Logical Form

FIGURE 6 ITEM_TREE Instance Structure

5 RM.DATA_STRUCTURE.STRUCTURE. REPRESENTATION Package

5.1 Overview

This package constains classes for a simple hierarhical representation of any data structure.

5.2 Class Descriptions

5.2.1 ITEM Class

CLASS	*ITEM (abstract)*	
Purpose	The abstract parent of CLUSTER and ELEMENT representation classes.	
CEN	ITEM class	
OMG HDTF	COAS::Observation	
Synapses	Item class	
GEHR	G1_HIERARCHICAL_ITEM	
HL7	n/a	
Inherit	LOCATABLE	
Attributes	Signature	Meaning

5.2.2 CLUSTER Class

CLASS	CLUSTER
Purpose	The grouping variant of ITEM, which may contain further instances of ITEM, in an ordered list.
CEN	ClusterOCC class
OMG HDTF	COAS::CompositeObservation
Synapses	Compound class
GEHR	G1_HIERARCHICAL_GROUP
HL7	Act_context

CLASS	CLUSTER	
Inherit	ITEM	
Attributes	Signature	Meaning
	items: List [ITEM]	Ordered list of items - CLUSTER or ELEMENT objects - under this CLUSTER.
Invariants	***Items_non_empty***: items /= Void ***and then not*** items.empty	

5.2.3 ELEMENT Class

CLASS	ELEMENT	
Purpose	The leaf variant of ITEM, to which a DATA_VALUE instance is attached.	
CEN	DataItem class	
OMG HDTF	COAS::AtomicObservation	
Synapses	Element class	
GEHR	G1_HIERARCHICAL_VALUE	
HL7	Act	
Inherit	ITEM	
Attributes	Signature	Meaning
	value: DATA_VALUE	data value of this leaf
	null_flavor: DV_CODED_TEXT	flavour of null value, e.g. indeterminate, not asked etc
Functions	Signature	Meaning
	is_null: Boolean	True if value logically not known, e.g. if indeterminate, not asked etc.
Invariants	***Null_flavor_indicated***: is_null ***xor*** null_flavour /= Void ***Null_flavour_valid***: is_null ***implies*** terminology("openehr").codes_for_group_name("null flavour", "en").has(null_flavor.defining_code)	

6 RM.DATA_STRUCTURE.HISTORY Package

6.1 Overview

The HISTORY package defines classes which formalise the concept of past, linear time, at which data of any complexity can be recorded. It allows both discrete (instantaneous) and continuous events (i.e. states) within periodic or aperiodic series.

The approach taken is that single samples, which are the most common in most circumstances, are represented in the same way as multiple samples, i.e. time series, allowing all software to access all data in a uniform way, regardless of whether it is a single measurement of weight, a long series of three- or four-dimensional images, or even a series of encapsulated multimedia items. This is clearly a better situation than past models, since there is no logical difference between one sample of any datum and many.

The model defines a structure which enables 'histories' consisting of a number of events situated along a timeline to be expressed. Histories may consist of discrete periodic events or states, i.e. values which are maintained for longer than an instant. Each state interval or event instant is associated with a spatial data structure. The effect is that repeated instances of spatially complex data can recur in

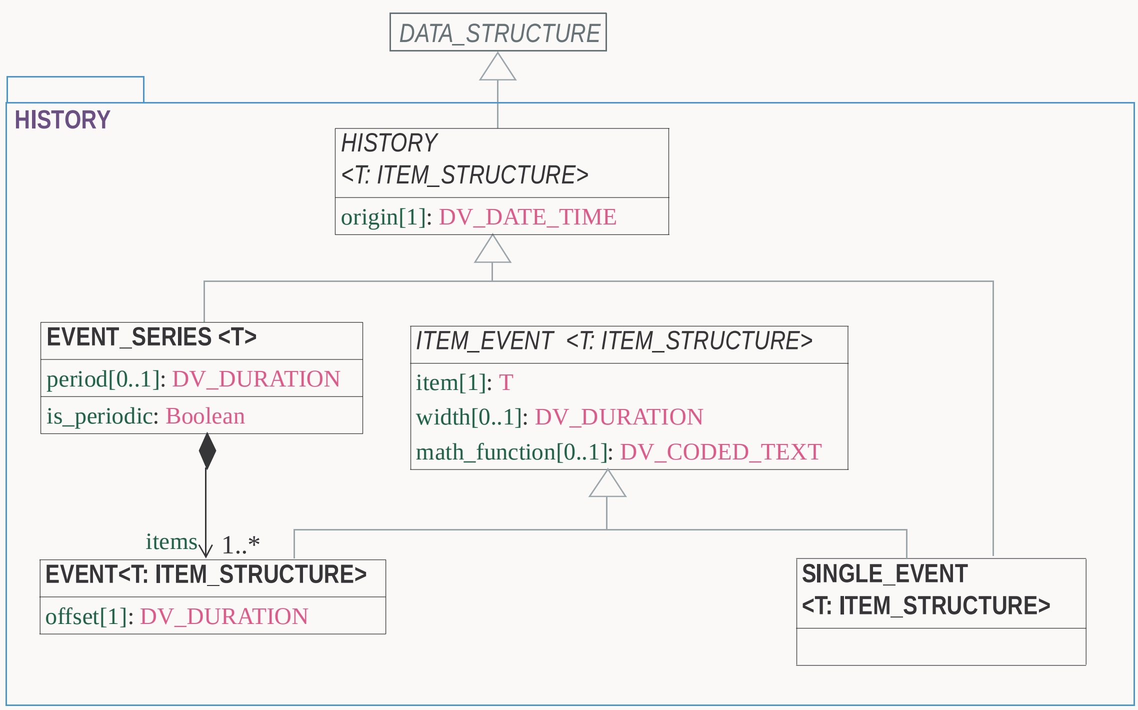


FIGURE 7 RM.DATA_STRUCTURE.HISTORY Package

time, corresponding to the way data are actually measured. A special type, `SINGLE_EVENT<T>` is provided to cater for the many cases where a history is only one event long. A periodic discrete series would be used to represent most vital signs monitor output, but also manual measurements repeated in time, as long as the measurer and the protocol remain the same. The History package is shown in FIGURE 7.

6.1.1 Scope

It should be noted that the intention of this model is to represent single sample and time-based data for which measurement protocol is the same. It is not intended for measurements in "coarse" time taken by different people, different instruments, or with any other difference in data-gathering technique. In these cases, separate, usually single-sample histories are used, usually occurring in distinct container objects (e.g. Compositions, in the EHR). Accordingly, in the general practice setting, the use of `HISTORY<T>` will correspond to measurement series which occur *during* the clinical session (i.e. during a patient contact). In a hospital setting, nurses' observations might occur in 4-hourly intervals, and there is no well-defined clinical session - simply a series of `ENTRYs` during the time of the episode. Two approaches are possible here.

- If each observation is to be committed to the EHR as soon as it is made, the result should be distinct Compositions in time, each with its Event_context corresponding to the period of the nurse's presence. Each Composition will contain the Single_event subtype of History (unless the nurse actually performed a series of measurements on the spot).
- If observations are not committed to the EHR immediately, but are stored elsewhere and only committed (say) at the end of each day, then the result will be a single Composition whose Event_context corrsponds to the data gathering period, and which contains an instance of the Event_series subtype of History, itself containing the multiple measurements made over the day.

Whether time-based data remain outside the record until a series of desired length is gathered, or entered as it occurs is completely up to the design of applications and systems; the approach taken should be based on the desired availability of the data in the system in question. If for example, it must be visible in the EHR as soon as the appropriate Compositions are written, then it should be represented as Histories in each relevant Composition; if it need only be available at some much later point in time (e.g. because it is known that no-one but the treating clinician is interested in it), then it can be stored in another system until sufficient items have been gathered for entry into the EHR.

6.1.2 Instantaneous and Interval Events

The model describes two kinds of "events": instantaneous or "point in time" events, and "interval events". The first kind have a width of zero, while interval events have non-zero width, meaning that their values effectively summarise the actual instantaneous values that must have occurred during the period of the event interval. For all interval events, it is essential to know the mathematical function of the value (which itself is in general a complex structure, such as a blood pressure) with respect to the actual instantaneous values which existed in the real world, and may have been sampled at a fine rate to generate the interval event. These functions include "minimum", "maximum", "mean", "mode" and so on (the full set of possibilities is coded by the *open*EHR Terminology group "event math functions"), and describe the mathematical meaning of all values in the Event data. The function indicator is provided as an attribute on `EVENT<T>` and `SINGLE_EVENT<T>`; in the case of event series, this allows for interval events of more than one mathematical function type to occur in the same history, for example a history of maxima and minima. Such data can be conveniently used for generating sophisticated graphs of the underlying datum over time.

6.2 Class Descriptions

6.2.1 HISTORY<T: ITEM_STRUCTURE> Class

CLASS	*HISTORY<T: ITEM_STRUCTURE> (abstract)*	
Purpose	The abstract parent class of various concrete historical structures, currently including discrete series and series of states, either of which may be periodic.	
CEN	Time was encoded as part of the `Item` structure.	
GEHR	Time was encoded as part of the `G1_HIERARCHICAL_PROPOSITION` structure.	
HL7	The data type History `HIST<T>` is equivalent in intention to `HISTORY<T>`.	
Inherit	`DATA_STRUCTURE`	
Attributes	**Signature**	**Meaning**
	origin: `DV_DATE_TIME`	Time origin of this event history. The first event is not necessarily at the origin point.
Invariants	***origin_exists***: origin /= Void	

6.2.2 EVENT_SERIES <T: ITEM_STRUCTURE> Class

CLASS	EVENT_SERIES<T: ITEM_STRUCTURE>	
Purpose	Defines the semantics of a time segment which consists of events which occur at known instants of time, and which may be periodically spaced. This class is generic, allowing types to be generated which are locked to particular structure types, such as `EVENT_SERIES<ITEM_LIST>`	
Inherit	`HISTORY<T>`	
Attributes	**Signature**	**Meaning**
	items: `List <EVENT<T>>`	The items in the series.
	period: `DV_DURATION`	period between samples in this segment if periodic
Functions	**Signature**	**Meaning**
	is_periodic: `Boolean`	Indicates whether history is periodic.
Invariants	***items_exists***: items /= Void **and then not** items.empty ***periodic_validity***: is_periodic ***xor*** period = Void	

6.2.3 ITEM_EVENT <T: ITEM_STRUCTURE> Class

CLASS	*ITEM_EVENT <T: ITEM_STRUCTURE> (abstract)*	
Purpose	Abstract generic class modelling an event not anchored in time.	
Attributes	Signature	Meaning
	item: T	The data of this event.
	width: DV_DURATION	Length of the interval during which the state was true. Void if an instantaneous event.
	math_function: DV_CODED_TEXT	Mathematical function for non-instantaneous events - e.g. “maximum”, “mean” etc. Coded using *open*EHR Terminology group “event math function”.
Functions	Signature	Meaning
	is_instantaneous: Boolean ***ensure*** width = Void ***implies*** Result	True if this event is instantaneous
Invariants	***item_exists***: item /= Void ***math_function_validity***: width /= Void ***implies*** (math_function /= Void ***and then*** terminology(“openehr”).codes_for_group_name(“event math function”, “en”).has(math_function.defining_code))	

6.2.4 EVENT <T: ITEM_STRUCTURE> Class

CLASS	**EVENT<T: ITEM_STRUCTURE>**	
Purpose	Defines a single event in a series. This class is generic, allowing types to be generated which are locked to particular spatial types, such as EVENT<ITEM_LIST>. In cases where samples are missing, there will correspondingly be missing EVENT instances. Every EVENT instance that is supplied must have an item.	
HL7	The data type HistoryItem HXIT<T> is close to EVENT<T> in its intent.	
Inherit	ITEM_EVENT<T>	
Attributes	Signature	Meaning
	offset: DV_DURATION	Offset of this sample from the origin of the history
Invariants	***offset_exists***: offset /= Void	

6.2.5 SINGLE_EVENT<T: ITEM_STRUCTURE> Class

CLASS	SINGLE_EVENT<T: ITEM_STRUCTURE>	
Purpose	A subtype of `HISTORY<T>` catering for the very common case of single events. The motivation for this class is to reduce the number of temporal objects associated with a datum to one.	
Inherit	`HISTORY<T>, ITEM_EVENT<T>`	
Attributes	**Signature**	**Meaning**
Invariants		

6.3 History Paths

History paths include the following possibilities:

- whole history by name: `"|" HISTORY.name`, e.g. `"|history"`
- whole history by time: `"|origin=<dt>"`, e.g. `"|origin=2001-05-10 16:45:00"`
- event: `"|" HISTORY.name "|" EVENT.name`, e.g. `"|history|event_3"`

Typical paths which refer to a particular item within the spatial data of an event series:

- 16th sample on lead 3 of an ECG (represented as a `ITEM_LIST` structure): `"|history|event_16|ECG_result|lead_3"`
- 3rd sample of apgar breathing datum of a newborn (apgar represented as a `ITEM_LIST` structure): `"|history|event_3|apgar_result|breathing"`
- 2min sample of apgar breathing datum of a newborn (apgar represented as a `ITEM_LIST` structure): `"|history|offset=2min|apgar_result|breathing"`

6.4 History Instance Structures

All data corresponding to events in historical time are represented as histories which ultimately contain one or more instances of a spatial data structure representing a particular instance of clinical data. A history consists of segments of time. Each segment is either a periodic discrete series - a series of time points - or a continuous time section of a certain duration. For each timepoint in a discrete series, there is an instance of the data structure, which might be a list, table, tree or other structure; for each continuous segment, there is one instance, representing the state of something which was true during the duration of the segment.

6.4.1 Single Sample

FIGURE 8 illustrates a single weight measurement in instance form. The event history objects contain the timing information, which in this case is simply the time of measurement (the origin).

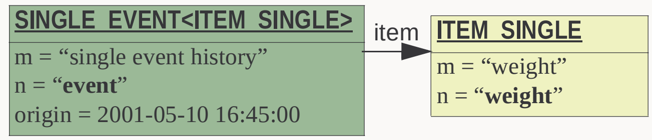


FIGURE 8 Single sample Instance Structure

6.4.2 State History

FIGURE 9 illustrates two time segments representing episodes of chest pain, the first at 5 minutes' offset from an initial event and lasting 5 minutes, the second 15 minutes later, and lasting 15 minutes.

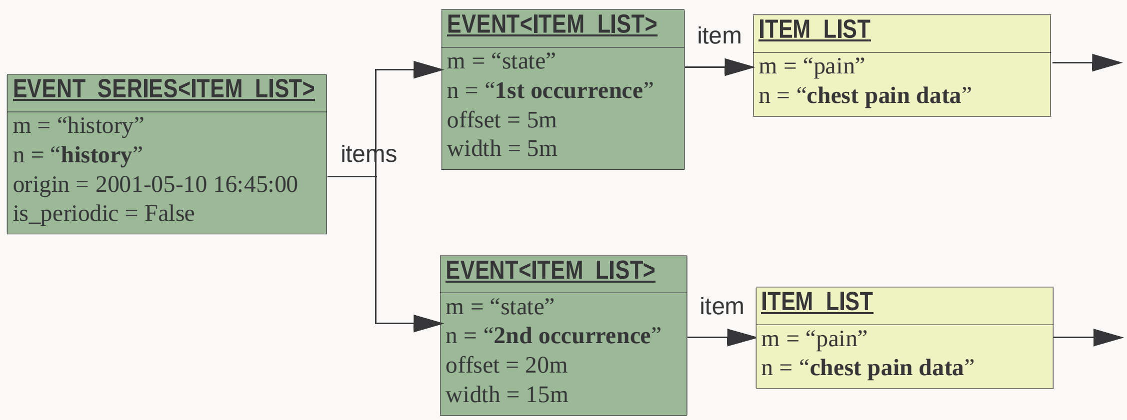


FIGURE 9 State History Instance Structure

A References

A.1 General

1 Beale T. *Archetypes: Constraint-based Domain Models for Future-proof Information Systems*. See http://www.deepthought.com.au/it/archetypes.html.

2 Beale T *et al*. *Design Principles for the EHR*. See http://www.deepthought.com.au/openEHR.

3 Schloeffel P. (Editor). Requirements for an Electronic Health Record Reference Architecture. International Standards Organisation, Australia; Feb 2002; ISO TC 215/SC N; ISO/WD 18308.

A.2 European Projects

4 Dixon R., Grubb P.A., Lloyd D., and Kalra D. *Consolidated List of Requirements. EHCR Support Action Deliverable 1.4*. European Commission DGXIII, Brussels; May 200159pp Available from http://www.chime.ucl.ac.uk/HealthI/EHCR-SupA/del1-4v1_3.PDF.

5 Dixon R, Grubb P, Lloyd D. *EHCR Support Action Deliverable 3.5: "Final Recommendations to CEN for future work"*. Oct 2000. Available at http://www.chime.ucl.ac.uk/HealthI/EHCR-SupA/documents.htm.

6 Dixon R, Grubb P, Lloyd D. *EHCR Support Action Deliverable 2.4 "Guidelines on Interpretation and implementation of CEN EHCRA"*. Oct 2000. Available at http://www.chime.ucl.ac.uk/HealthI/EHCR-SupA/documents.htm.

7 Ingram D. *The Good European Health Record Project*. Laires, Laderia Christensen, Eds. Health in the New Communications Age. Amsterdam: IOS Press; 1995; pp. 66-74.

8 *Deliverable 19,20,24: GEHR Architecture*. GEHR Project 30/6/1995

A.3 CEN

9 ENV 13606-1 - *Electronic healthcare record communication - Part 1: Extended architecture*. CEN/ TC 251 Health Informatics Technical Committee.

10 ENV 13606-4 - *Electronic Healthcare Record Communication standard Part 4: Messages for the exchange of information*. CEN/ TC 251 Health Informatics Technical Committee.

A.4 OMG

11 CORBAmed document: *Person Identification Service*. (March 1999). (Authors?)

12 CORBAmed document: *Lexicon Query Service*. (March 1999). (Authors?)

A.5 Software Engineering

13 Meyer B. *Object-oriented Software Construction*, 2nd Ed.
Prentice Hall 1997

14 Fowler M. *Analysis Patterns: Reusable Object Models*. Addison Wesley 1997

15 Fowler M, Scott K. *UML Distilled (2nd Ed.).* Addison Wesley Longman 2000

A.6 Resources

16 IANA - http://www.iana.org/.

END OF DOCUMENT