The GEHR Object Model Architecture

Authors: Thomas Beale

Revision: 4.1 draft E

Pages: 63

email: info@gehr.org **web**: www.gehr.org

Amendment Record

Issue	Details	Who	Date
1.1 Draft A	Initial Writing	T Beale	Jan 98
1.1 Draft B	First major content	T Beale	16 June 98
1.1 Draft C	Incorporated name changes to knowledge structures - PROP, SUBJ	T Beale	18 June 98
1.1 Draft D	Review by Thomas Beale, Dr Sam Heard	T Beale	24 Jul 98
1.1 Draft E	Pre-HISA Work by Thomas Beale, Dr Sam Heard	TB/SH	29 July 98
1.1 Draft F	Final alterations prior to public review	T Beale	20 Aug 98
1.1	First public review version		10 Sep 98
1.2 draft A	Additions for improved model	T Beale	25 Nov 98
2.0 draft A	Improved introduction; removed software guide, minor model improvements	T Beale	20 Sep 99
2.1 draft A	Major upgrade of text, classes and diagrams	T Beale	7 Oct 99
2.1 draft B	Minor Corrections, better section on archetypes, paths	T Beale	12 Oct 99
2.1 draft C	Removed scenario content to requirements	T Beale	20 Oct 99
2.1 draft D	Incorporated David Lloyd’s review comments	T Beale	24 Nov 99
2.1 draft E	Converted diagrams to UML	T Beale	20 Jan 2000
2.1 draft F	Split systems architecture content into own document. Added UMLS codes.	T Beale	9 Feb 2000
2.1 draft G	Converted archetype model to UML.	T Beale	1 Apr 2000
3.1 draft A	Updated to source version gehr_d1.0.13.	T Beale	5 May 2000
3.1 draft B	Added A_ORGANISER_ROOT class.	T Beale	17 May 2000
3.1 draft C	Corrected clinical class diagrams to show structural connection to HIERARCHICAL_PROPOSITION better; renamed PROPOSITION_CONTENT to DEFINITION_CONTENT; added PROCESS_CONTENT	T Beale	30 May 2000
3.1 draft D	Substantial rework of content and diagrams, including new clusters and naming from Eiffel model.	T Beale	9 June 2000
3.1 draft E	Further text improvements, major rewrite of DATA cluster section.	T Beale	25 June 2000

Issue	Details	Who	Date
3.1 draft F	Major update to record management section, including versioning, exchange.	T Beale	1 July 2000
3.1 draft G	Error corrections; added issue, problem and episode.	T Beale	1 Aug 2000
4.1 draft A	Error corrections; updated for reworked PROPOSITION cluster.	T Beale	20 Aug 2000
4.1 draft B	Units now represented as STRINGs, format according to Regenstrief UCUM proposal.	T Beale	27 Sep 2000
4.1 draft C	BOOLEAN DATA_VALUE type added.	T Beale	4 Oct 2000
4.1 draft D	Clashing DATA_VALUE types renamed to GEHR_XXX	T Beale	10 Oct 2000
4.1 draft E	Alter EHR, EHR_EXTRACT relationship to VERSIONED_TRANSACTION	T Beale	22 Mar 2001

Table of Contents

1 Introduction

1.1 Purpose

This document describes the architecture of the Good Electronic Health Record (GEHR) Object Model (GOM).

The intended audience includes:

- Software development groups using GEHR
- Academic groups using GEHR

This document references the The GEHR Object Model Technical Requirements and requirements in that document in particular. The format of these requirement references is <requirement label> p<page number> appearing in the side column. An example is given of the requirement for faithful recording of information which appears on page 14 of that document.

Req: legal:faithful p19

1.2 Browsing This Document

This document has active web links which will launch your web browser. These links are displayed as a hyperlink and will usually give the web address. If clicking on this hyperlink does not launch your web browser then you can set this up by choosing the Acrobat menu options File > Preferences > Weblink... and completing the dialog box.

1.3 Status

This document is under development, and will be published for inclusion in standards proposals and as documentation for software implementations.

The latest version can be found on http://www.gehr.org.

Work to be included in forthcoming versions:

- Consideration of the notion of “Care pathway”, a time-based overall planning strategy for the patient.
- Consideration of the CEN ENV 13606 notion of “folder”, which currently appears to serve the same purpose as ORGANISER or QUERY_CONTENT?
- Consideration of the CEN ENV 13606 notion of “link” which allows links between two (or more) entries to be stored in another entry.
- Tagging by “health model” is probably required, where values of health model include “western | ayurvedic | homeopathic | naturopathic | chinese | ...”. This is needed for filtering groups of transactions (or smaller fragments? - e.g. GP who prescribes homopathic remedies as well as cough tablets for flu sufferer).
- Role played by recorder/committer at time of committing may be important for filtering also; this info may not be in the demographic snapshot. Ex: GP who is recording data about him/herself might be a “patient”, not a practitioner). Motivation: to be able to filter out “patient” entered data,

"allied health professional", perhaps even particular specialties such as "anaesthetist".

1.3.1 Peer review

Known omissions or questions are indicated in the text with paragraphs like the following:

To Be Determined: not yet resolved

To Be Continued: more work required

Reviewers are encouraged to comment on and/or advise on these paragraphs as well as the main content.

The content of this document is published in Adobe PDF and HTML format.

Please send requests for information and review comments to info@gehr.org.

1.4 Overview

This document commences with a discussion of the formalisms (UML, and the Eiffel language) chosen for the GOM.

The main part of this document describes the formal model, which is a direct reflection of the requirements expressed in The GEHR Object Model Technical Requirements. It has the following layers:

EHR: the definition of the *electronic health record* (EHR), *extracts* of the record and associated semantics;

Transaction: definition of the *transaction*;

Navigation: definition of *organisers*;

Content: *knowledge structures* suitable for encoding clinical (and indeed most types of scientific or general knowledge);

Data: the definition of low level data types, suitable for encoding the values of any clinical datum, from quantities and text to multimedia.

Another part of the model describes the *archetypes* used to define concrete clinical content, that is to say, structures whose job it is to express constraints for legal record content.

2 Modelling Formalism

2.1 Alternatives

2.1.1 Eiffel & BON

see: Del 19 App A3

Of a number of modelling formalisms considered by the original GEHR project, the object-oriented language Eiffel was chosen. It satisfied all the known requirements, with the sole exception that the enumerated type construct (found in most transmission format specifications) is not directly available, but can easily be constructed.

see: "Object-Oriented Software Construction 2nd Ed" by Bertrand Meyer.

Eiffel is particularly strong in modelling semantics and constraint specification (assertions, invariants), while remaining simple - a minimalist approach and Pascal-like keywords make it easy to read. It is also 100% object-oriented, thereby avoiding the type-system ambiguities of other programming languages like C++ or Java in which inbuilt and class-based forms of basic types such as String exist. A further advantage of Eiffel is its concept of a cluster, which is a group of classes. Clusters are recognised by the tools, and enable lower level classes to be hidden in more abstract views of the case tool.

Eiffel also has the advantage of having a graphical counterpart, BON, whose formal grammar maps completely to that of Eiffel, with the sole exception of the aggregation association type (which does not exist natively in any mainstream object-oriented language to this author's knowledge). Eiffel and BON together therefore form an attractive solution for seamless software development.

see: "Seamless Object-Oriented Software Architecture" by Kim Walden and Jean-Marc Nerson.

2.1.2 UML

More recently, a new contender for modelling has appeared: the graphical Unified Modelling Language, or UML. UML is still in development, and takes some of its inspiration from Eiffel, notably constraints (defined in an Object Constraint Language). However, UML is not seamless in the way that Eiffel is: it does not (yet) have a textual counterpart, and cannot be compiled as software. It is also unclear whether bindings between UML and the various transmission and storage formats will be developed. Lastly, not all diagrammatic constructs have unambiguous semantics (as has been the case with previous graphical "languages" such as Rumbaugh and Booch), preventing true seamlessness.

Nevertheless, since UML has become a defacto modelling language, and is understood by a wide audience, it is used in this document to graphically render the Eiffel expression of the model.

2.1.3 Other Alternatives

Numerous alternatives were considered both in the original GEHR project, and more recently, including the following:

OMG IDL: the OMG's IDL language lacks assertions and generic types, and its type model is inconsistent (basic "types" are not the same as constructed types, due to the influence of C);

Rumbaugh/Booch/etc notations: none of these notations are formal, and all lack assertions. In any case, they have been more or less superseded by UML;

SGML/XML: SGML is overly complex, and very document-oriented. XML is simpler, but not well adapted to object modelling. Finding and agreeing a schema DTD would create too many problems. Expressing constraints would be very difficult; it is also difficult to construct a watertight type system in ?ML;

Z, Object Z, B: these are worthy of future consideration. Their use now is prevented mainly by a lack of industrial strength tools.

2.2 Graphical Notation

see: Martin Fowler with Kendall Scott - UML Distilled (2nd Ed.)

The notation used in this document closely follows the UML version 1.3. The following sections describe the major semantic constructs in the class diagrams in this document.

Cluster / Package

A collection of related classes, grouped for convenient management of development. Clusters may be nested hierarchically. Indicated graphically by a named blue dashed rectangle containing classes.

Class

The primary construct in object-oriented modelling and software development. A class defines objects in terms of *behaviour* and *state*, or in more technical terms, routines and attributes. The class definition is the template for creating *objects* at runtime, which are *instances* of the class.

Inheritance

Inheritance is a relationship between classes in which the definition of the descendant (inheriting class) is based on the ancestor. The descendant may change the ancestor's definition in certain ways, according to the rules of the formalism. Inheritance is not normally visible at runtime as a relationship between objects. A number of meanings can be assigned to inheritance relationships, including:

- Specialisation/generalisation
- Implementation re-use
- Facility inheritance
- Taxonomic classification

Association

Association is a relationship between classes which describes a runtime relationship between objects. Its cardinality may be single (1:1) or multiple (1:N). 1:N relationships are represented formally in Eiffel by the use of generic container classes, e.g. *employees*`:LINKED_LIST[PERSON]`.

3 The GEHR Object Model

3.1 Scope

A general comment is in order with respect to the model presented below: it should be remembered that the classes presented are models only of the GEHR requirements, not of a full application or component architecture. As such, it expresses what software engineers call an *analysis model* - the formalised output of a requirements analysis process. As described in the introduction, it is both an agreement on the meaning of the requirements, and an appropriate starting point for constructing software, database schemas, exchange definitions and as such, an appropriate target for standardisation.

3.1.1 Naming

Classes in the model are named according to the following conventions:

- All class names are in upper case, with underscore separators, as is standard in Eiffel and many other languages;
- "basic" classes for which direct bindings into representation formats can be *assumed*, retain their standard Eiffel names, e.g. STRING, LIST etc.
- Classes being *defined* in the model have names of the form: `Gn_XXX`, e.g. `G1_EHR`, where `n` is the version of GEHR.

see: RKmodel:limits in The GEHR Object Model Technical Requirements.

The last provision above ensures that definitions for the same logical class in different versions of GEHR are distinguished; as a consequence there will only ever be one definition of G1_EHR for example, ensuring that there are no surprises for software implementors.

Attribute and relationship names are in lower case, and are logical reflections of the purpose of the the relation. The notation *xxx*`:LIST[G]` denotes a generic class being used to provide a 1:N relation, and is preferable to the informal line/diamond/multiplicity notation found in UML, since the particular generic class chosen conveys the exact semantics. `LIST`, for example, implies ordering and non-unique membership, while `SET` implies unique membership. With the use of UML, both the Eiffel textual and UML graphical indications are given.

3.1.2 Boundaries of The Model

see: section 4.4 on page 47 of The GEHR Object Model Technical Requirements

In the Requirements, the boundaries of the model were discussed. The conclusions of this section are paraphrased here: *any type or container construct for which a native definition does not exist in a target communications formalism (e.g. CORBA IDL) should therefore have an explicit definition in the GOM*, regardless of whether such types might normally be available in the model formalism.

Examples of simple types often used in Eiffel for software development, but which are still explicitly declared in the GOM:

- `DATE`, `TIME`, `DATE_TIME` etc
- `MONEY`
- any enumerated type

All other constructed types, including external references are also declared explicitly in the model. The only assumed types are:

- `CHARACTER`
- `STRING`
- `INTEGER`
- `REAL`
- `DOUBLE`
- `BOOLEAN`
- `ARRAY[G] -> Array<G>` (physical container of items indexed by number)
- `LIST[G] -> List<G>` (implied order, non-unique membership)
- `SET[G] -> Set<G>` (no order, unique membership)
- `HASH_TABLE[G,H] -> Array<Boolean>, Array<G>, Array<H>`

3.1.3 Internal and External References

Req: exch:ext-ref-translat p49

In accordance with the Requirements, references to *external* data are explicitly modelled. While internal references can theoretically simply use the normal referencing mechanism of the modelling formalism, in practice this is only safe where it is known that the target of such references will be transmitted with the object containing the reference, which is only true for a) references within transactions, and b) references between transactions and a record extract (since the extract construct always travels with extracted transactions). References *between* transactions require a “smart” reference which records the key value of the target. Key values can be constructed from any unique feature or combination of features in the target object. In the model that follows, the class `G1_ANY_REF` models the idea of the smart reference.

3.2 Overview

FIGURE 1 illustrates the top-level view of the GEHR object model. It consists of 4 clusters (groups of classes). The following sections describe each cluster in turn.

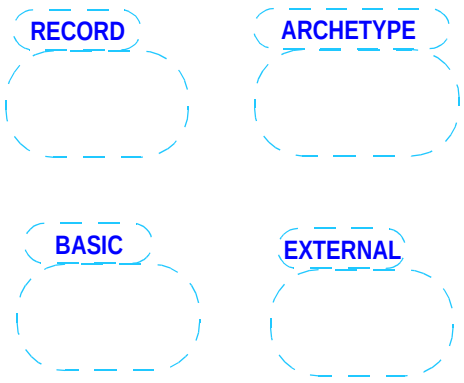


FIGURE 1 Top Level Clusters in the GOM

3.2.1 The Record

FIGURE 2 shows an abstract view of the RECORD cluster. Classes shaded cyan constitute the principal compositional classes in the model.

Classes are grouped into five clusters, as follows:

ehr: The top level record structure is the `EHR` - an explicit model of the "health record". At this level of the model, semantics governing the record as a whole - identification, transmission, merging - are defined. EHRs exist in repositories modelled by `EHR_SOURCE`. Each `EHR_SOURCE` knows its health-care facility (`HCF`). Both `EHRs` and `EHR_EXTRACTs` contain `VERSIONED_TRANSACTIONs`, including lists of persistent and event types.

transaction: The GOM's primary content container - the versioned transaction is defined by the classes `VERSIONED_TRANSACTION`, and `TRANSACTION`. `TRANSACTION` contains an `EHR_CONTENT`, under which content is arranged, and a `DEFINITION_CONTENT`, containing transaction-wide context.

navigation: At this level, the `ORGANISER` and `ORGANISER_ROOT` classes are defined. Their purpose is to give a navigational structure to the record, similar to "headings" in the paper record. Beneath `ORGANISERs`, content is found.

content: This cluster contains the part of the model which defines the fine-grain semantics of record content. In accordance with the discussion of the problem of abstract and concrete knowledge structures in The GEHR Object Model Technical Requirements, this cluster contains content structures for encoding general knowledge structures, using a knowledge representation model. The types `DEFINITION_CONTENT` (contextless statements), `OBSERVATION_CONTENT` (*a posteriori* observations), `SUBJECTIVE_CONTENT` (statements taken as *a posteriori* observations with some level of uncertainty), and `INSTRUCTION_CONTENT` (directions to do something at a certain time or when a condition is satisfied) are recognised. `PROCESS_CONTENT` and `QUERY_CONTENT` types are also included. Content is structured using the `HIERARCHICAL_PROPOSITION` class and its descendants, including `LIST_PROPOSITION`, `TABLE_PROPOSITION`.

data: The data level defines data types by which *values* are encoded. In addition to the expected types for general scientific use, such as `QUANTITY`, `TERM_TEXT`, `MULTI_MEDIA`, and so on.

path: A powerful aspect of the GEHR model is the item locator or "path". Paths are a concept similar to file system paths or URLs, and enable any item in the record to be fully identified by a text string. There is only one path specifier for any given node, ensuring uniqueness of identification in referencing.

3.2.2 Archetypes

Driving the shape of information defined by the concrete model described above, are *archetypes*, i.e. constraint descriptions and rules for valid clinical structures

Req: clin:arch p34

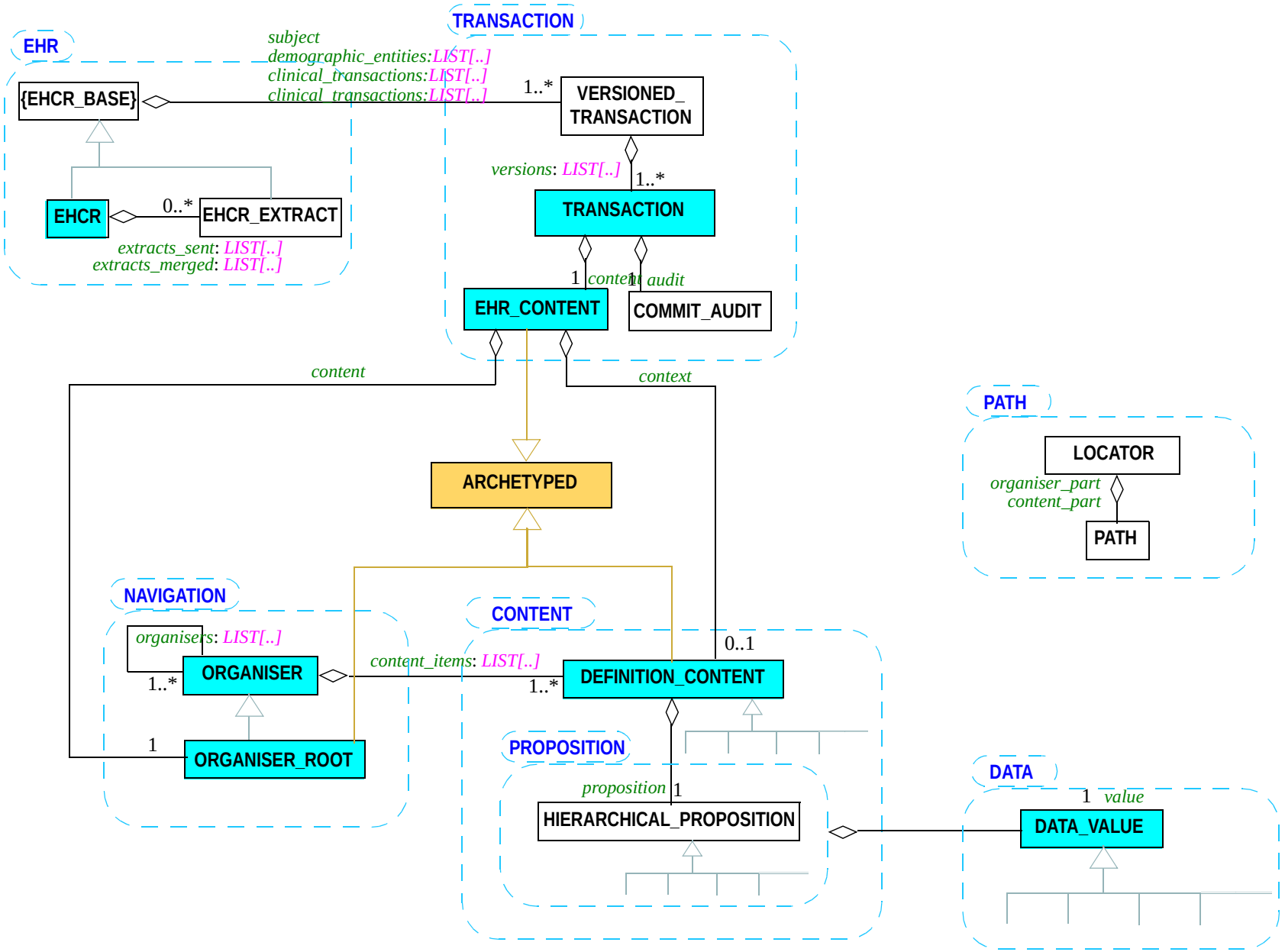


FIGURE 2 RECORD Cluster Overview

which can be built with the concrete classes. Three key classes in FIGURE 2 act as "root" points for archetyping; these are shown inheriting from a class ARCHETYPED.

3.2.3 BASIC Cluster

Basic types are defined in this cluster, including enumerated types for use in status variables elsewhere in the model.

3.2.4 EXTERNAL Cluster

A number of classes are defined to enable content within the record to refer to entities outside the record. The most obvious example is the URI, or unique resource identifier defined by the W3C consortium.

3.3 EHR & TRANSACTION Clusters

3.3.1 Overview

The content of a GEHR EHR consists of versioned transactions, each corresponding to a single *concept*. (In this document, the term “transaction” is often used to mean versioned transaction. To refer to a particular version, the term “version” or “transaction version” will be used). Like certain other entities in the GOM, versioned transaction content created by archetypes which define the concept and other details about transactions. All versions of a transaction correspond to the same concept and archetype.

Transactions are of two broad types: *persistent* and *event*; this distinction is mostly a logical one, with minimal underlying difference in the model. Persistent transactions are those which remain valid in time; they pertain to facts which can always be considered true of the patient (to the best of anyone’s knowledge). Event transactions record care actions carried out to/for the patient in time, but whose informational validity cannot generally be assumed to be durable.

Examples of persistent information include both the clinical, such as “family history” and “current medication” but also demographic entities which are referenced in clinical content, and the patient him/herself, known as the “subject” in GEHR. Examples of event information include the information recorded during contacts, e.g. subjective wellness, blood pressure, etc, and also test results, hospital admissions, discharges and so on.

FIGURE 3 illustrates an example of an EHR containing persistent and event transactions.

The remainder of this section describes the formal model of the GEHR EHR, which is illustrated in FIGURE 4, and the rationale for its semantics.

3.3.2 EHRs

The `EHR_BASE` class provides the common features found in both normal EHRs and extracts. `EHR_BASE` consists of four groups of `VERSIONED_TRANSACTIONs`: the *subject_transaction*, *demographic_entities*, *persistent_clinical_transactions*, and *event_clinical_transactions*, as illustrated in FIGURE 3.

Req: clin:record-concept p26

The first of these provides every `EHR` or `EHR_EXTRACT` with patient identification information, which in this model is a persistent transaction whose contents describe the subject of care.

Other demographic entities representing clinicians, recorders, HCFs and so on occur throughout the GEHR record, both in content context classes and in the transaction audit classes shown in FIGURE 4. These are maintained in their own transaction list, *demographic_entities*. The rationale for versioning demographic entities (including the subject) is discussed section 3.3.7.

Finally, clinical information of persistent and event types is found in the *persistent_clinical_transactions*, and *event_clinical_transactions* lists.

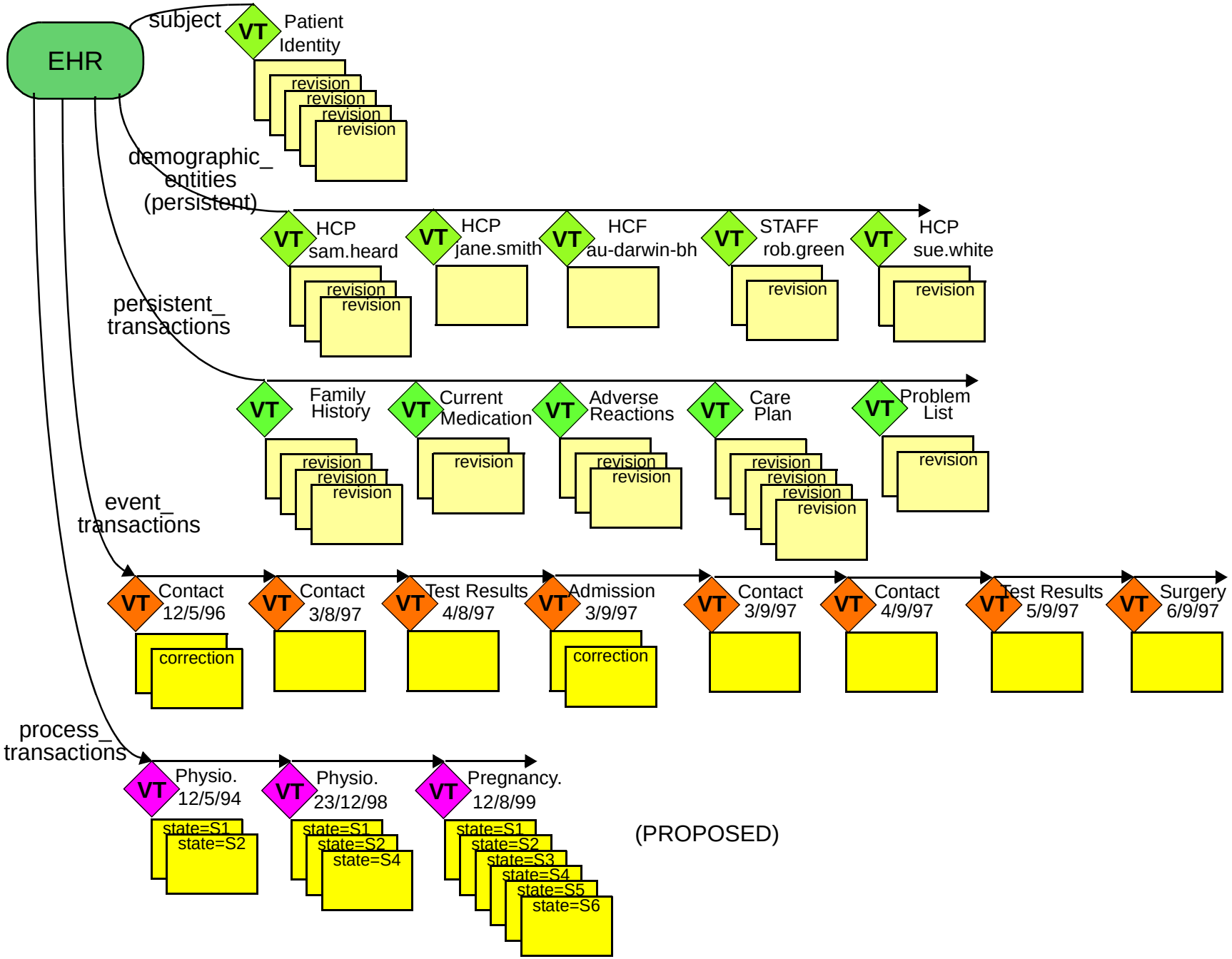


FIGURE 3 Versioned View of EHR

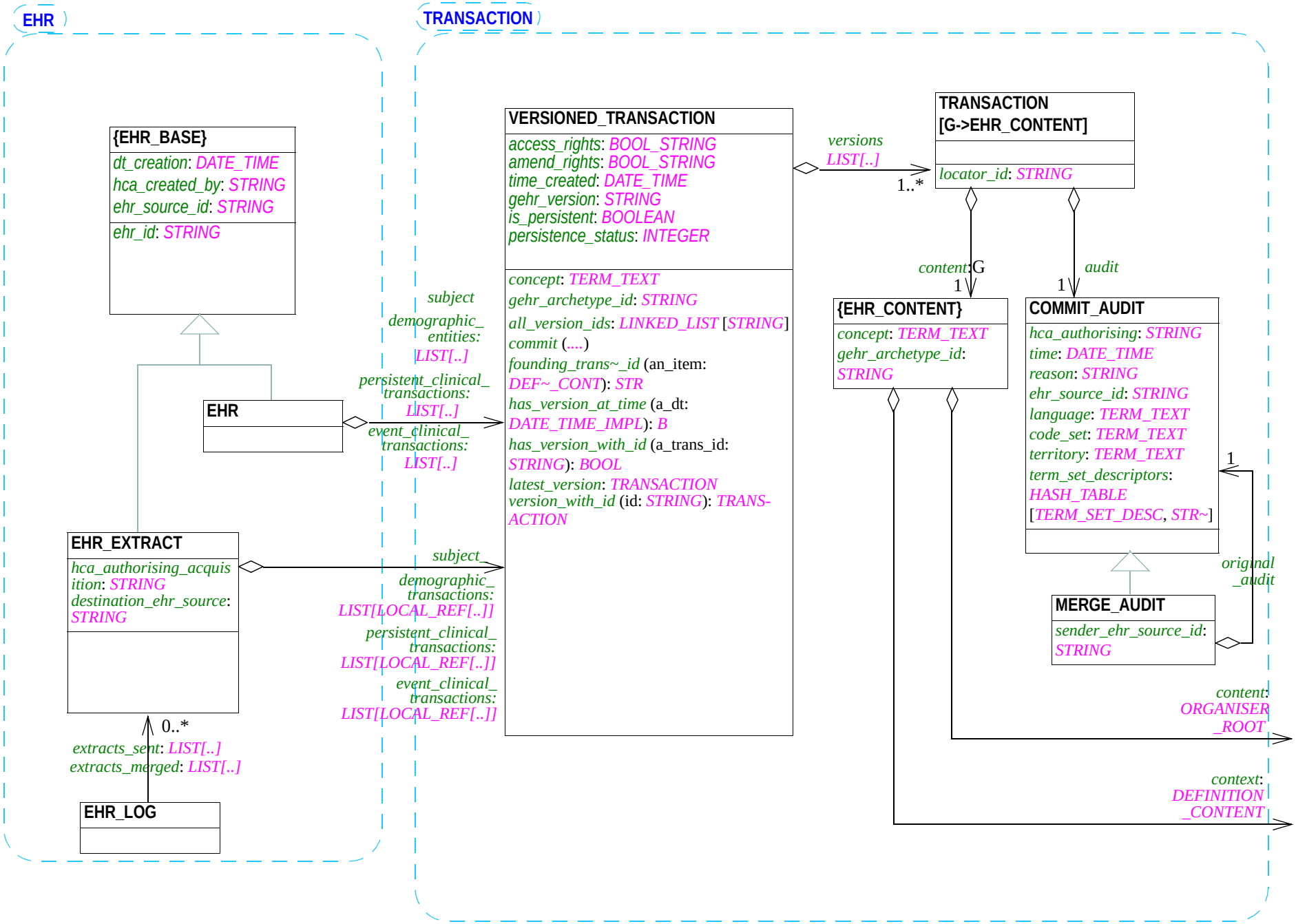


FIGURE 4 EHR and TRANSACTION Clusters

3.3.3 EHR Extracts

EHR_EXTRACTs are used to send selected versions of transactions at one EHR source to another. They contain an selection of VERSIONED_TRANSACTIONs from an EHR, each of which contains only the latest version from its originating VERSIONED_TRANSACTION.

The *extracts_created* and *extracts_merged* lists in an EHR_LOG record the EHR_EXTRACTs created locally and those received from elsewhere, respectively. The first list is used to differentiate between extracts created elsewhere, and those created locally, transmitted sometime in the past to other HCFs, and finally received as part of an EHR_EXTRACT from elsewhere.

The subject of the EHR (i.e. the patient) must be the same for all EHR_EXTRACTs merged. Human intervention may be required to check this (see Exchange And Merging below).

3.3.4 Versioned Transactions

Req: legal:non-erasure p22, legal:record-history p22

The purpose of the VERSIONED_TRANSACTION class is to provide a version control interface for creating and accessing the versions of a single logical transaction. Transactions are classified as *persistent* or *event*. Persistent transactions are more likely to have many revisions over time, whilst event transactions should have very few.

Examples of the *event* transaction type are contacts, progress notes, laboratory test results and drug administration records. *Persistent* transactions on the other hand include summaries, care plans and family history, and any other type which remains valid in the long term. For all transactions, the current information is available in the most recent version.

A new transaction version or versioned transaction is created for each interaction between a health care agent (either an HCP or a device) and the EHR. For event information, each distinct care event which creates new clinical information, causes a new VERSIONED_TRANSACTION to be created, whilst corrections to existing transactions cause new transaction versions. After a correction is committed, prior versions are by definition ‘incorrect’, and are only of interest when backtracking is required.

For persistent information, a new version in an existing VERSIONED_TRANSACTION will be created for an update to the information contained therein; information on a new theme will cause a new VERSIONED_TRANSACTION to be created.

New TRANSACTIONs in a VERSIONED_TRANSACTION will also be created where state variables need to be updated, for example in time-based processes modelled by PROCESS_CONTENT and INSTRUCTION_CONTENT.

3.3.5 Transaction Versions

Req: clin:trans-concept p26

TRANSACTIONs constitute the containers for clinical content in the GEHR EHR. A TRANSACTION is the combination of *content* and *audit* objects

Audit information is recorded in COMMIT_AUDIT, including the details of time committed, authorising health care agent, reason for new version, as well as locali-

sation information like language locale (*language*, *territory*, *code_set*), date/time formats, and so on, which are essential for textual information to be interpreted properly when sent elsewhere. The HCA (health care agent) referred to by a COMMIT_AUDIT object represents the person or device taking responsibility for committing the contents of the transaction, but it does imply that the same person or even one only person actually created the content. In an accident and emergency ward for example, multiple clinicians may create content simultaneously in a multi-user session; the person committing will usually be one of them, but may be another person altogether.

The descendant class MERGE_AUDIT supports merging of transactions from elsewhere, by providing an audit trail for the merge, while retaining the COMMIT_AUDIT object from the original commit. Both COMMIT_AUDIT and MERGE_AUDIT implement a *locator_id* function, which returns the globally unique identifier of the transaction; for locally created transactions, it is the triplet <*ehr source id*, *hca id*, *commit time*>, while for merged transactions, it is the same triplet, taken from the original commit audit object. In other words, regardless of how transaction versions are copied around, their global ids are immutable.

The content of a TRANSACTION is provided in the EHR_CONTENT class, which may be regarded as the starting point in the GOM for content of any kind (as opposed to the information management structures of transactions, versioning, and EHRs).

EHR_CONTENT has *content* and *context* attributes; the first is where the "information" of the transaction is found, the latter for "contextual" details, describing how/where the content came about. There are two broad categories of information which are expected to be constructed with EHR_CONTENT, namely, clinical and demographic. Currently these are not explicitly differentiated in the GOM - it is left up to archetypes to define the structures. However, as a guide, it is suggested that the archetypes be devised according to the following table.

	EHR_CONTENT **attribute**	
Information Category	*content* (ORGANISER_ROOT)	*context* (DEFINITION_CONTENT)
Clinical	Clinical content expressed in descendants of DEFINITION_CONTENT, organised under headings modelled by ORGANISER/ ORGANSIER_ROOT structures.	Clinical context, according to a broad classification of transactions such as: contact, report, summary, trigger and so on. Context for a contact would include: start and end times of consultation.
Demographic	Demographic entities expressed using DEFINITION_CONTENT structures, but with only a default ORGANISER/ ORGANSIER_ROOT structure, since headings are not needed. Example of HCP: - name - unique_id - other_identifiers	Origin of demographic information, including: - id of originating system - time of last update on originating system - method of conversion to GOM structure

This approach avoids the use of a concrete demographic model within the GOM, while ensuring that demographic entities do include a minimum of basic data. It should also be remembered that in the long term, demographic transactions for providers and health care facilities may not be needed, assuming interoperable online repositories of demographic reference data become available, although patient transactions are likely to be required, since online patient repositories are far more problematic from the privacy point of view.

3.3.6 Semantics of Versioning

The versioning concept used in `VERSIONED_TRANSACTION` is similar to that found in file versioning mechanisms such as SCCS and RCS. The contents of a `VERSIONED_TRANSACTION` should always be understood as being one logical entity, whose successive states in time (as well as auditing and context information) are captured in a series of `TRANSACTION` objects. Accordingly, the first `TRANSACTION` of a `VERSIONED_TRANSACTION` is created from an archetype and uses the attributes *concept* and *gehr_archetype_id* to record the logical concept and generating archetype id respectively; the `VERSIONED_TRANSACTION`'s *archetype_id* and *concept* are taken from this.

The process of creating a new version, including a first version, by an application or other calling software takes place as follows:

- If modifying:
  - Obtain the latest version of a `VERSIONED_TRANSACTION`, via the *latest_version* feature; calling this will return a `TRANSACTION` object.
  - Obtain the transaction identifier of latest version. This will be the parent version identifier of the intended new transaction.
  - Modify the `EHR_CONTENT` referred to by the *content* attribute of the `TRANSACTION` as required, using the relevant archetype.
  - Modify the *context*:`DEFINITION_CONTENT` attribute of the `TRANSACTION`, again using the correct archetype, if needed (e.g. if updating a demographic transaction).
- If creating the first version:
  - Create a new piece of *content* (of type `EHR_CONTENT`) using ehr_content, content and organiser archetypes.
  - Create a new `DEFINITION_CONTENT` representing the context.
  - The parent version id is `"first"`.
- Commit the `EHR_CONTENT` by calling a *commit* function in a factory object in the GEHR kernel, supplying a reason for change, and the authorising and legally responsible HCPs. Note that no "checking-out" was required to do this.
- The factory object calls `VERSIONED_TRANSACTION`.*commit*, passing its arguments, plus relevant locale information (language etc). The identifier of the latest version is compared to the parent identifier taken at the time of modification/creation. If these are not the same, another modifier has since committed changes, and the current modifications are potentially in

conflict with the new latest version; a merge resolution operation has to be performed, which may require the intervention of a human operator.

- The new content and other information is stored as a new version. If VERSIONED_TRANSACTION is implemented efficiently, this may mean that only changed objects are stored; conversely, a simple implementation would simply store each subsequent version *in toto*.

The only other way new versions can be added to a VERSIONED_TRANSACTION is if they are merged from an extract coming from another EHR source (see below).

Modification can only take place using the same archetype as in the previous version of the same transaction (given in the EHR_CONTENT.*archetype_id* attribute), or a compatible archetype. If a new version is added by merging, the archetype of the merged transaction must relate to the archetype previously used according to the same rules for locally created content.

To Be Determined: exact rules for archetype compatibility have not yet been determined.

Access to previous states of a logical transaction is provided via various features defined on VERSIONED_TRANSACTION:

- *version_with_id* (*trans_id*:STRING): TRANSACTION
- *version_at_time* (*a_committal_dt*:DATE_TIME): TRANSACTION

As indicated earlier, transaction ids are the triplet of *<EHR source, HCA id, date/time committal>*, so in fact the second function is really a form of the first. It is important to understand that the version at a previous point in time represents a *previously available informational state* of the EHR, at the particular EHR source. Such previously states include only those transactions from other EHR sources as had been merged at the that point in time, *regardless of whether the merged information pertains to clinical information recorded earlier.*

We can now begin to see that there are actually two candidates for a historical view of a versioned EHR:

Informational: previously available informational states of the actual record.

Clinical: virtual previous clinical descriptions of the patient, or in other words, previous states of a notional virtual record.

It is the first with which we are concerned for medico-legal purposes, since it represents the information actually available to clinicians at an HCF, at some point in time. But the second view may be useful for reconstructing an actual sequence of events as experienced by the patient.

To Be Continued: requirements for previous states of a virtual record?

3.3.7 Versioning of Demographic Information

Demographic information includes any information pertaining to identified real world entities, which generally come under the categories people, organisations, and places. Modern demographic models often refer to people and organisations as “parties”, with details of names, contacts, addresses, and so on being attached to some specialisation of a “party” class.

GEHR systems assumed the existence of a demographic service external to the EHR server, no matter how simple this may be. This appears to be a reasonable assumption, since a demographic service, whether in the form of a hospital PMI (patient master index) or a simple patient database in a GP setting, are almost always the first step in computerisation of health facilities.

In a naive version of a GEHR system, demographic entities could be recorded simply by using their key values from the external server. This would allow the EHR server to avoid modelling demographics at all, and would guarantee that only the latest (most correct) demographic information was used at any point in time, since live retrievals would always be done using the keys in the EHR.

Unfortunately, this approach does not address the following requirements:

- The state of demographic data at the time transactions were committed is required for medico-legal purposes. Like clinical information which is versioned, the past states of demographic entities could be relevant in legal or clinical investigations. For example, the professional qualifications, position, and even profession of health care professionals changes over time; decisions made when a student may not have the same legal ramifications as when a principal surgeon.
- A minimum of particular demographic data is required to be recorded in the record to guarantee that extracts are comprehensible to receivers who have no access to the sender's demographic service (or any equivalent).

Whilst it might be hoped that one day public demographic information will be globally available online, with a secure service infrastructure such as that proposed by the CORBAmed PIDS specification (see [8.]), it is certainly not the case today, and the limited coverage, incompatibilities and wildly differing levels of sophistication mean that for the moment, EHR proposals such as GEHR define a minimum demographic model.

Consequently, the GEHR approach is that an EHR server is supplied snapshots of demographic information sourced from external servers, which it then treats as any other data entering the record: it is added to a versioned transaction, and becomes part of the versioned record.

However, GEHR does not provide a demographic model as such, but takes an approach based (as usual) on archetypes. Archetypes are used to define basic models of the patient, HCF, and various types of HCP, all in terms of `ORGANISERs` and `DEFINITION_CONTENT`.

3.3.8 Exchange And Merging

Logically, transactions are the base unit of transfer in GEHR, that is to say, data is never transferred in smaller fragments than transactions. What does this statement really mean, given the non-trivial semantics of versioned content?

At the logical level, requestors require certain transactions, specified for example by a filter of some kind, e.g. "all diabetes transactions". In many cases, the transactions they are requesting do not exist in their system; effectively they are requesting new `VERSIONED_TRANSACTIONs` from the sender, which have never been received before. This will be the case with almost all event transactions.

However, with persistent transactions, the requestor system may well already have earlier versions of what they request, as would often be the case for “family history” or “current medication” transactions. In these cases, they are really asking for an “update”. When updates are received, the intention is to be able to *merge* them into existing versioned transactions, thus adding more recent modifications to a logical item of content, even though the new versions were created elsewhere.

Let us consider more closely the possible requirements for transfer of versions of transactions:

- Requestor wants the *latest version* of a transaction only. This is the most common type of request.
- Requestor wants *all versions* of a transaction. This might be because:
  - They are conducting a medical investigation into an error;
  - They want to review the multiple states in time of a health summary, family history, or other persistent transaction.
  - They want to review the progress of states of an instruction or process in time, e.g. for clinical quality assurance or educational purposes.
- Requestor has previously received some/all of the versioned transaction in question; they now want an update, i.e. *all versions since the latest one they have*.

A number of requirements hold regardless of which particular content the requestor asks for:

- The requestor usually wants the content only for the most recent version, but wants the auditing information for other versions. This applies in particular where versions contain large multimedia data items. Thus the ability to supply audit information without content is required.
- Even for requested content in latest versions, the requestor needs to have the option of including large data items inline or references only.

We must also consider the requirements of how versioned information is to be understood at any source:

- After merging transaction versions from elsewhere (including those for which there are no earlier versions at the receiver), the receiver must be able to differentiate between the time at which the transactions were added to the record (i.e. the point in time at which this information was available locally), and the time at which the information was committed in its original transaction (i.e. the time it was *available* at its source).
- The receiver must also be able to determine which HCP authorised the inclusion of merged transactions.
- The receiver must be able to determine both the original EHR source at which a transaction was created, and the EHR source from which it was most recently obtained.
- Where duplicates of a `TRANSACTION` are received over time (due to different requests), the receiver does not want to erroneously merge the

duplicates again, but still wants to record the fact and time of receipt, in case of medico-legal investigation.

To Be Determined: This situation not yet addressed in the model or kernel.

From the above we can define the semantics for transactions with respect to exchange:

- VERSIONED_TRANSACTIONs can only occur at an EHR_SOURCE by being created there. Consequently, to merge TRANSACTIONs from elsewhere when the receiver has no prior versions of the same transaction (and therefore no corresponding VERSIONED_TRANSACTION) requires the receiver to create a new VERSIONED_TRANSACTION into which received TRANSACTIONs can be merged.

  Doing otherwise - i.e. directly introducing VERSIONED_TRANSACTIONs from elsewhere prevents local merge audit information being recorded easily, and violates the local HCFs' right to manage all additions to the record.
- EHR_EXTRACTs are constructed by including required VERSIONED_TRANSACTIONs in one of two forms: either complete (with all their versions) or with only the latest version. Wherever merging occurs, MERGED_AUDITs are created, ensuring the audit information for the merge operation is captured, without removing the original audit information. This operation works even if the received transactions had already been merged from elsewhere.

The exact dialogue between requestor and sender is not specified here. However, it is useful to clarify which transaction attributes need to be preserved when transaction extracts are transmitted, including multiple times.

The first group is includes localisation details which are needed to make sense of text, dates and so on; thus: *language*, *code_set*, *territory*, date/time format, locale name etc found in the COMMIT_AUDIT class.

Next, we want to ensure commit audit details are transmitted as well, since it is useful to know where, when and why the information was committed in the first instance to the patient's record. Thus the following attributes, also from COMMIT_AUDIT: *hca_authorising*, *time*, *reason*, *ehr_source*. The original COMMIT_AUDIT object is thus retained by the recipient, along with its own merge audit details.

Finally, the same attributes as for commit are used to record audit information due to merging.

The question is: What do we need to send in an extract for a transaction which has already been merged from elsewhere? We can answer this by considering what happens if there is a medico-legal investigation. Most investigations will look only at the content that was available in an EHR source at a given time in the past; no more information than described above is needed for this.

However, in the (one imagines) rare event that an investigation needs to determine the history of exchange of a transaction - i.e. its "voyage" through various EHR sources - one more detail is required in an extract, namely the id of the sending

EHR source. The identifier of the transaction itself is not even needed, since transaction identifiers are immutable; all that is needed to find a transaction in the EHR source from which it came is to search for the transaction there - the normal retrieval mechanism will find it, and so on until the origin is reached.

3.4 Clinical Content

Three clusters contain classes which define the structural semantics of clinical content: NAVIGATION, CONTENT, and PROPOSITION. These are shown in detail in FIGURE 5.

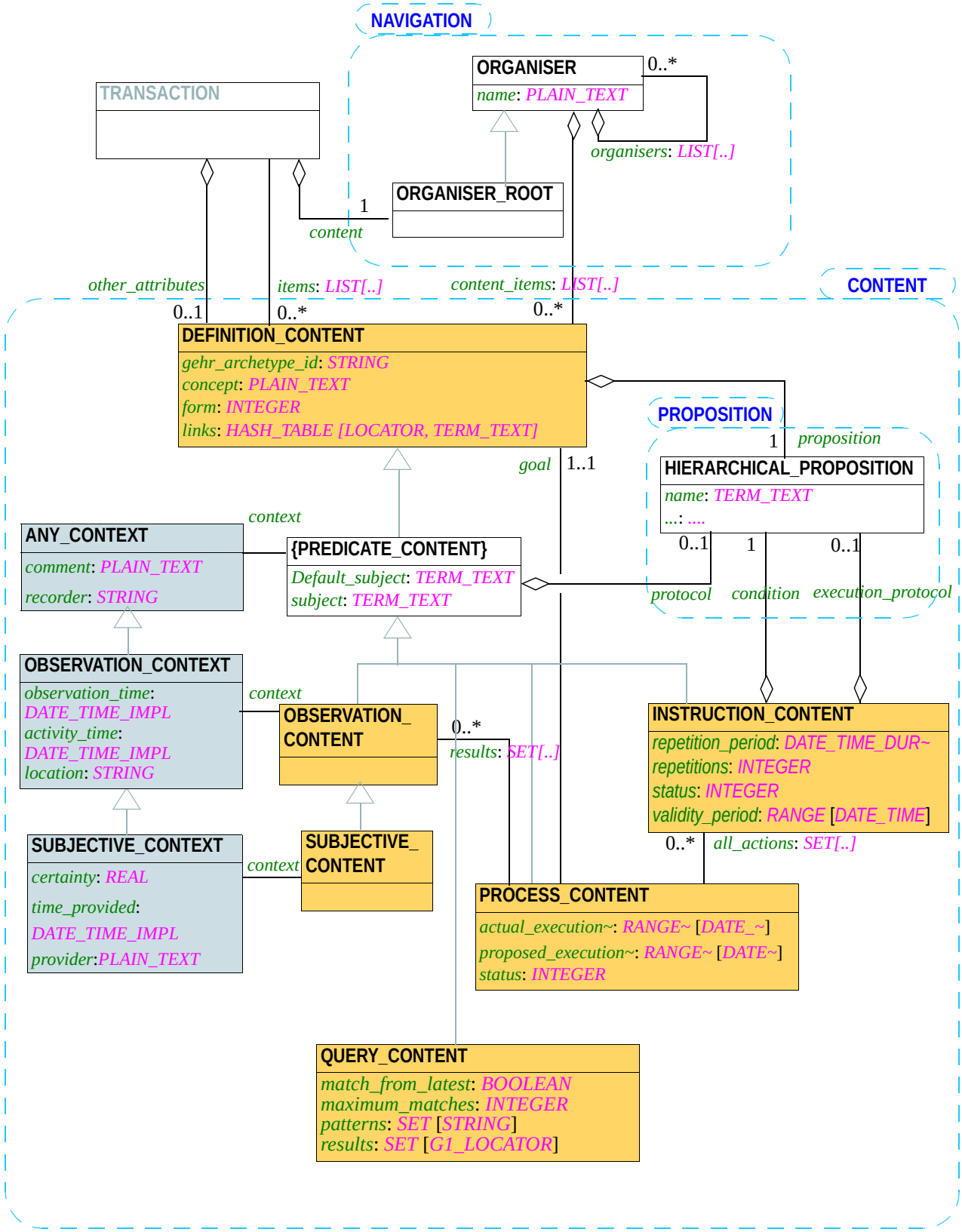


FIGURE 5 RECORD / NAVIGATION and RECORD / CONTENT Clusters

3.4.1 NAVIGATION Cluster

Organisers enable the construction of hierachical classification structures in which content entries are placed. They are similar in concept to “folders” in a file system, in that their role is to provide a navigational structure via which content items can be found. FIGURE 6 provides a instance view of the contents of a transaction within an example organiser structure (Weed’s SOAP model in this case). Content items are represented by “xC” elements (standing for `XXXX_CONTENT` objects), the root objects of content structures (see below). Each logically self-contained tree of organisers starts with an `ORGANISER_ROOT` object, which is the same as a normal `ORGANISER`, but can be archetyped - it thus records the archetype identifier, and concept, in the case below, “SOAP Organisers”.

It is the organiser structures which provide the default visual navigation structure within the transaction when it is recorded. Organisers can recursively contain other organisers, which finally contain content, allowing for extensive navigation structures, e.g. a “Problem” level of headings followed by the “SOAP” headings under each problem. Organiers may also contain ORGANISER_ROOT objects, allowing separately archetype-created subtrees to be included. The practical effect in the example below would be to allow the addition of new groups of sub-headings under each of the SOAP headings, each group defined by some other archetype.

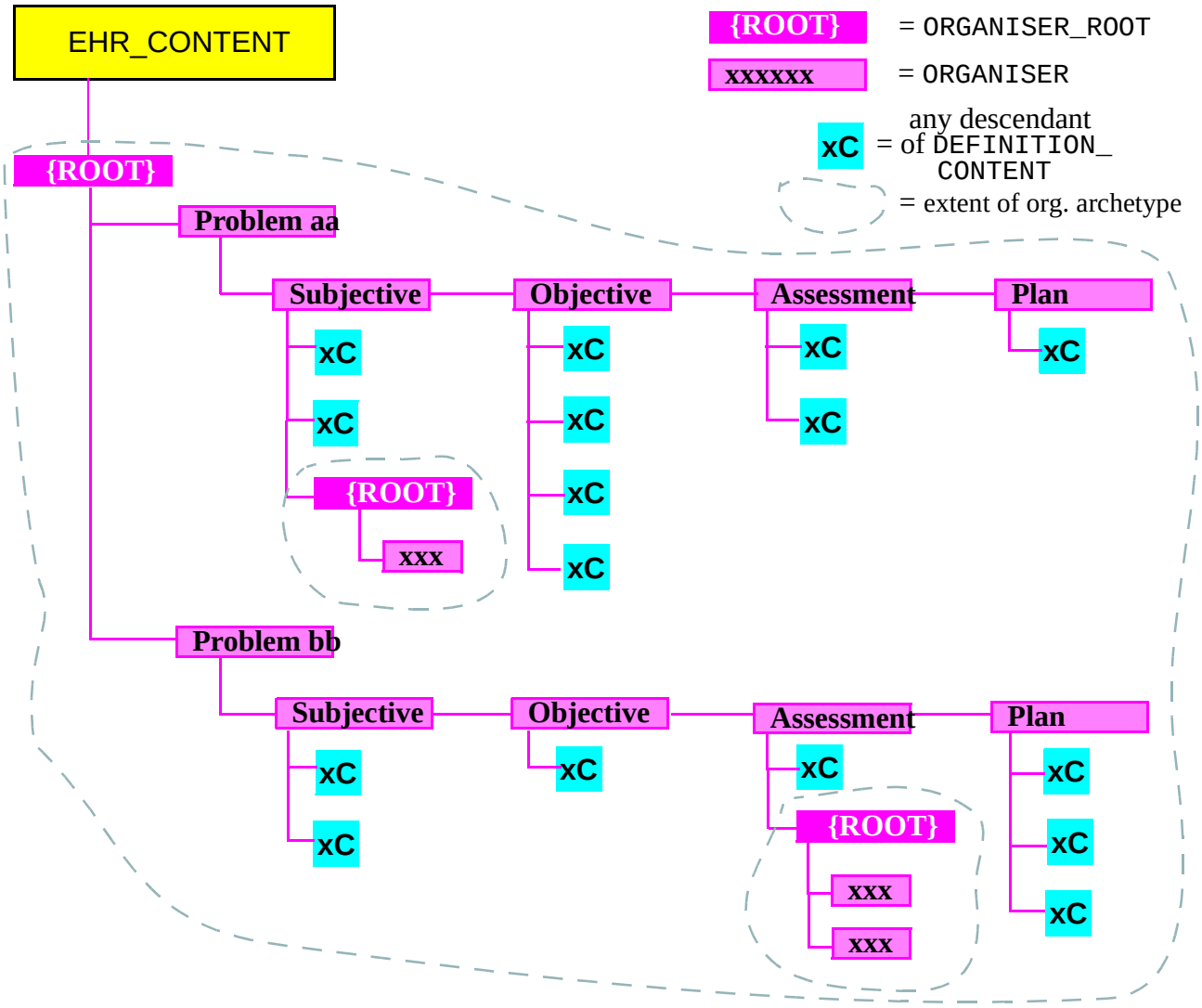


FIGURE 6 Organiser View of a General Practice Contact Transaction

3.4.2 CONTENT Cluster

Content is created under organisers and has two dimensions: its *knowledge species* and its *structural form*. Knowledge species are modelled as subtypes of `DEFINITION_CONTENT` (see orange-shaded classes in FIGURE 5):

`DEFINITION_CONTENT`: knowledge without context - *a priori* statements with no implied meaning with respect to any real world entity or process. Example: “weight = 85kg”.

`PREDICATE_CONTENT`: (abstract type). Content of this type relates to a subject, e.g. a patient.

`OBSERVATION_CONTENT`: *a posteriori* observations of real world phenomena about a subject. E.g. “patient’s weight is 85kg (as measured on mechanical scales)”.

`SUBJECTIVE_CONTENT`: reported opinions about real-world phenomena about a subject, e.g. “I believe father’s weight is 85 kg”.

`INSTRUCTION_CONTENT`: commands to do something for subject, e.g. “take one tablet three times a day before meals”.

`PROCESS_CONTENT`: a content type which models processes relating to the subject in which other content types figure as observations, orders, results, and goals.

`QUERY_CONTENT`: a content type which contains the definition and results of query. Queries are specified by match patterns for `LOCATORs`, and the results are recorded as a set or `LOCATORs`, avoiding the need to copy any content in the record.

Each of these types carries a set of context information, which can include:

protocol: how something is/was done

date/time observed: when the observation was made

recorder: who recorded the information: may be the clinician, or another staff-member

date/time provided: when subjective information was provided by the subject

provider: who provided subjective information: may be the subject, a relation of the subject, or the clinician.

certainty: percent probability of correctness ascribed by the clinician to subjective information.

Note that date/time of recording is stored in the transaction.

see: Clinical Knowledge Representation on page 26 of The GEHR Object Model Technical Requirements

Content is archetype-configured. Concrete structures are only provided for the generic knowledge representation structures referred to above, which along with the navigational organisers appear to cover the knowledge types likely to be required in a medical record (and indeed in any scientific record). The model chosen does not attempt to accurately reflect a purist version of the species of knowledge identified in epistemology (if such model even exists), but rather pragmatic aspects of actual scientific, clinical, and medical data and process. Hence, the types have been chosen by considering examples in the medical, clinical and scientific domains, as discussed in the Requirements.

The main motivation, in health record terms, for differentiating between these types is to define the contextual information required by each. These contextual attributes are described in the Requirements, and are directly reflected in the model illustrated in FIGURE 5.

3.4.2.1 Linked Content

Content trees may be related to each other via *links*, enabling *threads* to be formed. Links are illustrated in FIGURE 7. Links express relationships between separate content items, such as “in response to”, “request for pathology”, and in general, provide the means of causally connecting observations,

diagnoses, requests for pathology, test results and so on. Links thus provide support for a *process view* of events.

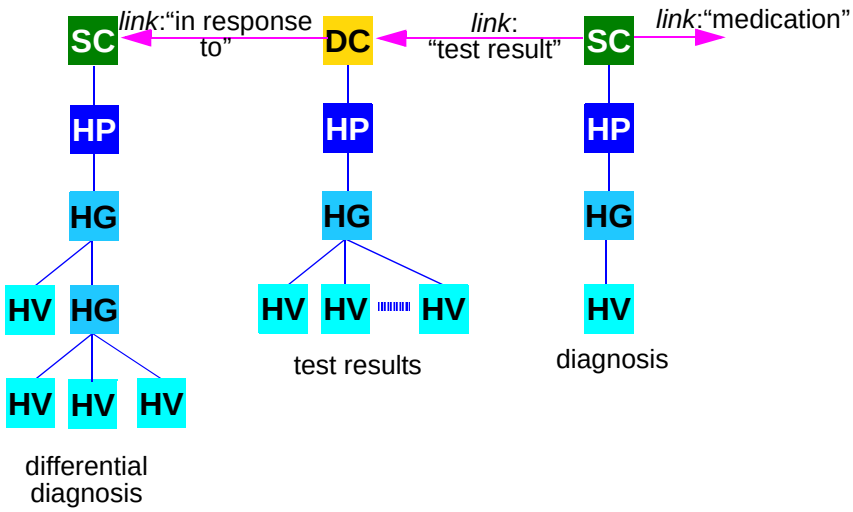


FIGURE 7 Linked Content structures

To Be Determined: whether links will use live refs or always use paths. Paths are safer and do not lose validity if transmitted without the target.

3.4.3 PROPOSITION Cluster

Classes in the PROPOSITION cluster define how data within XXX_CONTENT objects is represented. The intention is twofold:

- To provide a diversity of logical interfaces to content, such as single values, lists, tables, time series and so on.
- To define a standard physical representation - the hierarchy - ensuring that any content structure will always be comprehensible to old and new GEHR software, regardless of what logical interfaces are introduced over time

FIGURE 8 illustrates the internal structure of the HIERARCHICAL_PROPOSITION classes, along with subtypes representing various logical interfaces. The *value*:DATA_VALUE attribute of HIERARCHICAL_VALUE is where clinical structures connect to GEHR datatypes.

Req: clin:struct-hier on page 28

A standard hierarchical structure is defined by the generic classes in the PROPOSITION cluster: HIERARCHICAL_PROPOSITION is the hierachy root object, non-leaf nodes are HIERARCHICAL_GROUP objects, and leaf nodes are HIERARCHICAL_VALUE objects. Each node is capable of carrying a context object. The HIERARCHICAL_GROUP and HIERARCHICAL_VALUE object optionally carry context values; which override the values of the context attached to the HIERARCHICAL_PROPOSITION root object are assumed.

Together, the HIERARCHICAL_XXX classes provide the ability to build hierarchies of objects of any context type. To build a complete content item of a particular type, a root object fixes the type of the hierarchy. The XXX_CONTENT classes are used for this purpose; each of these classes has a *proposition*:HIERARCHICAL_PROPOSITION feature, linking it to its data hierarchy. The XXX_CONTENT root object also records the generating archetype name, and is the place where cross-reference links between content items are stored. The typical structure of a full content item (observation in this case)

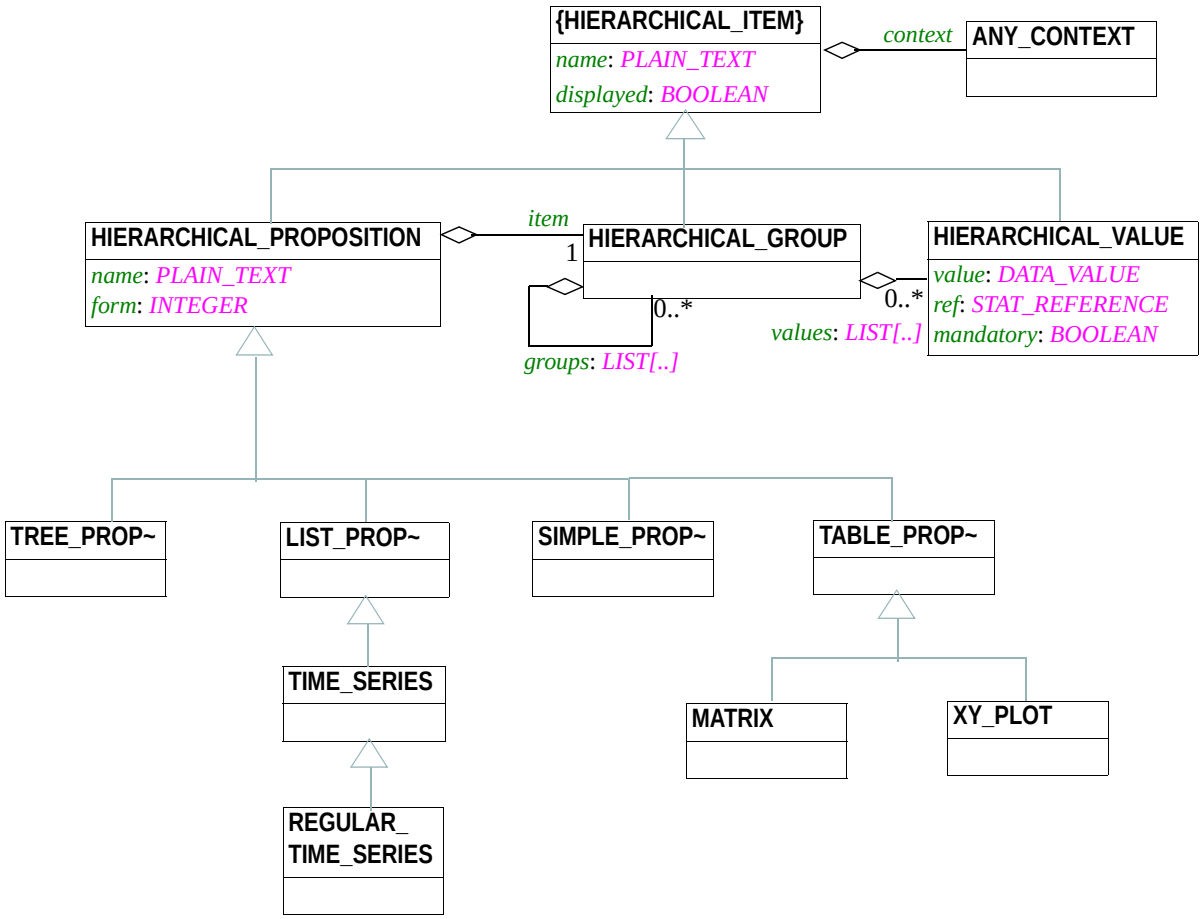


FIGURE 8 RECORD / CONTENT / PROPOSITION Cluster

is illustrated in FIGURE 9. (Note that figures elsewhere in this document generally omit context objects for brevity).

3.4.3.1 Form

How are alternative structures such as tables and time-series achieved using this model? Although the model defines hierarchical structure only, in fact any kind of table, tree, or series can be constructed, as long as a known discipline is used. The *form*:`INTEGER` feature in `HIERARCHICAL_PROPOSITION` indicate the logical structure type. Some of the possibilities are illustrated in FIGURE 10 and FIGURE 11, in which archetypal clinical data structures are shown, along with clinical data examples.

Req: clin:struct-table on page 28

FIGURE 11 shows how tabular information is represented using the standard hierarchical structure of the model. In each case, a known discipline is followed, and the logical structure type is recorded, enabling software to always process the hierarchy as the correct logical structure. Various kinds of tables are possible, including “table” (the relational concept) and “matrix” (named columns and rows, data occupies cells).

The particular encoding of logical data structures into hierarchies chosen here is is an example of what could be standardised on. With the idea of form, the GOM retains the advantage of using only one form of physical representation (hierarchy), while allowing the possibility of adding functional interfaces to logical data structures, making the business of database administrators much easier. Currently, logical interfaces for different forms have not been included in the GOM, but they are likely to

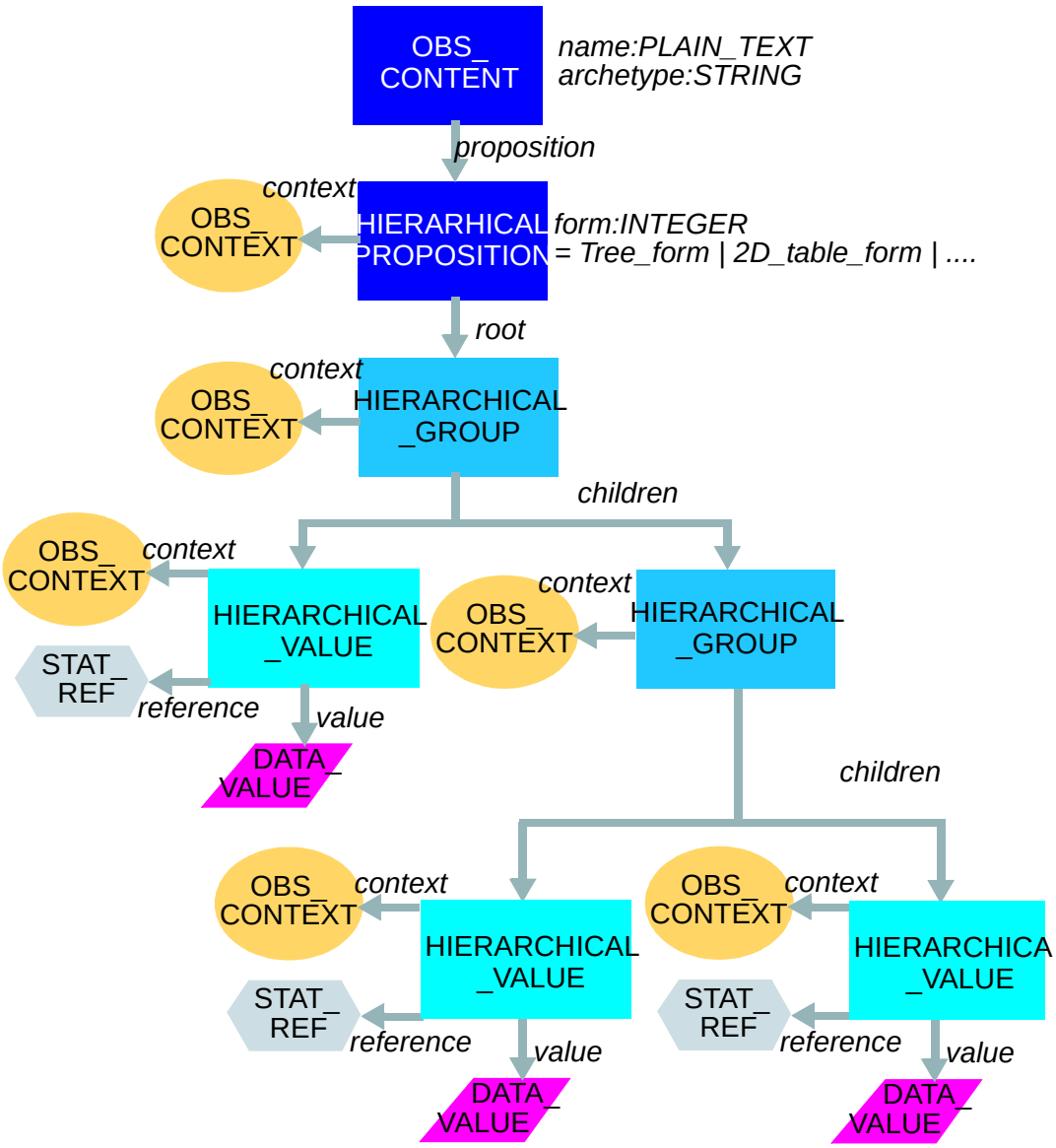


FIGURE 9 Object Structure of a Content Hierarchy

appear in component implementations of the GOM. It is thought that in the long term, it would be an advantage to have standardised functional interfaces to a standard selection of forms, allowing implementors of any GEHR software (whether it be a kernel component or an application) can safely make assumptions about how to interface to the underlying data. This approach does not prevent software from treating content in its basic form, i.e., hierarchy, or even in a different form than the specified original (e.g. treating a regular time series as a two-column table).

Note that the use of form here is not meant to satisfy requirements for presentation (e.g. recording preferred/original/intended presentation styles), which might be quite complex.

To Be Determined: presentation is currently seen to be mostly an application and archetype problem. It may be that an attribute is needed to record the name of e.g. an XML style sheet or similar in the EHR.

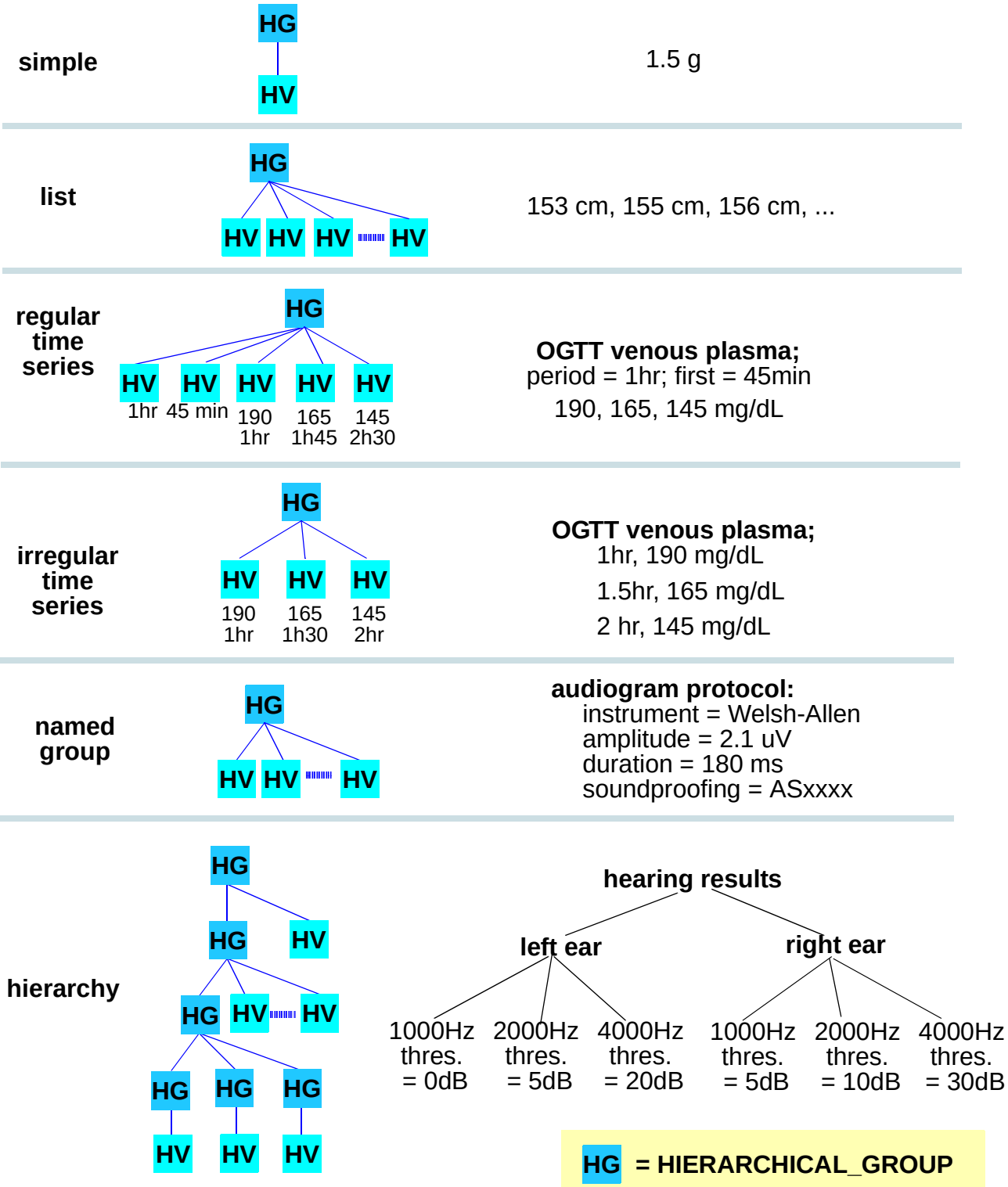
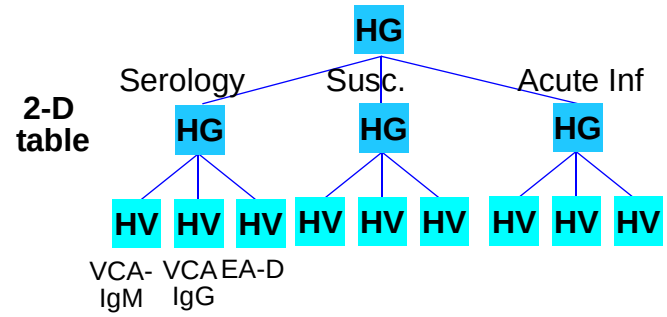


FIGURE 10 Archetypal Content structures

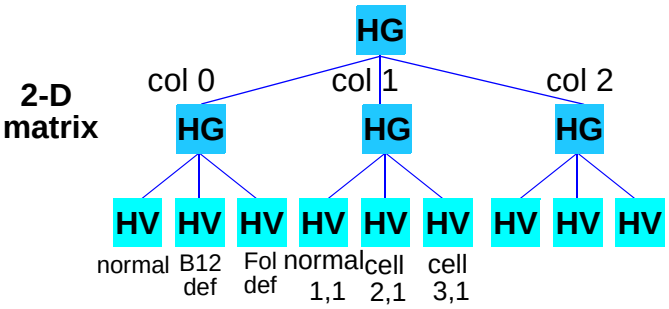
3.4.3.2 Qualifiers

It is quite common for a primary set of values to be used in conjunction with a secondary set, for instance, target reference values, or statistical mean values. Reference ranges are often complex tabular data, since they are often parameterised, e.g. by patient's height, weight, BP and so on. Such data is not strictly necessary in the record, since it can always be looked up, but it increases its usability,



Epstein-Barr virus serology

Serology	Susceptibility	Acute Infection	Prior Infection
VCA-IgM	Neg	Pos	Neg
VCA-IgG	Neg	Pos	Pos
EA-D	Neg	Var	Var
..	..	..	..



Lab Tests for B12 and/or Folate deficiency

Clinical	serum B12	serum folate	red cell folate
normal	normal	normal/low	normal
B12 def.	low	normal/high	low
Fol. def.	normal	low	low
both	low	low	low

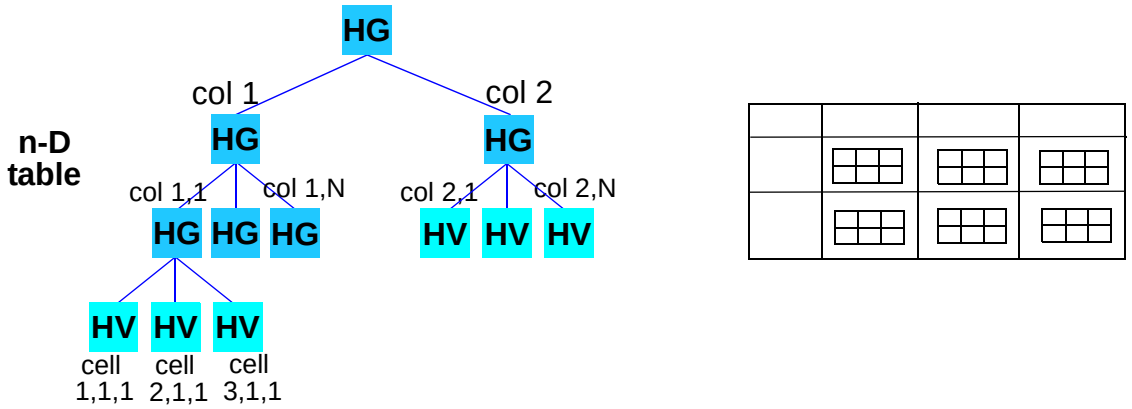


FIGURE 11 Archetypal Content Structures cont'd

since it is very convenient to have the *particular* reference relevant to the primary data on hand. Decision support systems can benefit from this; for example: Why did a clinician suggest a special diet for an infant? The answer may be clear from a comparison of current weight/height to weight reference ranges parameterised by age, height.

Reference data is modelled with the *reference* feature in the class `HIERARCHICAL_VALUE`, which is of type `STATISTICAL_REFERENCE` (see grey hexagons in FIGURE 9). The assumption here is that the form of (i.e. tree, list, table) of reference data is identical to that of the corresponding primary data.

To Be Determined: more sophisticated reference structures may be required in the future, also other kinds of qualifier data. Investigation required

3.4.4 CONTENT Sub-types

3.4.4.1 Definition Content

DEFINITION_CONTENT is used to record information with no context, such as reference or target values.

The name “definition” has been chosen to correspond to DEF mood in HL7 v3 USAM (2.4).

3.4.4.2 Predicate Content

Predicate content is information *about* someone, i.e. about the subject of the record. Accordingly, PREDICATE_CONTENT adds *subject*, and also *protocol* to DEFINITION_CONTENT. Protocol expresses how the information was arrived at (usually and experimental or decision-making protocol).

3.4.4.3 Observations

Req: clin:obs p31

Observation content is used to record objective observations. As described above, it is constructed from an OBSERVATION_CONTENT object. The OBSERVATION_CONTEXT class contains appropriate contextual information for observations, including *recorder*:STAFF_MEMBER and *dt_observed*:DATE_TIME.

Situations in which context values lower down the tree will be overridden include:

- Multiple clinicians recording values for a patient, most likely in a hospital monitoring situation. *Recorder* can be overridden in each case by the application software, so as to indicate the author of each and every value. Note that this model is not necessarily advocating such a practice, simply providing the data structures should it be required.
- Time series of values, e.g. blood glucose level, temperature. In this case the date/time of observation is set on each HIERARCHICAL_VALUE (or HIERARCHICAL_GROUP, if groups of values are being recorded). The resulting objects can be read by an application to construct a graphical trend.

3.4.4.4 Subjective Information

Req: clin:subj p31

Subjective content uses the same class structure as observations, allowing hierarchies of various flavours to be formed. The context data provided by SUBJECTIVE_CONTEXT, in addition to that from OBSERVATION_CONTEXT, is *provider*, *dt_provided*, and *certainty.* Usually these are only required on the root, but in some cases different certainties are required; differential diagnosis is such an example for information where the provider is the clinician; differing certainties (probabilities of correctness) are also not uncommon in patient-supplied information.

Context objects may be overridden on HIERARCHICAL_VALUE nodes in a subjective information tree, in, for example, a differential diagnosis, where different certainties are supplied.

3.4.4.5 Instructions

Req: clin:inst p32

Instructions are by their nature different from observations or statements, and in this model, INSTRUCTION_CONTENT objects are containers for the context of execution of the instruction - the *when*/*how*/*where* - while the details of the instruction - the *what* - is actually encoded as DEFINITION_CONTENT via the *action* relationship. Instruction content is used to record prescriptions, surgical procedures, therapies and any other action to be done to/for the patient.

Instructions have an implied state, which is often important to know, so that a determination of current, overdue and executing instructions can be made; for example, the list of medication type instructions in the "currently executing" state corresponds to what clinicians think of as "current medication". FIGURE 12 illustrates a likely state diagram for instruction content.

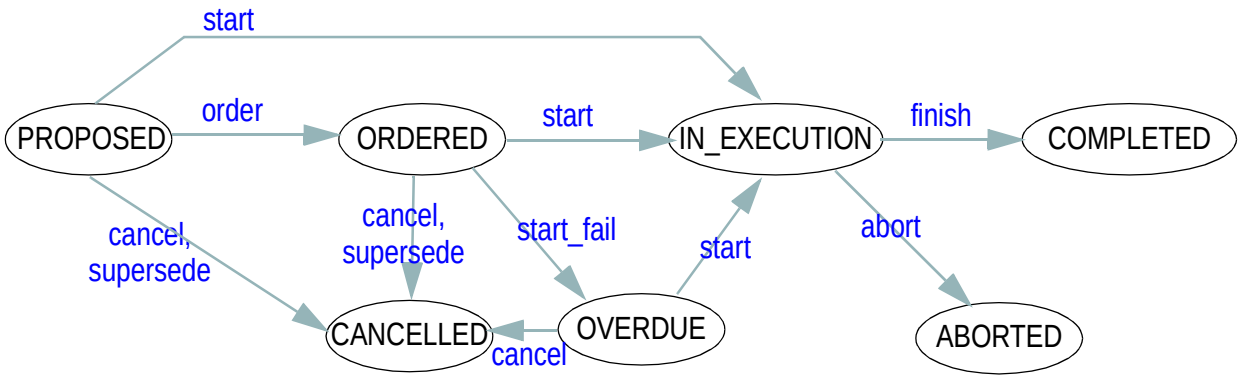


FIGURE 12 INSTRUCTION_CONTENT State Diagram

To Be Continued: this could be modelled completely in archetypes, with a special state variable archetype type.

3.4.4.6 Processes

"Process" in GEHR is a higher-level temporal concept which models a real-world goal-directed process. It is characterised by:

A goal: a statement of what the process intends to achieve, in the form of a `DEFINITION_CONTENT`.

Actions: a set of instructions to be performed to reach the goal.

Results: the set of results due to the actions.

Variance: a measure of variation between the desired goal and the actual results.

Status: the status of the overall process.

Intended period of execution: the time period over which it is hoped that the process will complete.

Actual period of execution: the time period over which the process actually ran.

Like the instruction content type, process content has an implied state, illustrated in FIGURE 13. Fewer states are required since the state of the process is essentially an abstracted view of the states of its constituent instructions.

3.4.4.7 Queries

To Be Continued: Results are LOCATORs; no content copying;

To Be Continued: Semantics of exchange - have to decide which transactions to take with transmitted query content.

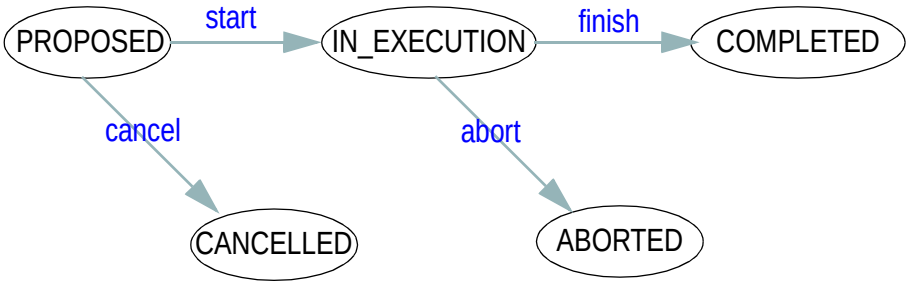


FIGURE 13 PROCESS_CONTENT State Diagram

3.5 DATA Cluster

In accordance with the Requirements, a number of data types are modelled in the GOM. These are illustrated in FIGURE 14; yellow classes are those which inherit from DATA_VALUE, allowing them to be used as content “data”.

Some DATA_VALUE class names commence with “GEHR_”, indicating that there is a clash with a native type, such as BOOLEAN or DATE. These classnames, while slightly annoying make it clear that an explicit GEHR model of the data type is being used.

3.5.1 GEHR_BOOLEAN Class

A Boolean class is provided for the purpose of representing truly boolean data, such as true/false or yes/no answers. For such data, it is important to devise the meanings (usually questions in subjective data) carefully, so thathe only allowed results are in fact true or false. There is a potential problem in dealing with various flavours of uncertainty with respect to boolean questions, with responses such as “not sure”, “unknown”, “not disclosed”, and so on. The approach recommended here is that if a true/false result is in theory knowable, it should be determined. If only uncertain answers are initially available, they should be stored in context attributes. Currently, SUBJECTIVE_CONTEXT provides a place to record certainty.

```
To Be Determined:it may be that context attributes need
     to  be  expanded  to  include  more  flavours  of
     “unknown” etc; see HL7’s approach:
     no-information (NI)
     answer-denied (?? GEHR addition)
          not-applicable (NA)
          applicable-but-unknown (UNK)
               not-asked (NASK)
               asked-but-unknown (ASKU)
                    temporarily-unavailable (NAV)
                    known-but-not-representable (OTH)
```

3.5.2 TEXT Cluster

Four primary text classes are included as follows:

PLAIN_TEXT: unformatted text, guaranteed to be comprehensible to any recipient.

Req: clin:data-text-plain p36

TERM_TEXT: text which arises from a term set, such as ICD10, ICPC, SNOMED or READ. This is used for coded terms. Since TERM_TEXT is a subtype of PLAIN_TEXT, it can be used in place of it, although this is not always the case; archetypes determine substitutability in context. TERM_TEXTs may have any number of qualifiers, from the same term set as the primary term; these can be used for laterality indication, for example.

Req: clin:data-text-term p36

PARAGRAPH_ITEM: one or more PLAIN_TEXTs forming a text section with which emphasis, style modifiers (e.g. font, colour etc), and a hypertext link may be associated. PARAGRAPH_ITEMs constitute pieces of text to be

Req: clin:data-text-link p36

used in the context of a paragraph, and allow a reasonable amount of formatting information to be recorded.

Req: clin:data-text p35

PARAGRAPH: a logical composite text value consisting of a series of PARAGRAPH_ITEMs, i.e. terms and plain text potentially with simple formatting and hypertext links, to form a larger tract of prose. PARAGRAPH is the standard way for constructing longer text items in summaries, reports and so on.

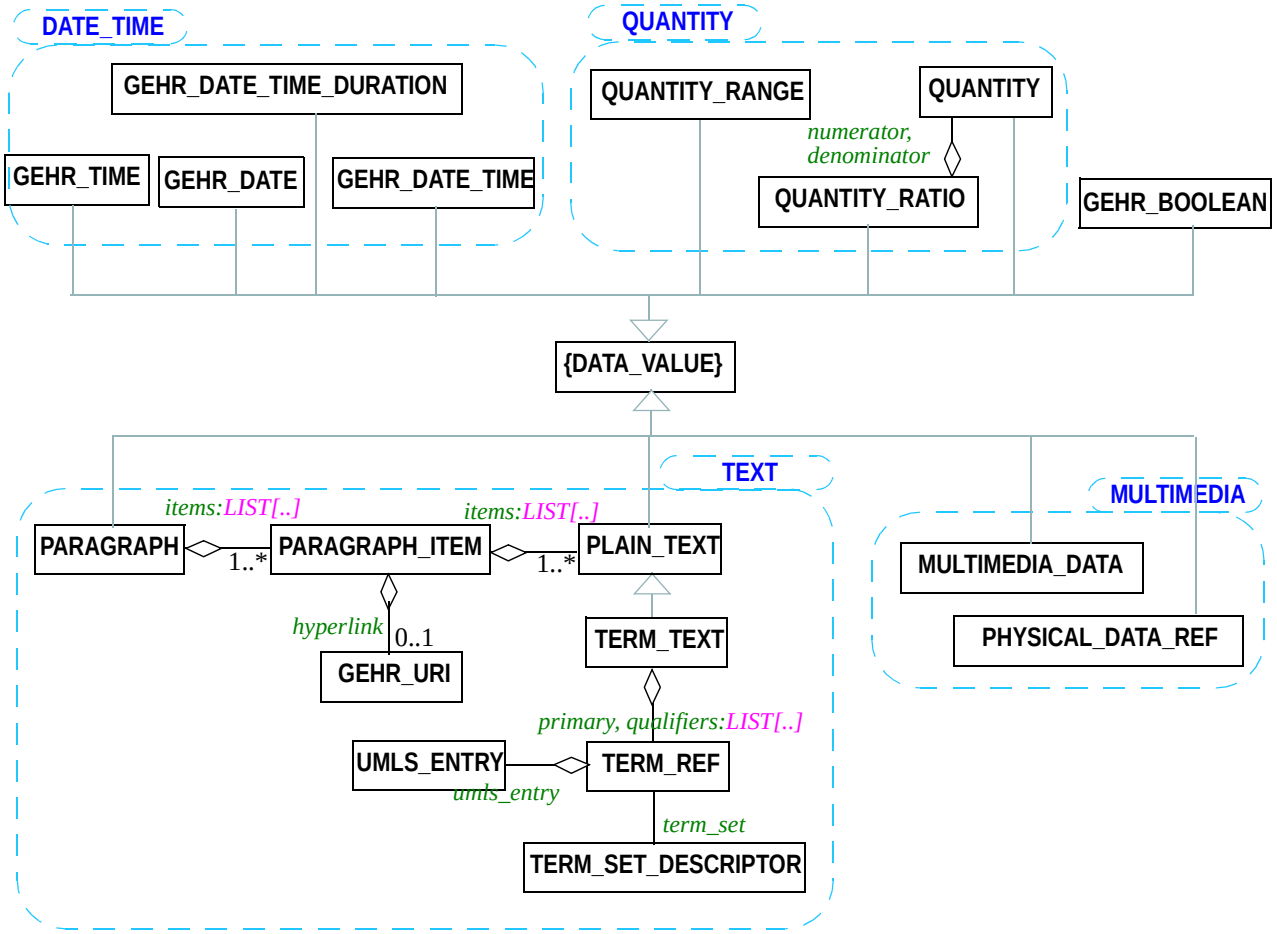


FIGURE 14 Data Cluster

Formatting and Hypertext

Formatting information needs to be recorded in the GEHR record, since readers of information may well consider that formatting (e.g. bolding etc) does contain information; doing otherwise would fail to satisfy the faithfulness of recording requirement. However, it must be recorded in such a way that a) recipients which have no/limited ways of actually using the formatting information can still read the text, and b) that formatting information is not stored in opaque fashion guaranteed to be readable by any recipient. It is for the second reason that straight XML is not allowed here (it can of course be used in a MULTIMEDIA_DATA item) - if it was, there is no guarantee that recipients of the text can process XML at all, or if they can, know the DTD; it would be difficult even to guarantee a minimim validity of a DTD. In any case, the overhead of DTDs for small pieces of text seems excessive.

If no formatting or hypertext is required, each PARAGRAPH_ITEM will be used for the purpose of representing sections of PLAIN_TEXT followed by TERM_TEXTs.

Hypertext links are URIs, including the GEHR_URI subtype; in this way both internal paths and internet URLs can be associated with text items. See section 3.6 below for details.

To create a piece of text larger than a word or term, a PARAGRAPH is used. Each item in a PARAGRAPH can be a PLAIN_TEXT or any descendant thereof. section FIGURE 15 illustrates the actual structure of a typical PARAGRAPH.

plain *term text* plain textpl a **intextplai** ntext *plaintext* plain
plaintextpla inte x tplainte xtplaintextplain *term text* textplaintext
plain textpl a intextplai ntext plain *term text* plaintex t pla

FIGURE 15 PARAGRAPH visual structure

UMLS Terms

see: www.nlm.nih.gov for UMLS

In the text cluster, TERM_TEXT inherits from PLAIN_TEXT, enabling free text items to be substituted by textual expansions of term set items. Where a TERM_TEXT is used, relevant term set information is recorded, via the classes TERM_REF, TERM_SET_DESC, and UMLS_ENTRY. The latter refers to a way of encoding terms developed at the National Library of Medicine in the United States, and consists of a meta-thesaurus, in which terms from any extant term set (such as READ, ICD, SNOMED, LOINC) can be encoded. UMLS entries are optional, but are extremely useful for decision support, and are encouraged. Currently, only the UMLS CUI (Concept Unique Identifier) is recorded, since the TERM_SET_DESC class already records the termset of origin, and the *expansion* attribute in the TERM_REF class records the intended string, i.e. the text equivalent of the UMLS SUI (String Identifier). If the needs of decision support mandate the inclusion of further UMLS attributes, the UMLS_ENTRY class provides a place for them.

FIGURE 16 shows the text cluster in detail.

Language and other Localisation Issues

At first sight, indicators for language, character set and territory (or “country”) seem to be missing from text classes in the GOM. In fact they are taken care of, but at the Transaction level, with the LOCALE class. All text items within a transaction are always understood as having the same language settings as the transaction itself. Questions naturally arise about whether this is a safe thing to do, for example:

- What about transactions which are sent to another language domain?
- What if parts of a transaction are included in another transaction as the result of a query?

For transmitted information there is never a problem: the transaction is the smallest atom of transmission, so there is no danger of text items being taken out of their localisation context.

The question of queries is also simply solved, if we refer back to the model of QUERY_CONTENT, in which it was stated that the result of any query is a set of LOCATORs, i.e. record paths. Remembering that all information in a transaction is

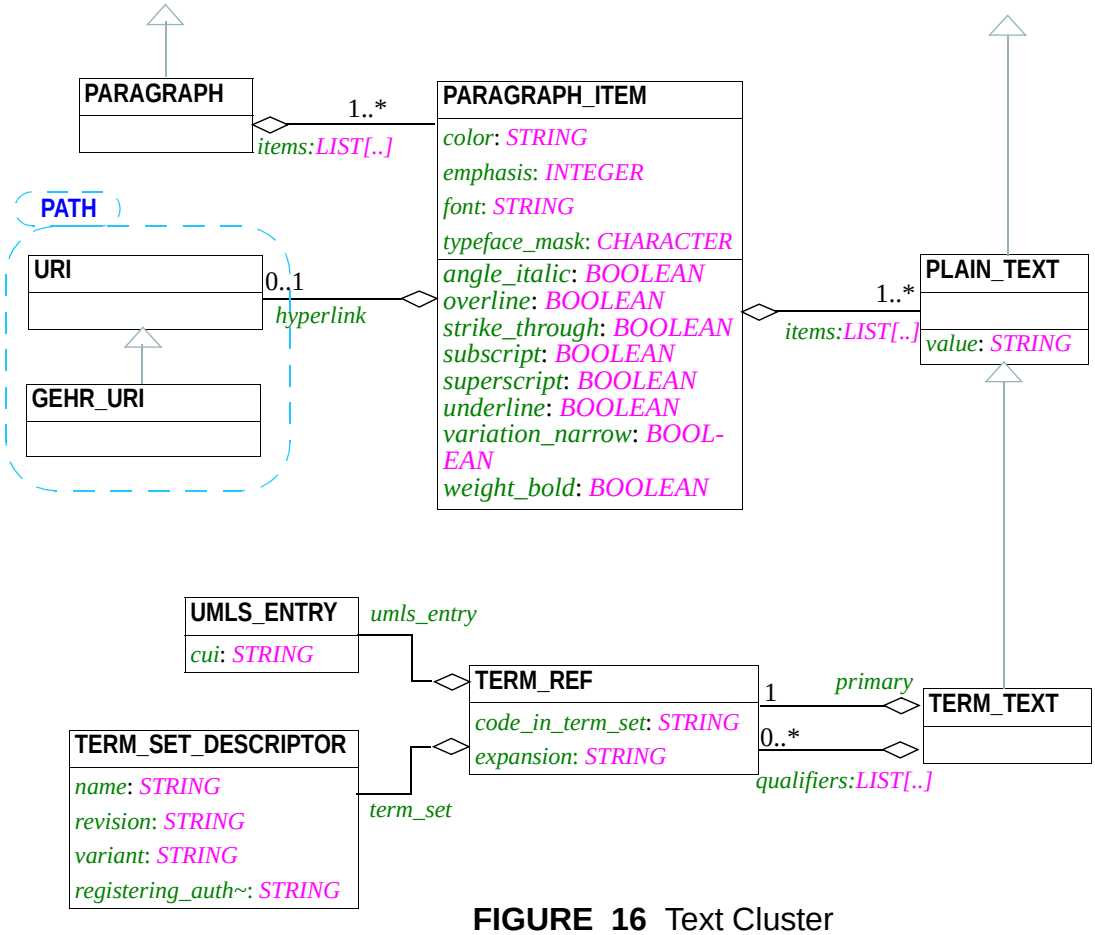


FIGURE 16 Text Cluster

version controlled, and that LOCATORs contain within them a transaction (i.e. version) identifier, then all query results are immutable in time, since (by definition) they refer to information items *inside transaction versions*, which are never changed.

Thus query results coming from different transactions, including those received from other language locales, may be visually integrated, but underlying information objects are never mixed together.

3.5.3 QUANTITY Cluster

Quantities in GEHR are modelled in a way which allows applications to store any quantity in the record, but does not attempt to provide full domain semantics for units, comarisons and so on. Three clinical quantity types are catered for in the QUANTITY cluster:

Req: clin:data-qty-num p36

QUANTITY: models any quantity including coded units (i.e. each unit is a TERM_TEXT), which may be mixed, and from any system. QUANTITY.*value* is of type NUMERIC, enabling any INTEGER, REAL or DOUBLE value to be used. The attribute QUANTITY.*precision* allows the number of digits of precision of the value to be recorded. Here, precision is understood as a characteristic of a number, not to be confused with

"accuracy" which is an attribute of a meansuring device, and should be recorded in the protocol for observations made with the device.

`QUANTITY_RANGE`: models ranges of quantities, such as target ranges for e.g. weight.

Req: clin:data-qty-rng p36

`QUANTITY_RATIO`: models a ratio of quantities as used for recording dosages (e.g. 5 mg / 100 ml), and drug amounts based on body wieight (e.g. 1 tablet / 10 kg).

Req: clin:data-qty-rat p36

Units in `QUANTITYs` and `QUANTITY_RANGEs` are implemented as STRINGs, but are assumed to be formatted according to the Unified Code for Units of Measure (UCUM) proposal, developed by Gunther Schadow and Clement J. McDonald of The Regenstrief Institute For Health Care, Indianapolis.

see: http://aurora.rg.iupui.edu/UCUM

This proposal is comprehensive, covering all useful unit systems, including SI, various imperial, customary mesaures, and some obscure measures, as well as clinically specific additions. As far as the GOM is concerned, units are simply expressed in strings such as:

`"kg/m^2", "m.s^-1", "km/h", "mm[Hg]"`

and so on. Metric prefixes, meaning-changing textual suffixes (e.g. `"[Hg]"` in `"mm[Hg]"`) and non-meaning-changing annotations (e.g. `"kg {total}"`) are recognised.

3.5.4 DATE_TIME Cluster

The `DATE_TIME` cluster includes three absolute date/time concepts: `GEHR_DATE`, `GEHR_TIME`, `GEHR_DATE_TIME`, and a relative concept: `GEHR_DATE_TIME_DURATION`.

Req: clin:data-dt p36

To Be Implemented:Incomplete and approximate date/times will soon be added. Approximate date/times have a certainty associated with each part of a data or time; error durations can also be associated, e.g. "plus or minus a week".

3.6 PATH Cluster

In the GEHR record, any item can be specified with a text path. Such paths are used for all cross references in the record, and also as the results of queries. The Path cluster is illustrated in FIGURE 17.

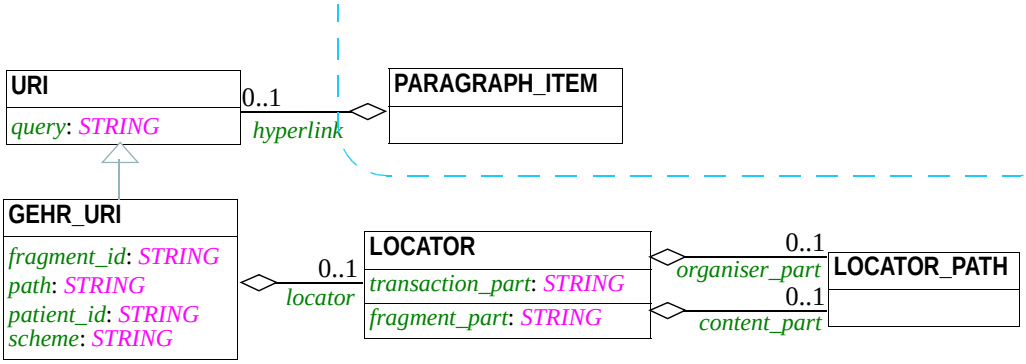


FIGURE 17 Path Cluster

The classes may be understood as follows:

PATH: a PATH object defines the path to an item in a GEHR record from some starting point. In the above diagram, PATHs are used to describe the path inside an ORGANISER tree, and also inside a content hierarchy.

LOCATOR: this is used to describe an absolute path to any item in a record, starting from the top of the record. See below.

GEHR_URI: A GEHR_URI is simply a normal URI which has the scheme name "gehr". Because GEHR_URI is a descendant of URI, it can be used as a hyperlink in a PARAGRAPH_ITEM. Normal (internet) and GEHR URIs are thus substitutable in this context. See below for more detail.

Locator Syntax

Locators are constructed according to the model:

```
transaction_part '#' [ organiser_part [content_part ] ]
```

where the sections are as follows:

transaction_part: transaction identifier, in the form $ehr_source '@' $hcp '@' $dt_committed. Here, $ehr_source is the GEHR unique identifier for the EHR source, $hcp is the name of the health care professional, and $dt_committed is the timestamp of the transaction. These three items are deemed to be sufficient to uniquely identify any transaction.

organiser_part: series of organisers, in the form '/' $organiser '/' $organiser '/' ..., which specify a particular node in the organiser hierarchy. '/' characters which appear in organisers are quoted in the Unix style, with a '\'.

content_part: series of content node names, in the form $name '|' $name '|' $name ..., which specify a particular content node in a content item. The name series is constructed according to the logical form of the item, rather than being strictly hierarchical. For most forms, the two will be the same, but for tables, they may be different; for example, while a cell in a 2-dimensional matrix called B12/Folate serology may actually appear in the physical path "B12"/"Folate serology"|"serum B12"|"B12 deficiency", it will be specified with the logical path "B12"/"Folate serology"|"B12 deficiency"|"serum B12".

This syntax allows locators to be partially specified, from as minimal as possible, indicating only a transaction, to a fully specified content leaf node. Typical examples are:

```
Aus-Nambour-BH-1@Dr Stephen Smith@03-05-1997
23:04:55#/"Diabetes"/"Subjective"
```

(everything under the "subjective" heading of "diabetes" in a transaction committed by Dr Stephen Smith at the primary EHR system at Nambour Base Hospital)

```
Fr-Nantes-HG-B@Cecile Guillou@01-04-1998 09:31:22
```

(a complete transaction committed by Cecile Guillou to the "B" EHR system at Nantes general hospital in France)

```
Uk-London-Bart-A@Dr Peter Cole@13-12-1990
09:22:00#/"Hearing"/"test results"/"audiology"|"hearing
threshold"|"left ear"|"1000Hz threshold"
```

(the 1000Hz threshold value of an audiogram test, recorded under the headings “Hearing/test results”, in a transaction committed at EHR source ‘A’ at St Bartholomew’s Hospital, London).

Locators are used for all cross-referencing in the GEHR record, ensuring readability by humans, as well as validity when extracts are transmitted elsewhere: even if the target of a path is not present, the path can be used to locate the missing item on demand.

GEHR URIs

A GEHR locator can be used as part of the GEHR version of a Uniform Resource Identifier (URI), which is defined by the WWW consortium (W3C) in RFC 2936 http://www.ietf.org/rfc/rfc2396.txt. This requires a “GEHR” scheme in the URI system. GEHR URIs would also require a patient identifier section. The form of a GEHR URI is thus:

```
gehr://some.ehr.domain/<patient-id>::<locator>
```

Because of the transmission semantics of transactions, i.e. that no smaller fragment of a record can be transmitted to another location, the URI fragment separator, ‘#’ has been used between transactions and lower-level items in the preceding locator specification. In URI terms, a “fragment” is any reference into a document which is intended as the object of a retrieval action; in other words, fragment identifiers are processed post-retrieval. This accords with the idea of the GEHR transaction as a transmittable item.

The definition of the `<patient-id>` section is as follows:

`To Be Determined:`

The special patient identifier `self` can also be used.

3.7 ARCHETYPE Cluster

GEHR archetypes are described in detail in the document The GEHR Archetype System. The GOM itself defines only very limited semantics for archetypes, as illustrated in FIGURE 18.

Req: clin:arch p34, clin:arch-extent p34

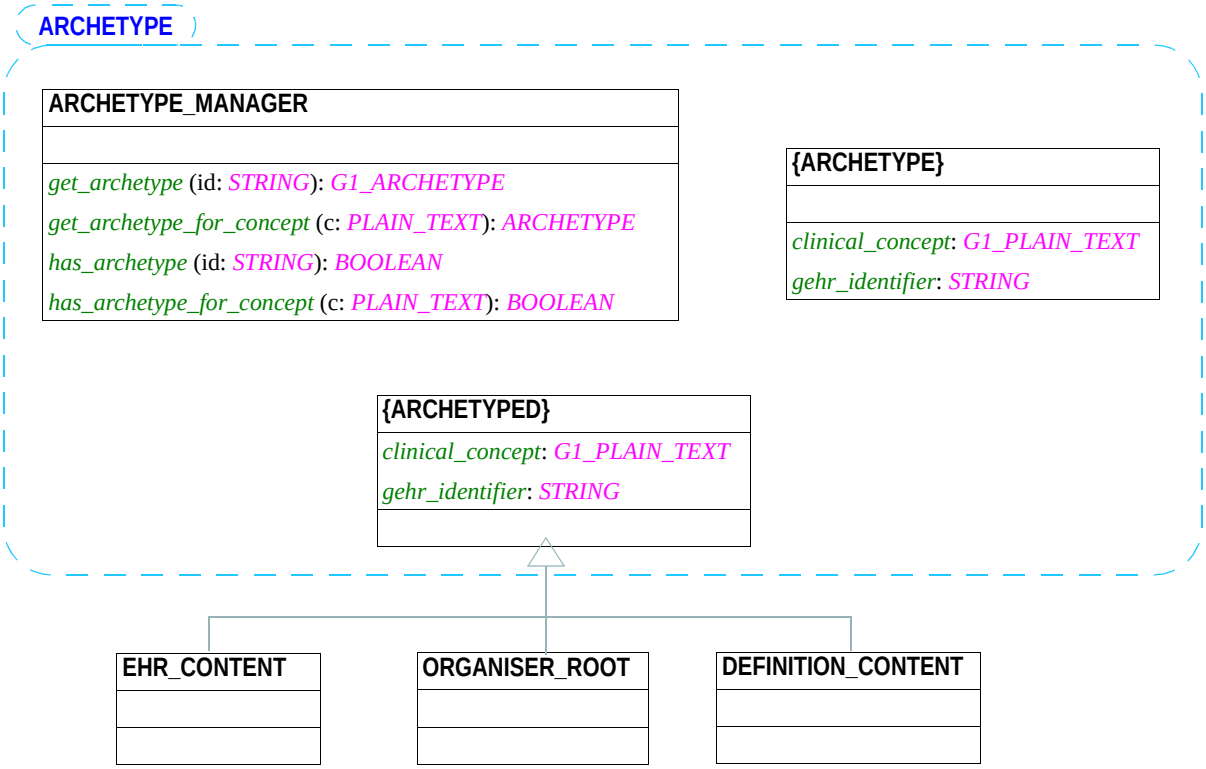


FIGURE 18 Archetype Cluster

In this model, an archetype manager provides an interface from which the existence of archetypes can be determined, and via which valid archetypes can be retrieved. Note that “archetypes” in this sense are progrmming language objects, not the external document form.

3.8 Semantics of Exchange

The ability to preserve not only the semantics of the EHR but also its medico-legal status and generic layout are ambitious requirements of GEHR. The architecture proposed in this document aims to address these requirements. Specific areas need to be addressed based on practical experience.

3.8.1 Content Integrity

To Be Determined:

3.8.2 Terms

To Be Determined:

3.8.3 Local Termsets

To Be Determined:

3.8.4 Cross-references

To Be Determined:

3.8.5 Bulky Data

To Be Determined:

3.8.6 Local Class Extensions

To Be Determined:

4 Clinical Scenarios

This section shows how the architecture can be used to create information structures corresponding to the clinical scenarios described in the requirements.

To Be Continued:This section will eventually contain instance diagrams showing how the record is modified under these scenarios, in particular, the archetype structures driving concrete record structures.

4.1 Basic Content Types

4.1.1 Weight

FIGURE 19 illustrates a typical example of observation content: a patient's weight, along with a protocol and reference range. These last two are probably overkill for what is essentially a single numerical value, but it illustrates the principle for more complex data.

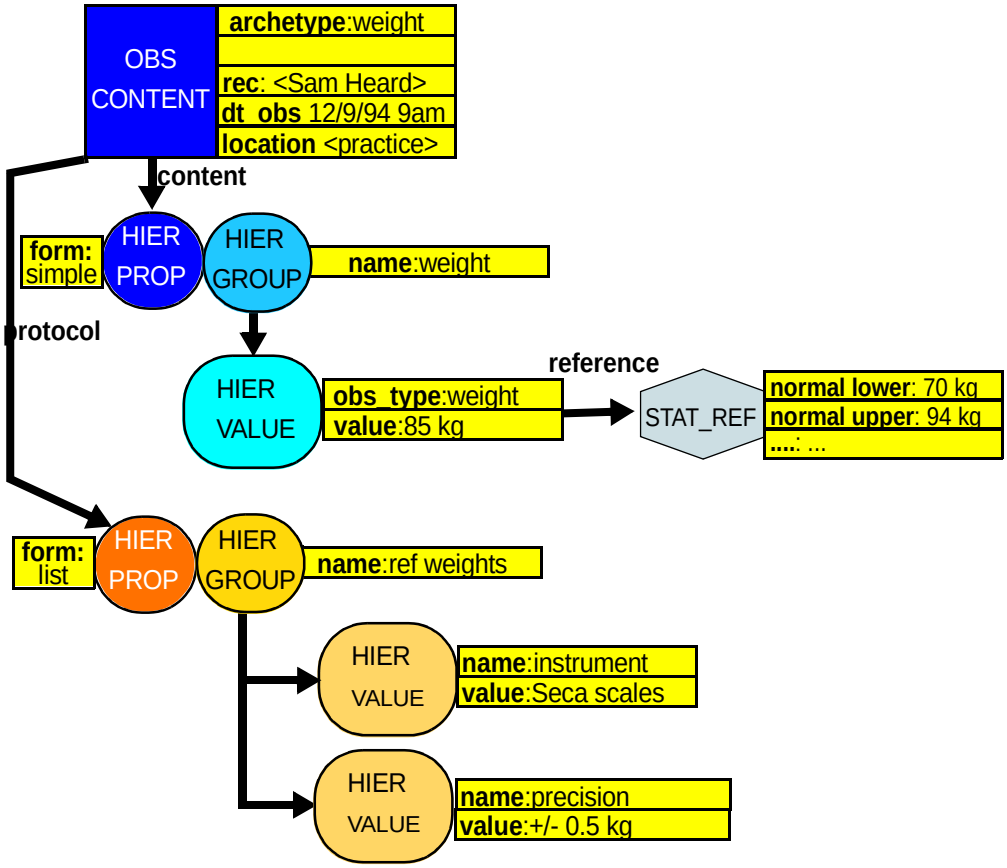


FIGURE 19 Phenomenon: weight with reference range and protocol

4.1.2 Blood Pressure

A third example, that of blood pressure is included for completeness, and because it has presented so many problems for representation in the past, being essentially a binary sample from a range.

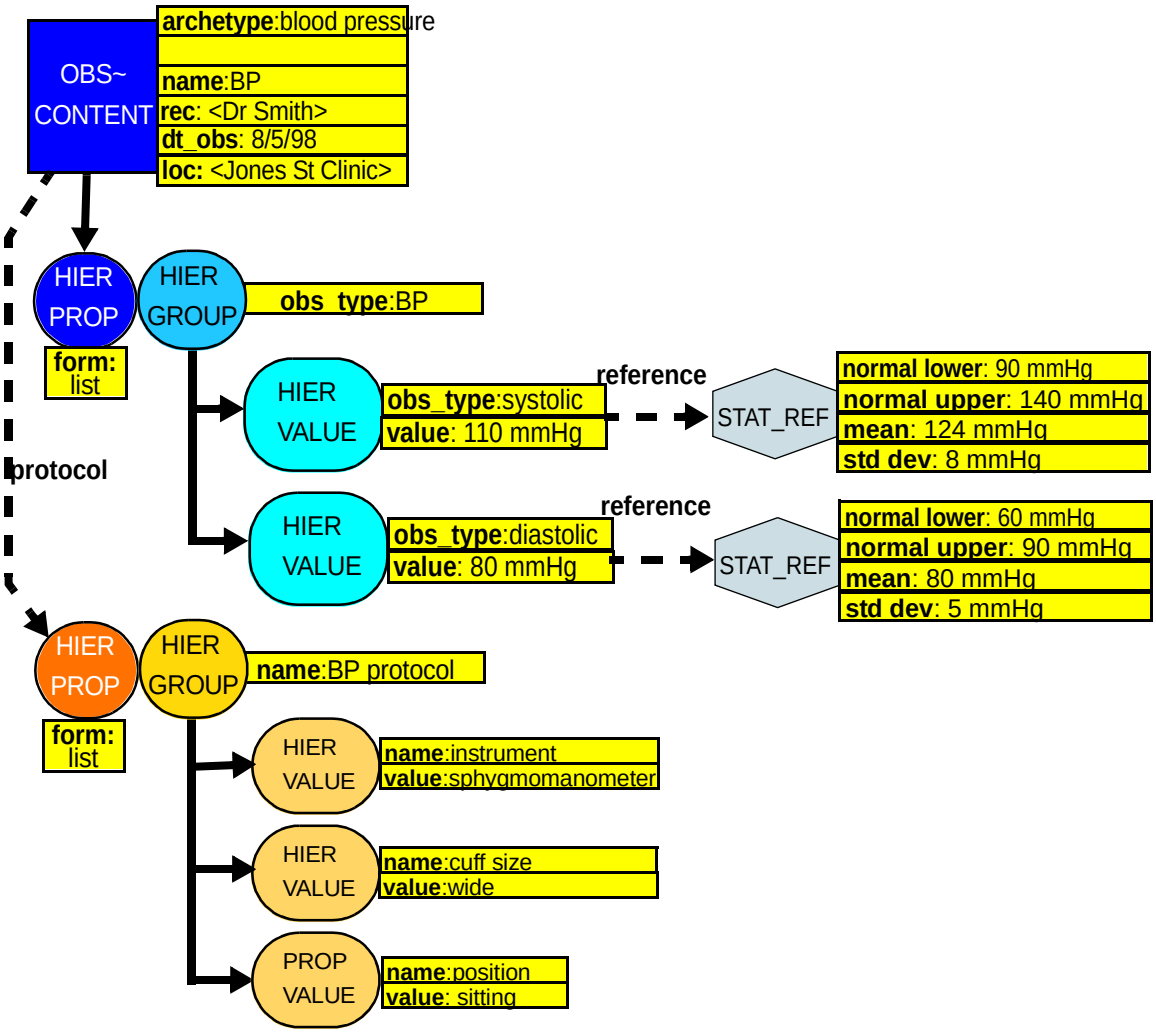


FIGURE 20 Phenomenon: Blood Pressure with reference

4.1.3 Audiology

The audiology example in FIGURE 21 illustrates observation data of the same general form, but higher complexity. In this case, the protocol and reference data are quite likely to be required by the clinician.

4.1.4 Family History

To Be Continued:

4.1.5 Lifestyle

A typical subjective structure is illustrated in FIGURE 22, illustrating two lifestyle items, both provided by the patient to the doctor at his practice. The probability of correctness in each case is assessed by the clinician, not the patient, although it is usually based on what the patient says. (This must be the case, since the clinician must be able to have the final word, especially in cases where the provider is deficient but does not recognise it).

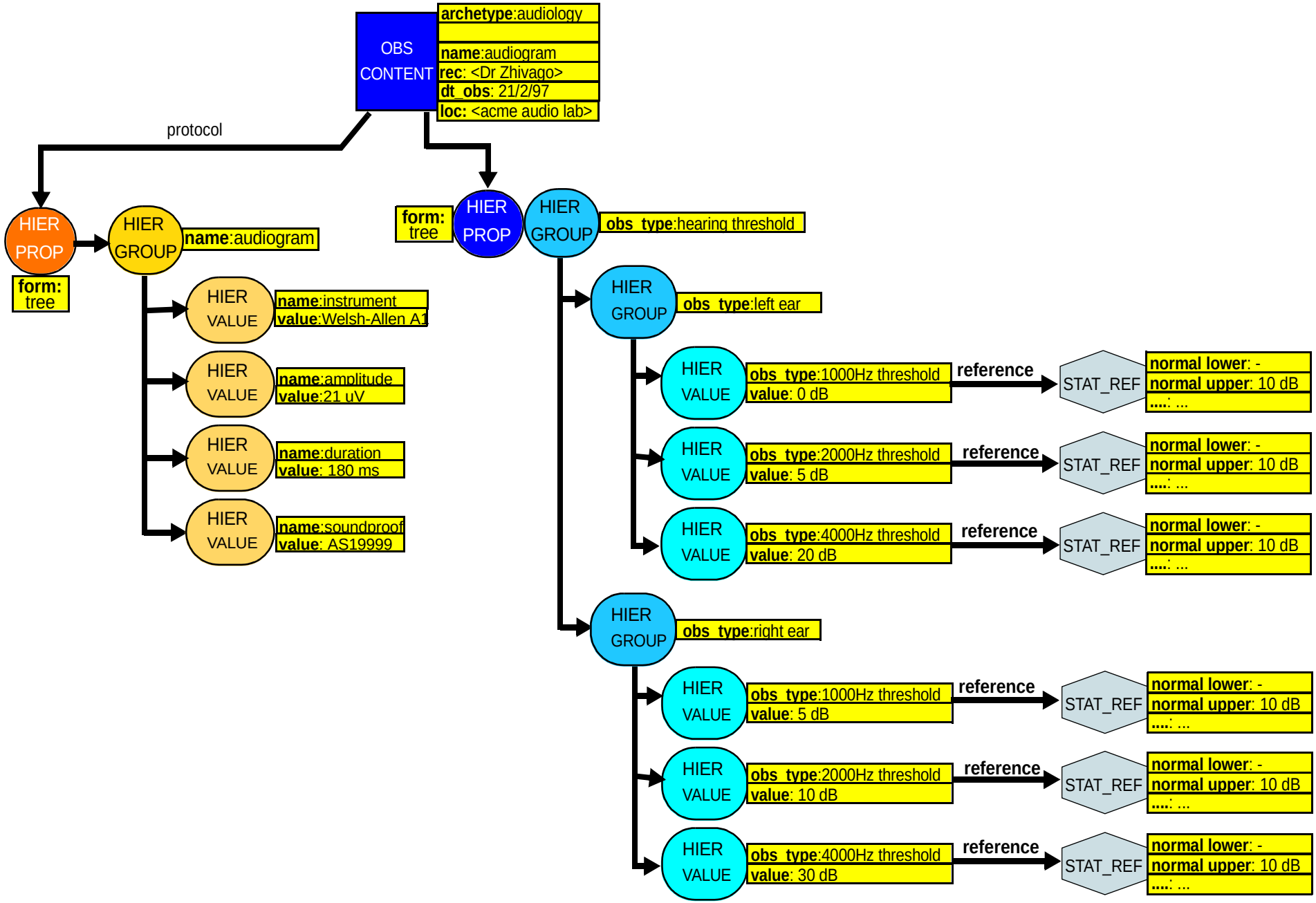


FIGURE 21 Phenomenon: audiology test results with reference

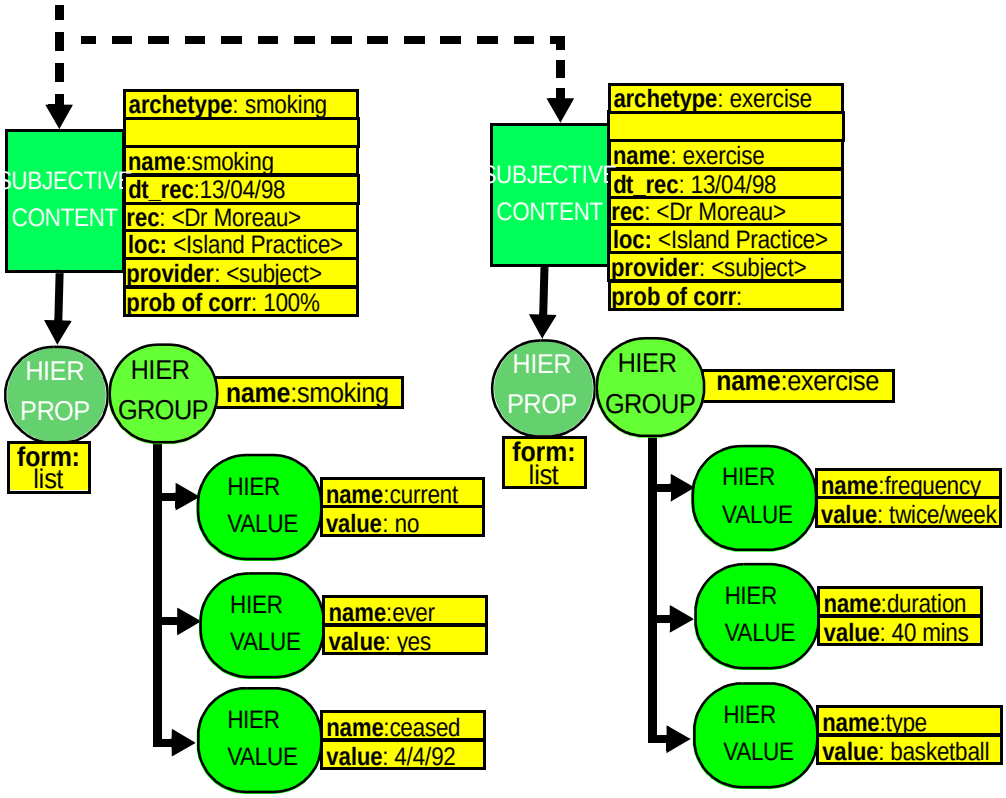


FIGURE 22 Subjective Information: Lifestyle

4.1.6 Differential Diagnosis

A second example (FIGURE 23) shows a differential diagnosis, which illustrates not only the structure of a diagnosis, but also the differential certainties allocated by the clinician.

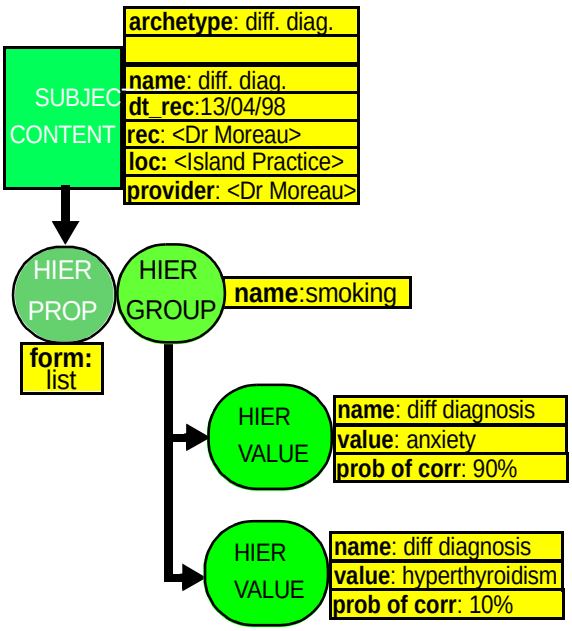


FIGURE 23 Subjective Information: differential diagnosis

4.1.7 PAP Smear

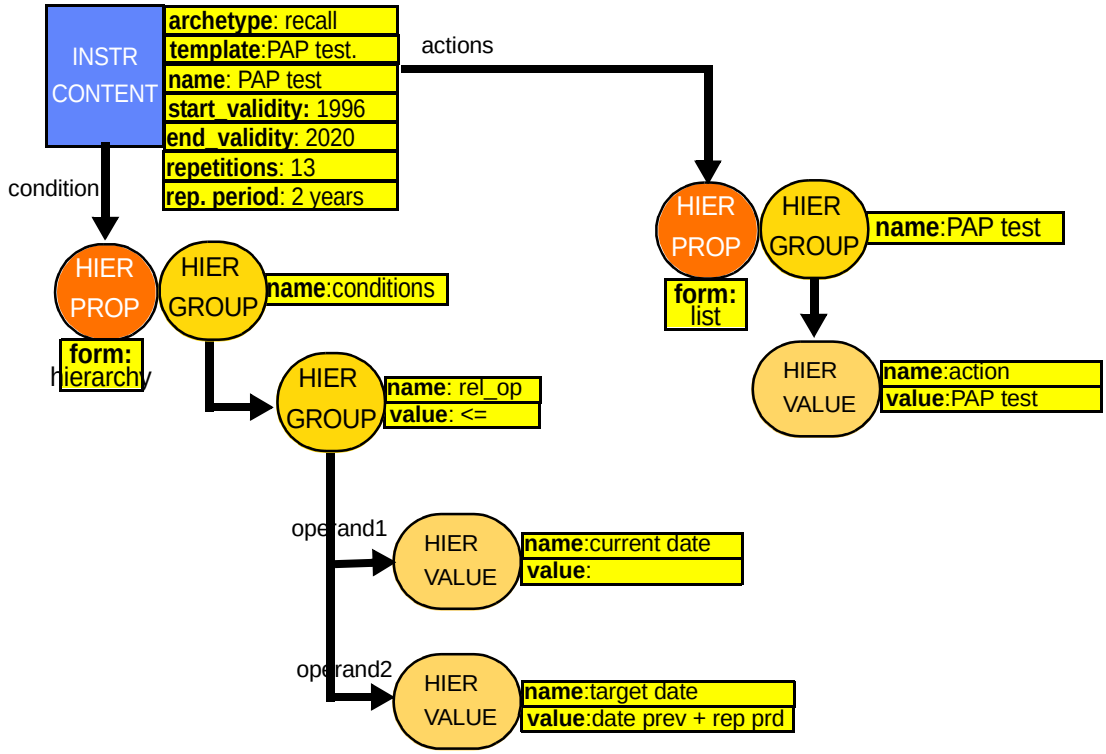


FIGURE 24 Instruction: recall (PAP)

4.1.8 Medication Order and Prescription

"Prescription" is a deceptively simple concept in the real world, whose implementation in information systems can be confusing if it is not properly understood. At the outset, what consumers typically call a "prescription" needs to be understood as by clinicians: *as a communication of a series of medication orders to a filler.*

Medication orders can be modelled using `INSTRUCTION_CONTENT` objects, which provide all the necessary protocol and time-based administration information, as well as drug or therapy details. FIGURE 25 illustrates an `INSTRUCTION_CONTENT` used to model an antibiotic prescription. As with all other GEHR content, it is an archetype which determines the particular structure and allowed content values

So what does a prescription look like in GEHR? Let us firstly define more closely the meaning of the term as used in GEHR:

- A prescription is a communication from a clinician to a filler of medication orders, e.g. a pharmacy. It is not in itself a *clinical* statement (such as the list of medications for a patient in a care plan), but rather a convenient list of items expected to be available at a pharmacy.
- There may be more than one prescription for a care plan or other clinical plan which generates medication orders. This may be because:
  - Not all medication orders can be filled by the same pharmacy (e.g. some of the drugs are available at a specialist or hospital pharmacy only); some orders may be filled elsewhere entirely, such as a natural therapies shop. Some orders may not be for drugs as such, but for therapies.

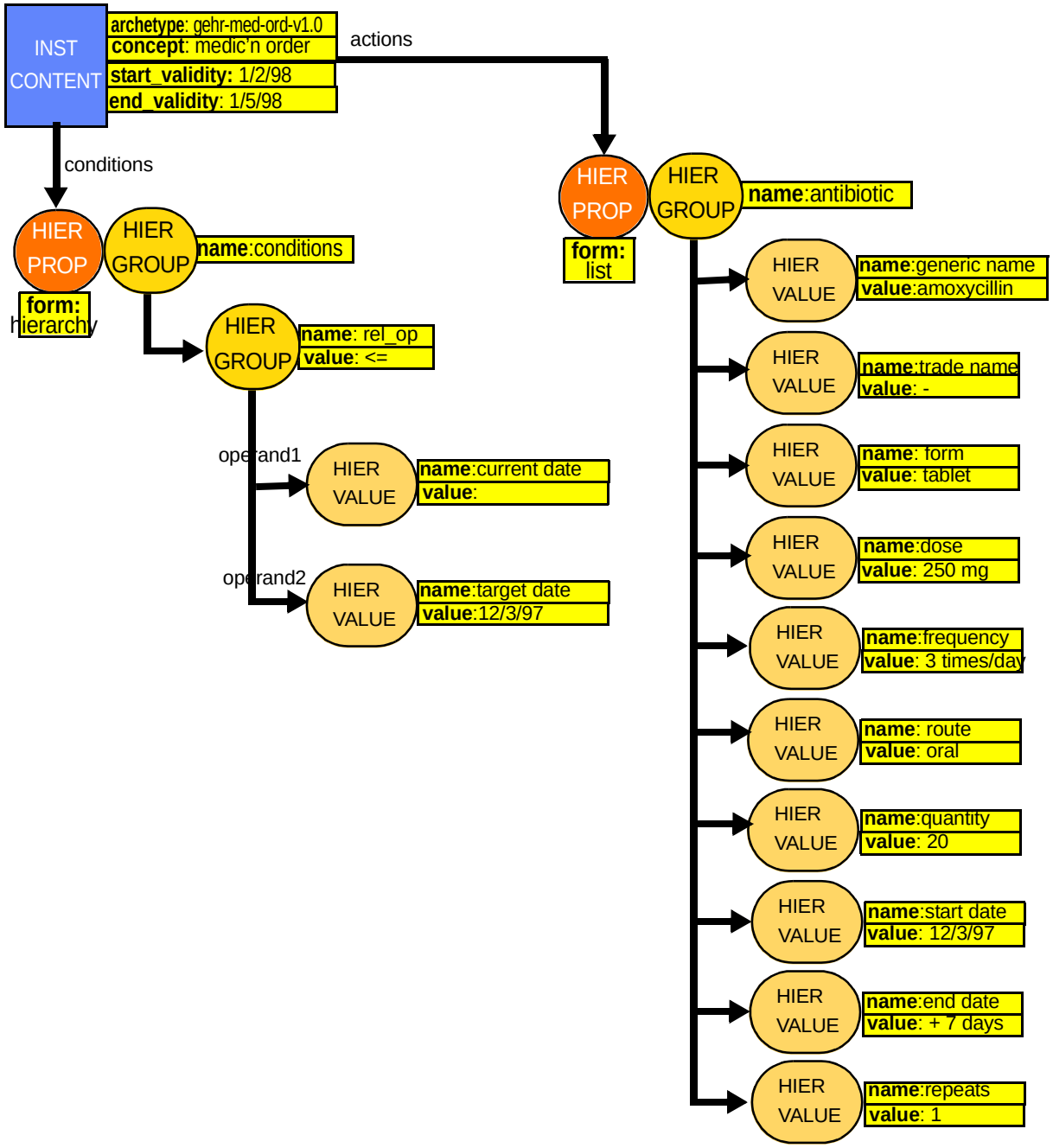


FIGURE 25 Instruction Structure: prescription

- There may be legislation preventing more than a certain number of drugs to be entered on one prescription.
- The clinician may write multiple prescriptions to become active on different dates.

- Since the fillers of medication orders will not in general be EHR sources, the prescription as a communication does not need to be an EHR extract of any kind. If in electronic form, it should be a message to the appropriate system used at pharmacies (an HL7 message may be required, for example). Therefore, the prescription message does not need to obey the rules of EHR extracts, merging of versioned transactions, etc.

- Despite all of the above, there does need to be some way to link the medication orders recorded in the EHR with a prescription.

Prescriptions are typically handled in today's clinics and hospitals by a special prescribing software package, which is capable of recording the details of medication orders, and then of sending the result to a printer or to another electronic system (such as in the hospital pharmacy). In a GEHR world, these same packages would continue to function with two differences:

- The medication order data would be obtained from the EHR, via the GEHR kernel.
- Medication orders in the EHR are marked with prescription identification information, allowing users of the EHR to know that a prescription has been issued. Such information could be used in a prescription or order management system to determine the status of a particular prescription.

4.2 Higher Clinical Concepts

4.2.1 Problem

Problems are usually understood in medicine as clinically assessed conditions of a medium- to long-term duration, including such things as asthma or fractured femur, but not usually short-lived illnesses such as common cold. In normal clinical practice, the information relating to a problem includes subjective information from the patient, observations, diagnoses or assessments, reports, and a treatment or management plan (modelled using instructions and processes).

In terms of GEHR records, the (technical) problem is how to link the relevant items together under a banner such as "asthma" or "diabetes", and also to record other information which is not an observation of some kind.

Because clinical problems are by their nature "ongoing", it is appropriate to use persistent transactions for them. References to other information in the record can be included in a problem transaction by adding `QUERY_CONTENTs` to it as well as `LOCATORs` to specific items. Descriptive content can be added to the transaction in the form of `PARAGRAPHs` and `MULTIMEDIA` content.

4.2.2 Issue

Some clinicians, particularly those in allied health, use the term *issue* to refer to a problem as subjectively experienced by the patient, such as "can't climb stairs" or "unable to leave the house". Normally an issue relates to one or more underlying problems, i.e. real clinical phenomena diagnosed by objective clinical processes. However, issues are important for patients, and by extension for carers, since they are likely to correspond to the treatment priorities of the patient.

In terms of data in the EHR, an issue is likely to correspond to a number of objective and subjective items, and potentially instructions and processes, probably with extra authored content as well. The choice of which items constitute an issue is up to the patient (or perhaps a relative or guardian thereof), and is therefore likely to be useful in systems where both patients and allied health professionals can enter data.

A reasonable way to model an issue in a GEHR record appears to be the same as for problems: create a persistent transaction for each issue, and to add one or more of the following types of information to it:

- `QUERY_CONTENT` objects whose results include all objects in the record matching a query filter, for example, "all blood glucose levels taken within the last 30 days". Remembering

that the results in a QUERY_CONTENT are GEHR LOCATORs to other content, the query thus provides a way to automatically include items of interest to the patient view of the issue.

- Manually added LOCATORs, referring to specifically chosen items in the record. These would be used to include specific past items which the patient decides in retrospect are germane to the issue, as well as new items as they occur. For instance, during the monitoring of an issue called “migraines” (remembering that what a patient calls “migraines” might be called something else by clinicians), the patient might record their own observations of headaches in event transactions, and then add LOCATORs to the issue transaction referring to those headaches they thought were “migraines” (as opposed to, for example, allergy- or sinus-induced headaches).
- Manually added descriptive content, in the form of text (PARAGRAPHs), graphics and sound (MULTIMEDIA).

All of the above kinds of information would be added to the issue transaction by creating new versions of it, as is done with all persistent transactions. Thus, in a similar manner as for problems being monitored by clinicians caring for the patient, the patient builds up his/her own “story” of their own health problems in persistent transactions which are available to clinicians elsewhere in the health system.

It should be noted that for patient-added data, the audit information of recorder, provider, and committer will always be the patient (or other relevant person), ensuring that even data considered by the patient to be objective, and as such recorded as OBSERVATION_CONTENT, will not be confused with data added by health care professionals, which may be considered by them to be less fallible. (Of course, carers from different healthcare cultures such as western medicine and naturopathy may regard each others’ observations in the record as being more or less fallible....).

4.2.3 Episode

Episode is one of the problematic terms in clinical medicine: it can be used to refer to e.g. a single hospital admission, a flare-up of a chronic condition such as gout, or a set of interactions between the patient and one health management entity (which might cover more than one health care provider).

Fortunately, this does not matter too much in the GEHR record. A similar approach as for problems and issues can be used, in that a transaction can be constructed containing references to other transactions and items considered to be part of the episode; it is up to the particular users to define what an episode is.

However, the choice of whether to use a persistent or event transaction may not be so obvious. If episodes are created after the fact (i.e. after all information pertaining to the episode has been added to the record), and if they are not considered to be long-term useful items of information (e.g. they may simply be used by billing or other applications to create a report, and then not used), an event transaction may be more appropriate. However, if episode transactions themselves are to be edited numerous times, or are considered long-term important information, persistent transactions may be more appropriate. The disadvantage here is the obscuring of truly persistent information such as “current medication” by numerous.

To Be Determined: maybe a new kind of transaction is needed - i.e. one that acts like a persistent transaction, but is not mixed in with the importance persistent transactions. Perhaps a “view” transaction?

4.3 Basic Scenarios

4.3.1 New Patient

To Be Determined:

4.3.2 Birth

To Be Determined:

4.3.3 Death

To Be Determined:

4.4 Hospital Scenarios

4.4.1 Admission

To Be Determined:

4.4.2 Contact

To Be Determined:

4.4.3 Surgical Procedures

To Be Determined:

4.4.4 Intensive Care

To Be Determined: It is this scenario that has led to the optional inclusion of the *recorder* with all items within a transaction.

4.4.5 Out Patients

To Be Continued:

4.5 Pathology Information

To Be Continued:

4.6 Clinical Processes

The GOM allows for such processes to be identified and monitored using the links in PROP_XXX classes.

To Be Continued:

4.7 Problem-oriented Medicine

A contact event transaction with root organisers Problem and sub-organisers SOAP would be chosen by clinicians taking this approach. An archetype for this type of contact record will be developed by the GEHR foundation.

To Be Continued:

4.8 Decision Support

To Be Continued:

4.9 Population Medicine

To Be Continued:

A References

A.1 GEHR Project

1. GEHR Project - *Deliverable 4: GEHR Requirements for Clinical Comprehensiveness*
GEHR Project 1992

2. GEHR Project - *Deliverable 7: Clinical Functional Specifications*
GEHR Project 1993

3. GEHR Project - *Deliverable 8: Ethical and legal Requirements of GEHR Architecture and Systems*
GEHR Project 1994

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

A.2 CEN

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

A.3 HL7

6. HL7 version 3 deliverable: *The Unified Service Action Model : Documentation for the clinical area of the HL7 Reference Information Model*. (Revision 2.4+)
Schadow G. Russler D., Mead C., Case J., McDonald C.

7. HL7 version 3 deliverable: *Version 3 Data Types*. (DRAFT Revision 1.0).
Schadow G., Biron P.

A.4 OMG

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

9. CORBAmed document: *Clinical Observations Access Service*. (Jan 2000)
3M, Care Data Systems, CareFlow/Net, HBO & C, LANL, and others.

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

A.5 Software Engineering

11. Meyer, Bertrand - *Object-oriented Software Construction*, 2nd Ed.
Prentice Hall 1997

12. Walden, Kim and Nerson, Jean-Marc - *Seamless Object-oriented Software Architecture*.
Prentice Hall 1994

13. Gamma E., Helm R., Johnson R., and Vlissides, J. - *Design patterns of Reusable Object-oriented Software*
Addison-Wesley 1995

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

15. Martin Fowler with Kendall Scott - *UML Distilled (2nd Ed.)*
Addison Wesley Longman 2000

END OF DOCUMENT